Moussaif Hermes-Fugu Fusion HyperSkills KB

500-page technical knowledge base for AGI-oriented hyperskills, hyperloops, learned orchestration, OpenRouter fusion routing, NVIDIA/NIM deployment, MCD governance, vascular compute, falsification courts, and self-reconstruction memory.

Zouhair Moussaif / Hermes Agent OS

Fugu-style single-interface multi-agent orchestration

Open-source/open-weight oriented - provider-sovereign - evidence-governed

Design KB: technical framework, not executed benchmark proof. Generated June 26, 2026.

Document Contract

What this 500-page KB is allowed to claim and how it should be used.

CONTRACT

This book is a design knowledge base for building Hermes as a Fugu-style orchestration system. It defines hyperskills and hyperloops as executable knowledge units. It does not claim that benchmark results have been executed. Every score, when used, is a target envelope or evaluation design until measured in a real harness.

CORE AIM

Convert the Moussaif MCAI/MICAGI/neoAGI concepts into operational components: MCD scoring, Vascular Compute Routing, agent courts, GraphRAG checks, tool loops, memory reconstruction, and provider-sovereign model fusion.

NON-NEGOTIABLES

- No single-vendor dependency.
- No blind model worship: every output must be routed, checked, ranked, and remembered.
- Hosted-free endpoints are prototyping surfaces; sensitive production routes should move to open-weight local or NIM/self-hosted deployments.
- Novelty must pass demolition: a new concept is not valuable until it survives falsification, implementation pressure, and field transfer.

How to Read This KB

Operate it as a technical manual, a concept ontology, and a routing design book.

USE PATTERN
Read a page as a card: definition, function, loop contract, metrics, failure modes, and implementation note. Together, the cards form a knowledge machine. They can be transformed into prompts, model registry entries, MCP tools, test tasks, or implementation tickets.

OPERATING LEVELS
Level 1: vocabulary. Level 2: loop design. Level 3: technical orchestration. Level 4: benchmarked governance. Level 5: self-improving router memory. Level 6: sovereign deployment across hosted, self-hosted, and local open-weight models.

```
KB page -> prompt contract -> agent role -> route policy -> eval case -> memory up
  date
```

IMMEDIATE USE
- Build registry.json from the model and role definitions.
- Build route_score.ts from the formulas.
- Build court.ts from the Falsification Court pages.
- Build golden_tasks.yaml from the evaluation pages.
- Build domain adapters from the deployment pages.

Hermes-Fugu System Map

From user intent to dynamic agentic scaffold to verified world effect.

SYSTEM MAP

Hermes receives intent, extracts the active difference, estimates complexity, chooses a scaffold, routes to models and tools, collects evidence, falsifies outputs, fuses the answer, acts through the world loop when allowed, and reconstructs memory from what happened.

```
User Intent
  -> Difference Extraction
  -> MCD Scoring
  -> Vascular Compute Routing
  -> Agentic Scaffold
  -> Model/Tool Workers
  -> Falsification Court
  -> Fusion Synthesizer
  -> World-Effect Loop
  -> Self-Reconstruction Memory
```

FUGU-STYLE ADAPTATION

The external interface stays simple: one Hermes endpoint. Internally, Hermes may act like Fugu-Lite with one worker, or like Fugu-Ultra with a deeper team of planner, coder, verifier, critic, domain adapter, and synthesizer.

External Anchors and Fresh Tooling Facts

Current facts from Sakana Fugu, OpenRouter, and NVIDIA NIM.

SAKANA FUGU ANCHOR
The current Fugu framing is a learned orchestrator: a language model that exposes a multi-agent system through a single interface, dynamically deciding model selection, delegation, verification, and synthesis for complex tasks.

OPENROUTER ANCHOR
OpenRouter currently provides free model variants and a free router, but availability and capacity are not guaranteed. That makes it ideal for Hermes prototyping, model discovery, and stochastic route exploration, not for secrets or guaranteed production SLAs.

NVIDIA NIM ANCHOR
NIM exposes OpenAI-compatible APIs and a path from free serverless development endpoints to self-hosted deployment. For Hermes, this means the provider adapter can remain stable while the backend moves from hosted experiments to sovereign GPU infrastructure.

ENGINEERING CONSEQUENCE
- Design the provider layer first.
- Separate route policy from model names.
- Separate public/free routes from sensitive/local routes.
- Store evidence about provider health and model wins.
- Expect the pool to change.

Notation and Core Equations

Compact math for routing, MCD, confidence, latency, and court decisions.

```
MCD = wD*D + wI*I + wT*T + wA*A + wP*P + wC*C - wN*N
RouteScore = fit + evidence + history - latency_tax - risk_tax - cost_tax
FusionConfidence = support_ratio - contradiction_density + verifier_pass + evidenc
  e_freshness
AgencyRegret = best_known_route_score - chosen_route_actual_score
HyperloopClosure = verified_world_effect / intended_world_effect
```

INTERPRETATION

D is difference intensity, I is informational density, T is internal-time depth, A is agency pressure, P is practical world-effect, C is causal coherence, and N is noise/hijack pressure. These values can begin as heuristics and later become learned predictors.

ROUTER DISCIPLINE

The router should never ask only which model is strongest. It must ask which model, under which role, with which tool permission, under which privacy tier, for which subgoal, with which verifier and fallback path.

Full Outline

Page ranges and how the knowledge base is organized.

OUTLINE

- 1 Cover
- 2-8 Orientation
- 9-38 Core Glossary
- 39-78 Foundations
- 79-148 Core Hyperskills
- 149-218 Core Hyperloops
- 219-278 Fusion Router Engineering
- 279-328 Agent Roles and Prompt Contracts
- 329-378 Evaluation, Memory, Benchmark Tribunal
- 379-428 Implementation Blueprints
- 429-468 Security and Anti-Hijack
- 469-488 Training and Evolution
- 489-496 Deployment Roadmap
- 497-500 References

DENSITY RULE

Every page is written as a module because orchestration knowledge must be executable. A chapter is not a decoration; it is a collection of objects that can become code, prompts, tests, registries, and research tasks.

Build Position

The Moussaif thesis: AGI progress as governed orchestration, not monolithic prediction.

THESIS
The central Moussaif thesis for Hermes is that AGI-like progress is not obtained by asking one model to become everything. It emerges from governed difference: extracting the right difference, routing compute through the right vessel, forcing criticism, acting in the world, and rebuilding memory from the result.

REVOLUTIONARY ANGLE
The proposed invention is not merely multi-agent chat. It is a hyperskill operating system: each skill is a controlled loop, each loop has evidence and failure contracts, and the router learns which loops create reliable world-effects under constraints.

NAME
Moussaif Hermes-Fugu Fusion KB: a technical doctrine for Vascular Knowledge Routing, Hyperloop Skill Theory, Inventive Entropy Governance, Recursive Demolition Courts, and sovereign model orchestration.

Core Glossary Module 1: Hyperskill

Definitions that convert vocabulary into implementation objects.

DEFINITION 1: HYPERSKILL
Hyperskill is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Hyperskill.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Hyperskill is useful only if it improves verified task success or reduces preventable error.

DEFINITION 2: HYPERLOOP
Hyperloop is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Hyperloop.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Hyperloop is useful only if it improves verified task success or reduces preventable error.

DEFINITION 3: FUGU-STYLE ORCHESTRATOR
Fugu-style orchestrator is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Fugu-style orchestrator.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Fugu-style orchestrator is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 2: OpenRouter fusion router

Definitions that convert vocabulary into implementation objects.

DEFINITION 4: OPENROUTER FUSION ROUTER
OpenRouter fusion router is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for OpenRouter fusion router.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: OpenRouter fusion router is useful only if it improves verified task success or reduces preventable error.

DEFINITION 5: NVIDIA NIM PROVIDER LAYER
NVIDIA NIM provider layer is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for NVIDIA NIM provider layer.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: NVIDIA NIM provider layer is useful only if it improves verified task success or reduces preventable error.

DEFINITION 6: MCD - META-COMPLEXITY DIFFERENTIATION
MCD - Meta-Complexity Differentiation is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for MCD - Meta-Complexity Differentiation.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: MCD - Meta-Complexity Differentiation is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 3: Vascular Compute Router

Definitions that convert vocabulary into implementation objects.

DEFINITION 7: VASCULAR COMPUTE ROUTER
Vascular Compute Router is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Vascular Compute Router.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Vascular Compute Router is useful only if it improves verified task success or reduces preventable error.

DEFINITION 8: DIFFERENCE EXTRACTION
Difference Extraction is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Difference Extraction.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Difference Extraction is useful only if it improves verified task success or reduces preventable error.

DEFINITION 9: INNER TIME ENGINE
Inner Time Engine is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Inner Time Engine.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Inner Time Engine is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 4: Agency Selector

Definitions that convert vocabulary into implementation objects.

DEFINITION 10: AGENCY SELECTOR
Agency Selector is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Agency Selector.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Agency Selector is useful only if it improves verified task success or reduces preventable error.

DEFINITION 11: FALSIFICATION COURT
Falsification Court is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Falsification Court.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Falsification Court is useful only if it improves verified task success or reduces preventable error.

DEFINITION 12: PERTURBATION COURT
Perturbation Court is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Perturbation Court.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Perturbation Court is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 5: World-Effect Loop

Definitions that convert vocabulary into implementation objects.

DEFINITION 13: WORLD-EFFECT LOOP
World-Effect Loop is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for World-Effect Loop.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: World-Effect Loop is useful only if it improves verified task success or reduces preventable error.

DEFINITION 14: SELF-RECONSTRUCTION MEMORY
Self-Reconstruction Memory is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Self-Reconstruction Memory.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Self-Reconstruction Memory is useful only if it improves verified task success or reduces preventable error.

DEFINITION 15: GRAPHRAG VALIDATION LAYER
GraphRAG validation layer is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for GraphRAG validation layer.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: GraphRAG validation layer is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 6: Tool/Memory check

Definitions that convert vocabulary into implementation objects.

DEFINITION 16: TOOL/MEMORY CHECK
Tool/Memory check is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Tool/Memory check.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Tool/Memory check is useful only if it improves verified task success or reduces preventable error.

DEFINITION 17: CAUSAL NOVELTY ENGINE
Causal Novelty Engine is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Causal Novelty Engine.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Causal Novelty Engine is useful only if it improves verified task success or reduces preventable error.

DEFINITION 18: INVENTIVE ENTROPY GOVERNANCE
Inventive Entropy Governance is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Inventive Entropy Governance.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Inventive Entropy Governance is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 7: Recursive Demolition Court

Definitions that convert vocabulary into implementation objects.

DEFINITION 19: RECURSIVE DEMOLITION COURT
Recursive Demolition Court is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Recursive Demolition Court.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Recursive Demolition Court is useful only if it improves verified task success or reduces preventable error.

DEFINITION 20: VASCULAR KNOWLEDGE ROUTING
Vascular Knowledge Routing is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Vascular Knowledge Routing.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Vascular Knowledge Routing is useful only if it improves verified task success or reduces preventable error.

DEFINITION 21: SELECTIVE ASSISTANCE
Selective Assistance is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Selective Assistance.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Selective Assistance is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 8: Anti-hijack gate

Definitions that convert vocabulary into implementation objects.

DEFINITION 22: ANTI-HIJACK GATE
Anti-hijack gate is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Anti-hijack gate.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Anti-hijack gate is useful only if it improves verified task success or reduces preventable error.

DEFINITION 23: CLAIM RANK LEDGER
Claim rank ledger is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Claim rank ledger.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Claim rank ledger is useful only if it improves verified task success or reduces preventable error.

DEFINITION 24: CAPABILITY VECTOR
Capability vector is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Capability vector.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Capability vector is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 9: Provider sovereignty

Definitions that convert vocabulary into implementation objects.

DEFINITION 25: PROVIDER SOVEREIGNTY
Provider sovereignty is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Provider sovereignty.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Provider sovereignty is useful only if it improves verified task success or reduces preventable error.

DEFINITION 26: MODEL POOL TOPOLOGY
Model pool topology is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Model pool topology.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Model pool topology is useful only if it improves verified task success or reduces preventable error.

DEFINITION 27: AGENTIC SCAFFOLD
Agentic scaffold is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Agentic scaffold.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Agentic scaffold is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 10: Behavioral model composition

Definitions that convert vocabulary into implementation objects.

DEFINITION 28: BEHAVIORAL MODEL COMPOSITION
Behavioral model composition is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Behavioral model composition.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Behavioral model composition is useful only if it improves verified task success or reduces preventable error.

DEFINITION 29: SINGLE-INTERFACE MULTI-AGENT SYSTEM
Single-interface multi-agent system is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Single-interface multi-agent system.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Single-interface multi-agent system is useful only if it improves verified task success or reduces preventable error.

DEFINITION 30: QUALITY-LATENCY FRONTIER
Quality-latency frontier is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Quality-latency frontier.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Quality-latency frontier is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 11: Route confidence

Definitions that convert vocabulary into implementation objects.

DEFINITION 31: ROUTE CONFIDENCE
Route confidence is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Route confidence.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Route confidence is useful only if it improves verified task success or reduces preventable error.

DEFINITION 32: EVIDENCE BUDGET
Evidence budget is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Evidence budget.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Evidence budget is useful only if it improves verified task success or reduces preventable error.

DEFINITION 33: LATENCY TAX
Latency tax is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Latency tax.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Latency tax is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 12: Risk tax

Definitions that convert vocabulary into implementation objects.

DEFINITION 34: RISK TAX
Risk tax is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Risk tax.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Risk tax is useful only if it improves verified task success or reduces preventable error.

DEFINITION 35: FUSION CONFIDENCE
Fusion confidence is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Fusion confidence.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Fusion confidence is useful only if it improves verified task success or reduces preventable error.

DEFINITION 36: DOMAIN ADAPTER
Domain adapter is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Domain adapter.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Domain adapter is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 13: Skill atom

Definitions that convert vocabulary into implementation objects.

DEFINITION 37: SKILL ATOM
Skill atom is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Skill atom.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Skill atom is useful only if it improves verified task success or reduces preventable error.

DEFINITION 38: LOOP GRAMMAR
Loop grammar is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Loop grammar.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Loop grammar is useful only if it improves verified task success or reduces preventable error.

DEFINITION 39: PLANNER-WORKER-VERIFIER TRIAD
Planner-worker-verifier triad is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Planner-worker-verifier triad.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Planner-worker-verifier triad is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 14: Synthesis head

Definitions that convert vocabulary into implementation objects.

DEFINITION 40: SYNTHESIS HEAD
Synthesis head is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Synthesis head.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Synthesis head is useful only if it improves verified task success or reduces preventable error.

DEFINITION 41: REPLAY MEMORY
Replay memory is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Replay memory.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Replay memory is useful only if it improves verified task success or reduces preventable error.

DEFINITION 42: BENCHMARK TRIBUNAL
Benchmark tribunal is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Benchmark tribunal.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Benchmark tribunal is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 15: Policy envelope

Definitions that convert vocabulary into implementation objects.

DEFINITION 43: POLICY ENVELOPE
Policy envelope is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Policy envelope.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Policy envelope is useful only if it improves verified task success or reduces preventable error.

DEFINITION 44: PROMPT CONTRACT
Prompt contract is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Prompt contract.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Prompt contract is useful only if it improves verified task success or reduces preventable error.

DEFINITION 45: TOOL-CALL COVENANT
Tool-call covenant is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Tool-call covenant.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Tool-call covenant is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 16: Long-context compression

Definitions that convert vocabulary into implementation objects.

DEFINITION 46: LONG-CONTEXT COMPRESSION
Long-context compression is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Long-context compression.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Long-context compression is useful only if it improves verified task success or reduces preventable error.

DEFINITION 47: CONTEXT PINNING
Context pinning is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Context pinning.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Context pinning is useful only if it improves verified task success or reduces preventable error.

DEFINITION 48: UNCERTAINTY LEDGER
Uncertainty ledger is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Uncertainty ledger.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Uncertainty ledger is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 17: Failure-mode map

Definitions that convert vocabulary into implementation objects.

DEFINITION 49: FAILURE-MODE MAP
Failure-mode map is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Failure-mode map.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Failure-mode map is useful only if it improves verified task success or reduces preventable error.

DEFINITION 50: DYNAMIC AGENT TOPOLOGY
Dynamic agent topology is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Dynamic agent topology.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Dynamic agent topology is useful only if it improves verified task success or reduces preventable error.

DEFINITION 51: VERIFIER QUORUM
Verifier quorum is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Verifier quorum.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Verifier quorum is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 18: Agent pool health

Definitions that convert vocabulary into implementation objects.

DEFINITION 52: AGENT POOL HEALTH
Agent pool health is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Agent pool health.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Agent pool health is useful only if it improves verified task success or reduces preventable error.

DEFINITION 53: FALLBACK LADDER
Fallback ladder is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Fallback ladder.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Fallback ladder is useful only if it improves verified task success or reduces preventable error.

DEFINITION 54: PROVIDER ABSTRACTION
Provider abstraction is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Provider abstraction.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Provider abstraction is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 19: Model capability card

Definitions that convert vocabulary into implementation objects.

DEFINITION 55: MODEL CAPABILITY CARD
Model capability card is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Model capability card.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Model capability card is useful only if it improves verified task success or reduces preventable error.

DEFINITION 56: TASK GRAPH
Task graph is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Task graph.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Task graph is useful only if it improves verified task success or reduces preventable error.

DEFINITION 57: SUBGOAL LATTICE
Subgoal lattice is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Subgoal lattice.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Subgoal lattice is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 20: Execution trace

Definitions that convert vocabulary into implementation objects.

DEFINITION 58: EXECUTION TRACE
Execution trace is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Execution trace.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Execution trace is useful only if it improves verified task success or reduces preventable error.

DEFINITION 59: OBSERVABILITY SPINE
Observability spine is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Observability spine.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Observability spine is useful only if it improves verified task success or reduces preventable error.

DEFINITION 60: TELEMETRY-TO-ROUTING LOOP
Telemetry-to-routing loop is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Telemetry-to-routing loop.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Telemetry-to-routing loop is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 21: Human override seam

Definitions that convert vocabulary into implementation objects.

DEFINITION 61: HUMAN OVERRIDE SEAM
Human override seam is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Human override seam.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Human override seam is useful only if it improves verified task success or reduces preventable error.

DEFINITION 62: MCP CONNECTOR LAYER
MCP connector layer is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for MCP connector layer.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: MCP connector layer is useful only if it improves verified task success or reduces preventable error.

DEFINITION 63: CODEX BRIDGE
Codex bridge is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Codex bridge.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Codex bridge is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 22: WordPress builder loop

Definitions that convert vocabulary into implementation objects.

DEFINITION 64: WORDPRESS BUILDER LOOP
WordPress builder loop is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for WordPress builder loop.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: WordPress builder loop is useful only if it improves verified task success or reduces preventable error.

DEFINITION 65: REPOSITORY SURGERY LOOP
Repository surgery loop is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Repository surgery loop.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Repository surgery loop is useful only if it improves verified task success or reduces preventable error.

DEFINITION 66: SECURITY SCOPE GUARD
Security scope guard is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Security scope guard.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Security scope guard is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 23: Semantic diff

Definitions that convert vocabulary into implementation objects.

DEFINITION 67: SEMANTIC DIFF
Semantic diff is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Semantic diff.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Semantic diff is useful only if it improves verified task success or reduces preventable error.

DEFINITION 68: PATCH PROPOSAL
Patch proposal is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Patch proposal.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Patch proposal is useful only if it improves verified task success or reduces preventable error.

DEFINITION 69: SANDBOX EVALUATOR
Sandbox evaluator is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Sandbox evaluator.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Sandbox evaluator is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 24: Red-team lens

Definitions that convert vocabulary into implementation objects.

DEFINITION 70: RED-TEAM LENS
Red-team lens is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Red-team lens.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Red-team lens is useful only if it improves verified task success or reduces preventable error.

DEFINITION 71: SELF-REPAIR TRACE
Self-repair trace is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Self-repair trace.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Self-repair trace is useful only if it improves verified task success or reduces preventable error.

DEFINITION 72: MEMORY DISTILLATION
Memory distillation is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Memory distillation.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Memory distillation is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 25: Skill promotion

Definitions that convert vocabulary into implementation objects.

DEFINITION 73: SKILL PROMOTION
Skill promotion is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Skill promotion.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Skill promotion is useful only if it improves verified task success or reduces preventable error.

DEFINITION 74: TASK FAMILY
Task family is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Task family.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Task family is useful only if it improves verified task success or reduces preventable error.

DEFINITION 75: TRANSFER BRIDGE
Transfer bridge is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Transfer bridge.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Transfer bridge is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 26: Domain resonance

Definitions that convert vocabulary into implementation objects.

DEFINITION 76: DOMAIN RESONANCE
Domain resonance is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Domain resonance.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Domain resonance is useful only if it improves verified task success or reduces preventable error.

DEFINITION 77: NOVELTY PRESSURE
Novelty pressure is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Novelty pressure.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Novelty pressure is useful only if it improves verified task success or reduces preventable error.

DEFINITION 78: FALSE NOVELTY DETECTOR
False novelty detector is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for False novelty detector.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: False novelty detector is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 27: Sovereign deployment path

Definitions that convert vocabulary into implementation objects.

DEFINITION 79: SOVEREIGN DEPLOYMENT PATH
Sovereign deployment path is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Sovereign deployment path.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Sovereign deployment path is useful only if it improves verified task success or reduces preventable error.

DEFINITION 80: HOSTED-FREE PROTOTYPE
Hosted-free prototype is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Hosted-free prototype.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Hosted-free prototype is useful only if it improves verified task success or reduces preventable error.

DEFINITION 81: SELF-HOSTED PRODUCTION
Self-hosted production is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Self-hosted production.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Self-hosted production is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 28: Hyperskill

Definitions that convert vocabulary into implementation objects.

DEFINITION 82: HYPERSKILL
Hyperskill is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Hyperskill.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Hyperskill is useful only if it improves verified task success or reduces preventable error.

DEFINITION 83: HYPERLOOP
Hyperloop is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Hyperloop.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Hyperloop is useful only if it improves verified task success or reduces preventable error.

DEFINITION 84: FUGU-STYLE ORCHESTRATOR
Fugu-style orchestrator is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Fugu-style orchestrator.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Fugu-style orchestrator is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 29: Hyperskill

Definitions that convert vocabulary into implementation objects.

DEFINITION 85: HYPERSKILL
Hyperskill is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Hyperskill.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Hyperskill is useful only if it improves verified task success or reduces preventable error.

DEFINITION 86: HYPERLOOP
Hyperloop is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Hyperloop.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Hyperloop is useful only if it improves verified task success or reduces preventable error.

DEFINITION 87: FUGU-STYLE ORCHESTRATOR
Fugu-style orchestrator is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Fugu-style orchestrator.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Fugu-style orchestrator is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Core Glossary Module 30: Hyperskill

Definitions that convert vocabulary into implementation objects.

DEFINITION 88: HYPERSKILL
Hyperskill is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Hyperskill.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Hyperskill is useful only if it improves verified task success or reduces preventable error.

DEFINITION 89: HYPERLOOP
Hyperloop is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Hyperloop.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Hyperloop is useful only if it improves verified task success or reduces preventable error.

DEFINITION 90: FUGU-STYLE ORCHESTRATOR
Fugu-style orchestrator is defined as an operational object in the Hermes-Fugu architecture, not as a slogan. It has a state, input signals, output obligations, measurable failure modes, and a routing consequence. When the object is invoked, Hermes must know what evidence it needs, what tool access is allowed, which model role should own it, and what memory artifact must be written after execution.

IMPLEMENTATION CONTRACT
- Inputs: task intent, context fragment, privacy tier, evidence budget, and current route memory for Fugu-style orchestrator.
- Outputs: updated state, confidence estimate, claim rank, and a recommended next loop.
- Metric: Fugu-style orchestrator is useful only if it improves verified task success or reduces preventable error.

```
object { name, role, inputs, outputs, metrics, failure_modes, next_loops }
```

Foundation Primitive 1: Hyperskill

Binding Hyperskill to hyperskill Intent Crystallization and loop Difference-to-Route Loop.

TECHNICAL DEFINITION

Hyperskill is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS

In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Hyperskill allows Hermes to decide when Single-best worker selection is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA

- Skill binding: Intent Crystallization = state transform + evidence contract + verifier rule.
- Loop binding: Difference-to-Route Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Hyperskill"
  primary_hyperskill: "Intent Crystallization"
  default_loop: "Difference-to-Route Loop"
  escalation: "Single-best worker selection"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE

Hyperskill fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 2: Hyperloop

Binding Hyperloop to hyperskill MCD Scoring and loop Retrieve-Reason-Verify Loop.

TECHNICAL DEFINITION

Hyperloop is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS

In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Hyperloop allows Hermes to decide when Multimodal branch routing is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA

- Skill binding: MCD Scoring = state transform + evidence contract + verifier rule.
- Loop binding: Retrieve-Reason-Verify Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Hyperloop"
  primary_hyperskill: "MCD Scoring"
  default_loop: "Retrieve-Reason-Verify Loop"
  escalation: "Multimodal branch routing"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE

Hyperloop fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 3: Fugu-style orchestrator

Binding Fugu-style orchestrator to hyperskill Provider-Agnostic Routing and loop Bug-Repro-Fix Loop.

TECHNICAL DEFINITION

Fugu-style orchestrator is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS

In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Fugu-style orchestrator allows Hermes to decide when Risk-taxed route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA

- Skill binding: Provider-Agnostic Routing = state transform + evidence contract + verifier rule.
- Loop binding: Bug-Repro-Fix Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Fugu-style orchestrator"
  primary_hyperskill: "Provider-Agnostic Routing"
  default_loop: "Bug-Repro-Fix Loop"
  escalation: "Risk-taxed route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE

Fugu-style orchestrator fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 4: OpenRouter fusion router

Binding OpenRouter fusion router to hyperskill Long-Context Compression and loop Assumption-Attack-Repair Loop.

TECHNICAL DEFINITION

OpenRouter fusion router is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS

In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. OpenRouter fusion router allows Hermes to decide when Canary route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA

- Skill binding: Long-Context Compression = state transform + evidence contract + verifier rule.
- Loop binding: Assumption-Attack-Repair Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "OpenRouter fusion router"
  primary_hyperskill: "Long-Context Compression"
  default_loop: "Assumption-Attack-Repair Loop"
  escalation: "Canary route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE

OpenRouter fusion router fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 5: NVIDIA NIM provider layer

Binding NVIDIA NIM provider layer to hyperskill GraphRAG Anchoring and loop Context-Compress-Preserve Loop.

TECHNICAL DEFINITION
NVIDIA NIM provider layer is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS
In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. NVIDIA NIM provider layer allows Hermes to decide when MCP tool-aware route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA
- Skill binding: GraphRAG Anchoring = state transform + evidence contract + verifier rule.
- Loop binding: Context-Compress-Preserve Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "NVIDIA NIM provider layer"
  primary_hyperskill: "GraphRAG Anchoring"
  default_loop: "Context-Compress-Preserve Loop"
  escalation: "MCP tool-aware route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE
NVIDIA NIM provider layer fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 6: MCD - Meta-Complexity Differentiation

Binding MCD - Meta-Complexity Differentiation to hyperskill Codebase Cartography and loop Novelty-Propose-Invalidate Loop.

TECHNICAL DEFINITION

MCD - Meta-Complexity Differentiation is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS

In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. MCD - Meta-Complexity Differentiation allows Hermes to decide when Compression-before-route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA

- Skill binding: Codebase Cartography = state transform + evidence contract + verifier rule.
- Loop binding: Novelty-Propose-Invalidate Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "MCD - Meta-Complexity Differentiation"
  primary_hyperskill: "Codebase Cartography"
  default_loop: "Novelty-Propose-Invalidate Loop"
  escalation: "Compression-before-route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE

MCD - Meta-Complexity Differentiation fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 7: Vascular Compute Router

Binding Vascular Compute Router to hyperskill Terminal Observation Reading and loop Benchmark-Run-Regression Loop.

TECHNICAL DEFINITION

Vascular Compute Router is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS

In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Vascular Compute Router allows Hermes to decide when Human override route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA

- Skill binding: Terminal Observation Reading = state transform + evidence contract + verifier rule.
- Loop binding: Benchmark-Run-Regression Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Vascular Compute Router"
  primary_hyperskill: "Terminal Observation Reading"
  default_loop: "Benchmark-Run-Regression Loop"
  escalation: "Human override route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE

Vascular Compute Router fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 8: Difference Extraction

Binding Difference Extraction to hyperskill Contradiction Detection and loop Provider-Fallback-Resume Loop.

TECHNICAL DEFINITION

Difference Extraction is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS

In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Difference Extraction allows Hermes to decide when No-tool safe route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA

- Skill binding: Contradiction Detection = state transform + evidence contract + verifier rule.
- Loop binding: Provider-Fallback-Resume Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Difference Extraction"
  primary_hyperskill: "Contradiction Detection"
  default_loop: "Provider-Fallback-Resume Loop"
  escalation: "No-tool safe route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE

Difference Extraction fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 9: Inner Time Engine

Binding Inner Time Engine to hyperskill Counterexample Search and loop WordPress-Theme-Plugin Loop.

TECHNICAL DEFINITION

Inner Time Engine is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS

In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Inner Time Engine allows Hermes to decide when Fugu-Lite route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA

- Skill binding: Counterexample Search = state transform + evidence contract + verifier rule.
- Loop binding: WordPress-Theme-Plugin Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Inner Time Engine"
  primary_hyperskill: "Counterexample Search"
  default_loop: "WordPress-Theme-Plugin Loop"
  escalation: "Fugu-Lite route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE

Inner Time Engine fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 10: Agency Selector

Binding Agency Selector to hyperskill Security Scope Control and loop Agent-Role-Rebalance Loop.

TECHNICAL DEFINITION

Agency Selector is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS

In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Agency Selector allows Hermes to decide when Agency selection route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA

- Skill binding: Security Scope Control = state transform + evidence contract + verifier rule.
- Loop binding: Agent-Role-Rebalance Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Agency Selector"
  primary_hyperskill: "Security Scope Control"
  default_loop: "Agent-Role-Rebalance Loop"
  escalation: "Agency selection route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE

Agency Selector fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 11: Falsification Court

Binding Falsification Court to hyperskill API Fallback Design and loop Fugu-Ultra Team Loop.

TECHNICAL DEFINITION

Falsification Court is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS

In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Falsification Court allows Hermes to decide when Domain resonance route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA

- Skill binding: API Fallback Design = state transform + evidence contract + verifier rule.
- Loop binding: Fugu-Ultra Team Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Falsification Court"
  primary_hyperskill: "API Fallback Design"
  default_loop: "Fugu-Ultra Team Loop"
  escalation: "Domain resonance route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE

Falsification Court fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 12: Perturbation Court

Binding Perturbation Court to hyperskill Screenshot-to-Task Conversion and loop Inner-Time Projection Loop.

TECHNICAL DEFINITION
Perturbation Court is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS
In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Perturbation Court allows Hermes to decide when Tool-failure route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA
- Skill binding: Screenshot-to-Task Conversion = state transform + evidence contract + verifier rule.
- Loop binding: Inner-Time Projection Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Perturbation Court"
  primary_hyperskill: "Screenshot-to-Task Conversion"
  default_loop: "Inner-Time Projection Loop"
  escalation: "Tool-failure route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE
Perturbation Court fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 13: World-Effect Loop

Binding World-Effect Loop to hyperskill Self-Reconstruction and loop Self-Reconstruction Loop.

TECHNICAL DEFINITION
World-Effect Loop is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS
In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. World-Effect Loop allows Hermes to decide when Single-best worker selection is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA
- Skill binding: Self-Reconstruction = state transform + evidence contract + verifier rule.
- Loop binding: Self-Reconstruction Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "World-Effect Loop"
  primary_hyperskill: "Self-Reconstruction"
  default_loop: "Self-Reconstruction Loop"
  escalation: "Single-best worker selection"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE
World-Effect Loop fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 14: Self-Reconstruction Memory

Binding Self-Reconstruction Memory to hyperskill Domain Vocabulary Injection and loop Causal Novelty Loop.

TECHNICAL DEFINITION

Self-Reconstruction Memory is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS

In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold.
Self-Reconstruction Memory allows Hermes to decide when Multimodal branch routing is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA

- Skill binding: Domain Vocabulary Injection = state transform + evidence contract + verifier rule.
- Loop binding: Causal Novelty Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Self-Reconstruction Memory"
  primary_hyperskill: "Domain Vocabulary Injection"
  default_loop: "Causal Novelty Loop"
  escalation: "Multimodal branch routing"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE

Self-Reconstruction Memory fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 15: GraphRAG validation layer

Binding GraphRAG validation layer to hyperskill Simulation Honesty and loop Long-Session Identity Loop.

TECHNICAL DEFINITION
GraphRAG validation layer is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS
In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. GraphRAG validation layer allows Hermes to decide when Risk-taxed route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA
- Skill binding: Simulation Honesty = state transform + evidence contract + verifier rule.
- Loop binding: Long-Session Identity Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "GraphRAG validation layer"
  primary_hyperskill: "Simulation Honesty"
  default_loop: "Long-Session Identity Loop"
  escalation: "Risk-taxed route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE
GraphRAG validation layer fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 16: Tool/Memory check

Binding Tool/Memory check to hyperskill Novelty Generation and loop Evidence Budget Loop.

TECHNICAL DEFINITION
Tool/Memory check is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS
In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Tool/Memory check allows Hermes to decide when Canary route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA
- Skill binding: Novelty Generation = state transform + evidence contract + verifier rule.
- Loop binding: Evidence Budget Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Tool/Memory check"
  primary_hyperskill: "Novelty Generation"
  default_loop: "Evidence Budget Loop"
  escalation: "Canary route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE
Tool/Memory check fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 17: Causal Novelty Engine

Binding Causal Novelty Engine to hyperskill Inventive Entropy Control and loop Cost Elasticity Loop.

TECHNICAL DEFINITION

Causal Novelty Engine is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS

In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Causal Novelty Engine allows Hermes to decide when MCP tool-aware route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA

- Skill binding: Inventive Entropy Control = state transform + evidence contract + verifier rule.
- Loop binding: Cost Elasticity Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Causal Novelty Engine"
  primary_hyperskill: "Inventive Entropy Control"
  default_loop: "Cost Elasticity Loop"
  escalation: "MCP tool-aware route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE

Causal Novelty Engine fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 18: Inventive Entropy Governance

Binding Inventive Entropy Governance to hyperskill World-Effect Measurement and loop Worker Disagreement Loop.

TECHNICAL DEFINITION
Inventive Entropy Governance is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS
In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Inventive Entropy Governance allows Hermes to decide when Compression-before-route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA
- Skill binding: World-Effect Measurement = state transform + evidence contract + verifier rule.
- Loop binding: Worker Disagreement Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Inventive Entropy Governance"
  primary_hyperskill: "World-Effect Measurement"
  default_loop: "Worker Disagreement Loop"
  escalation: "Compression-before-route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE
Inventive Entropy Governance fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 19: Recursive Demolition Court

Binding Recursive Demolition Court to hyperskill Model Disagreement Mining and loop Tool Failure Recovery Loop.

TECHNICAL DEFINITION

Recursive Demolition Court is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS

In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Recursive Demolition Court allows Hermes to decide when Human override route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA

- Skill binding: Model Disagreement Mining = state transform + evidence contract + verifier rule.
- Loop binding: Tool Failure Recovery Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Recursive Demolition Court"
  primary_hyperskill: "Model Disagreement Mining"
  default_loop: "Tool Failure Recovery Loop"
  escalation: "Human override route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE

Recursive Demolition Court fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 20: Vascular Knowledge Routing

Binding Vascular Knowledge Routing to hyperskill Prompt Contracting and loop Rollback Memory Loop.

TECHNICAL DEFINITION

Vascular Knowledge Routing is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS

In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Vascular Knowledge Routing allows Hermes to decide when No-tool safe route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA

- Skill binding: Prompt Contracting = state transform + evidence contract + verifier rule.
- Loop binding: Rollback Memory Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Vascular Knowledge Routing"
  primary_hyperskill: "Prompt Contracting"
  default_loop: "Rollback Memory Loop"
  escalation: "No-tool safe route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE

Vascular Knowledge Routing fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 21: Selective Assistance

Binding Selective Assistance to hyperskill Structured Output Repair and loop Hyperloop Compilation Loop.

TECHNICAL DEFINITION
Selective Assistance is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS
In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Selective Assistance allows Hermes to decide when Fugu-Lite route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA
- Skill binding: Structured Output Repair = state transform + evidence contract + verifier rule.
- Loop binding: Hyperloop Compilation Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Selective Assistance"
  primary_hyperskill: "Structured Output Repair"
  default_loop: "Hyperloop Compilation Loop"
  escalation: "Fugu-Lite route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE
Selective Assistance fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 22: Anti-hijack gate

Binding Anti-hijack gate to hyperskill WordPress Builder Planning and loop Reinforcement Routing Loop.

TECHNICAL DEFINITION

Anti-hijack gate is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS

In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Anti-hijack gate allows Hermes to decide when Agency selection route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA

- Skill binding: WordPress Builder Planning = state transform + evidence contract + verifier rule.
- Loop binding: Reinforcement Routing Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Anti-hijack gate"
  primary_hyperskill: "WordPress Builder Planning"
  default_loop: "Reinforcement Routing Loop"
  escalation: "Agency selection route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE

Anti-hijack gate fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 23: Claim rank ledger

Binding Claim rank ledger to hyperskill cPanel Automation Planning and loop SLA-Aware Dispatch Loop.

TECHNICAL DEFINITION

Claim rank ledger is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS

In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Claim rank ledger allows Hermes to decide when Domain resonance route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA

- Skill binding: cPanel Automation Planning = state transform + evidence contract + verifier rule.
- Loop binding: SLA-Aware Dispatch Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Claim rank ledger"
  primary_hyperskill: "cPanel Automation Planning"
  default_loop: "SLA-Aware Dispatch Loop"
  escalation: "Domain resonance route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE

Claim rank ledger fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 24: Capability vector

Binding Capability vector to hyperskill Rollback Design and loop Deployment Readiness Loop.

TECHNICAL DEFINITION
Capability vector is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS
In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Capability vector allows Hermes to decide when Tool-failure route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA
- Skill binding: Rollback Design = state transform + evidence contract + verifier rule.
- Loop binding: Deployment Readiness Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Capability vector"
  primary_hyperskill: "Rollback Design"
  default_loop: "Deployment Readiness Loop"
  escalation: "Tool-failure route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE
Capability vector fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 25: Provider sovereignty

Binding Provider sovereignty to hyperskill Telemetry Summarization and loop Research Program Loop.

TECHNICAL DEFINITION

Provider sovereignty is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS

In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Provider sovereignty allows Hermes to decide when Single-best worker selection is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA

- Skill binding: Telemetry Summarization = state transform + evidence contract + verifier rule.
- Loop binding: Research Program Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Provider sovereignty"
  primary_hyperskill: "Telemetry Summarization"
  default_loop: "Research Program Loop"
  escalation: "Single-best worker selection"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE

Provider sovereignty fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 26: Model pool topology

Binding Model pool topology to hyperskill Agent Role Assignment and loop Plan-Act-Observe Loop.

TECHNICAL DEFINITION

Model pool topology is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS

In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Model pool topology allows Hermes to decide when Multimodal branch routing is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA

- Skill binding: Agent Role Assignment = state transform + evidence contract + verifier rule.
- Loop binding: Plan-Act-Observe Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Model pool topology"
  primary_hyperskill: "Agent Role Assignment"
  default_loop: "Plan-Act-Observe Loop"
  escalation: "Multimodal branch routing"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE

Model pool topology fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 27: Agentic scaffold

Binding Agentic scaffold to hyperskill Recursive Delegation Control and loop Coder-Verifier-Patcher Loop.

TECHNICAL DEFINITION
Agentic scaffold is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS
In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Agentic scaffold allows Hermes to decide when Risk-taxed route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA
- Skill binding: Recursive Delegation Control = state transform + evidence contract + verifier rule.
- Loop binding: Coder-Verifier-Patcher Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Agentic scaffold"
  primary_hyperskill: "Recursive Delegation Control"
  default_loop: "Coder-Verifier-Patcher Loop"
  escalation: "Risk-taxed route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE
Agentic scaffold fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 28: Behavioral model composition

Binding Behavioral model composition to hyperskill Rate Limit Handling and loop Evidence-Claim-Rank Loop.

TECHNICAL DEFINITION
Behavioral model composition is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS
In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Behavioral model composition allows Hermes to decide when Canary route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA
- Skill binding: Rate Limit Handling = state transform + evidence contract + verifier rule.
- Loop binding: Evidence-Claim-Rank Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Behavioral model composition"
  primary_hyperskill: "Rate Limit Handling"
  default_loop: "Evidence-Claim-Rank Loop"
  escalation: "Canary route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE
Behavioral model composition fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 29: Single-interface multi-agent system

Binding Single-interface multi-agent system to hyperskill Open-Weight Migration Planning and loop Multimodal-Parse-Explain Loop.

TECHNICAL DEFINITION

Single-interface multi-agent system is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS

In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Single-interface multi-agent system allows Hermes to decide when MCP tool-aware route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA

- Skill binding: Open-Weight Migration Planning = state transform + evidence contract + verifier rule.
- Loop binding: Multimodal-Parse-Explain Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Single-interface multi-agent system"
  primary_hyperskill: "Open-Weight Migration Planning"
  default_loop: "Multimodal-Parse-Explain Loop"
  escalation: "MCP tool-aware route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE

Single-interface multi-agent system fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 30: Quality-latency frontier

Binding Quality-latency frontier to hyperskill Evaluation Harness Design and loop Telemetry-Score-Route Loop.

TECHNICAL DEFINITION
Quality-latency frontier is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS
In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Quality-latency frontier allows Hermes to decide when Compression-before-route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA
- Skill binding: Evaluation Harness Design = state transform + evidence contract + verifier rule.
- Loop binding: Telemetry-Score-Route Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Quality-latency frontier"
  primary_hyperskill: "Evaluation Harness Design"
  default_loop: "Telemetry-Score-Route Loop"
  escalation: "Compression-before-route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE
Quality-latency frontier fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 31: Route confidence

Binding Route confidence to hyperskill Golden-Set Curation and loop GraphRAG-Answer-Audit Loop.

TECHNICAL DEFINITION

Route confidence is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS

In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Route confidence allows Hermes to decide when Human override route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA

- Skill binding: Golden-Set Curation = state transform + evidence contract + verifier rule.
- Loop binding: GraphRAG-Answer-Audit Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Route confidence"
  primary_hyperskill: "Golden-Set Curation"
  default_loop: "GraphRAG-Answer-Audit Loop"
  escalation: "Human override route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE

Route confidence fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 32: Evidence budget

Binding Evidence budget to hyperskill Agent Persona Stability and loop MCP-Select-Call-Observe Loop.

TECHNICAL DEFINITION

Evidence budget is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS

In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Evidence budget allows Hermes to decide when No-tool safe route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA

- Skill binding: Agent Persona Stability = state transform + evidence contract + verifier rule.
- Loop binding: MCP-Select-Call-Observe Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Evidence budget"
  primary_hyperskill: "Agent Persona Stability"
  default_loop: "MCP-Select-Call-Observe Loop"
  escalation: "No-tool safe route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE

Evidence budget fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 33: Latency tax

Binding Latency tax to hyperskill Dialogue Repair and loop NIM-Deploy-Benchmark Loop.

TECHNICAL DEFINITION

Latency tax is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS

In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Latency tax allows Hermes to decide when Fugu-Lite route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA

- Skill binding: Dialogue Repair = state transform + evidence contract + verifier rule.
- Loop binding: NIM-Deploy-Benchmark Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Latency tax"
  primary_hyperskill: "Dialogue Repair"
  default_loop: "NIM-Deploy-Benchmark Loop"
  escalation: "Fugu-Lite route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE

Latency tax fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 34: Risk tax

Binding Risk tax to hyperskill Patent/Literature Investigation and loop Prompt-Schema-Repair Loop.

TECHNICAL DEFINITION
Risk tax is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS
In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Risk tax allows Hermes to decide when Agency selection route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA
- Skill binding: Patent/Literature Investigation = state transform + evidence contract + verifier rule.
- Loop binding: Prompt-Schema-Repair Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Risk tax"
  primary_hyperskill: "Patent/Literature Investigation"
  default_loop: "Prompt-Schema-Repair Loop"
  escalation: "Agency selection route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE
Risk tax fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 35: Fusion confidence

Binding Fusion confidence to hyperskill Cyber Recon Boundary Control and loop Fugu-Lite Single Worker Loop.

TECHNICAL DEFINITION

Fusion confidence is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS

In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Fusion confidence allows Hermes to decide when Domain resonance route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA

- Skill binding: Cyber Recon Boundary Control = state transform + evidence contract + verifier rule.
- Loop binding: Fugu-Lite Single Worker Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Fusion confidence"
  primary_hyperskill: "Cyber Recon Boundary Control"
  default_loop: "Fugu-Lite Single Worker Loop"
  escalation: "Domain resonance route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE

Fusion confidence fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 36: Domain adapter

Binding Domain adapter to hyperskill Architecture Stress Testing and loop Vascular Compute Allocation Loop.

TECHNICAL DEFINITION
Domain adapter is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS
In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Domain adapter allows Hermes to decide when Tool-failure route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA
- Skill binding: Architecture Stress Testing = state transform + evidence contract + verifier rule.
- Loop binding: Vascular Compute Allocation Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Domain adapter"
  primary_hyperskill: "Architecture Stress Testing"
  default_loop: "Vascular Compute Allocation Loop"
  escalation: "Tool-failure route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE
Domain adapter fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 37: Skill atom

Binding Skill atom to hyperskill Knowledge Base Expansion and loop World-Effect Learning Loop.

TECHNICAL DEFINITION

Skill atom is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS

In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Skill atom allows Hermes to decide when Single-best worker selection is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA

- Skill binding: Knowledge Base Expansion = state transform + evidence contract + verifier rule.
- Loop binding: World-Effect Learning Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Skill atom"
  primary_hyperskill: "Knowledge Base Expansion"
  default_loop: "World-Effect Learning Loop"
  escalation: "Single-best worker selection"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE

Skill atom fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 38: Loop grammar

Binding Loop grammar to hyperskill Loop Grammar Compilation and loop Anti-Hijack Loop.

TECHNICAL DEFINITION
Loop grammar is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS
In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Loop grammar allows Hermes to decide when Multimodal branch routing is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA
- Skill binding: Loop Grammar Compilation = state transform + evidence contract + verifier rule.
- Loop binding: Anti-Hijack Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Loop grammar"
  primary_hyperskill: "Loop Grammar Compilation"
  default_loop: "Anti-Hijack Loop"
  escalation: "Multimodal branch routing"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE
Loop grammar fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 39: Planner-worker-verifier triad

Binding Planner-worker-verifier triad to hyperskill Ethical Constraint Routing and loop User-Style Alignment Loop.

TECHNICAL DEFINITION
Planner-worker-verifier triad is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS
In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold.
Planner-worker-verifier triad allows Hermes to decide when Risk-taxed route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA
- Skill binding: Ethical Constraint Routing = state transform + evidence contract + verifier rule.
- Loop binding: User-Style Alignment Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Planner-worker-verifier triad"
  primary_hyperskill: "Ethical Constraint Routing"
  default_loop: "User-Style Alignment Loop"
  escalation: "Risk-taxed route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE
Planner-worker-verifier triad fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Foundation Primitive 40: Synthesis head

Binding Synthesis head to hyperskill Uncertainty-to-Action Conversion and loop Field Instrumentation Loop.

TECHNICAL DEFINITION
Synthesis head is a primitive used to stabilize open-ended agentic behavior. It converts vague intelligence into a repeatable transformation: context enters, constraints are ranked, a route is chosen, evidence is collected, a court attacks the result, and a memory trace updates future routing. Its purpose is not to make a model sound smarter but to reduce ungoverned variance in multi-model execution.

WHY IT MATTERS
In a Fugu-style architecture, the strongest unit is not a single model call but the orchestrated scaffold. Synthesis head allows Hermes to decide when Canary route is sufficient and when to escalate into deeper fusion. This is how Hermes can behave as one interface while internally using a team.

OPERATIONAL FORMULA
- Skill binding: Uncertainty-to-Action Conversion = state transform + evidence contract + verifier rule.
- Loop binding: Field Instrumentation Loop = repeated closure until confidence, budget, or human gate ends the cycle.
- MCD binding: increase compute only when difference intensity and world-effect pressure justify latency.
- Court binding: every high-impact claim must survive attack from a role not responsible for producing it.

```
route_policy:
  primitive: "Synthesis head"
  primary_hyperskill: "Uncertainty-to-Action Conversion"
  default_loop: "Field Instrumentation Loop"
  escalation: "Canary route"
  stop_when: confidence >= 0.82 and contradiction_density <= 0.12
```

FAILURE MODE
Synthesis head fails when it becomes decorative terminology. The implementation test is simple: if removing the primitive from Hermes does not change routing, evaluation, memory, or action policy, then it is not yet a real primitive.

Hyperskill 1: Intent Crystallization

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION

Intent Crystallization is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP

The default loop is Spec-Generate-Test Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Router as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT

- Owner role: Router.
- Primary metric: Task success rate.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Intent Crystallization") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Spec-Generate-Test Loop"
  owner: "Router"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Task success rate")
}
```

MOUSSAIF INTERPRETATION

Intent Crystallization is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 2: Task Decomposition

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION

Task Decomposition is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP

The default loop is Evidence-Claim-Rank Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Researcher as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT

- Owner role: Researcher.
- Primary metric: Patch acceptance rate.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Task Decomposition") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Evidence-Claim-Rank Loop"
  owner: "Researcher"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Patch acceptance rate")
}
```

MOUSSAIF INTERPRETATION

Task Decomposition is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 3: MCD Scoring

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
MCD Scoring is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Assumption-Attack-Repair Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Coder as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Coder.
- Primary metric: Regression-free change.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("MCD Scoring") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Assumption-Attack-Repair Loop"
  owner: "Coder"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Regression-free change")
}
```

MOUSSAIF INTERPRETATION
MCD Scoring is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 4: Model Capability Reading

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION

Model Capability Reading is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP

The default loop is Risk-Scope-Action Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Security Analyst as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT

- Owner role: Security Analyst.
- Primary metric: Contradiction density.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Model Capability Reading") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Risk-Scope-Action Loop"
  owner: "Security Analyst"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Contradiction density")
}
```

MOUSSAIF INTERPRETATION

Model Capability Reading is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 5: Provider-Agnostic Routing

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION

Provider-Agnostic Routing is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP

The default loop is Multimodal-Parse-Explain Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Verifier as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT

- Owner role: Verifier.
- Primary metric: Uncertainty calibration.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Provider-Agnostic Routing") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Multimodal-Parse-Explain Loop"
  owner: "Verifier"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Uncertainty calibration")
}
```

MOUSSAIF INTERPRETATION

Provider-Agnostic Routing is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 6: Latency-Aware Delegation

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Latency-Aware Delegation is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Context-Compress-Preserve Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Contradiction Judge as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Contradiction Judge.
- Primary metric: Latency per solved subtask.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Latency-Aware Delegation") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Context-Compress-Preserve Loop"
  owner: "Contradiction Judge"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Latency per solved subtask")
}
```

MOUSSAIF INTERPRETATION
Latency-Aware Delegation is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 7: Long-Context Compression

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION

Long-Context Compression is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP

The default loop is Memory-Distill-Reuse Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Performance Engineer as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT

- Owner role: Performance Engineer.
- Primary metric: Free-tier fallback rate.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Long-Context Compression") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Memory-Distill-Reuse Loop"
  owner: "Performance Engineer"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Free-tier fallback rate")
}
```

MOUSSAIF INTERPRETATION

Long-Context Compression is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 8: Evidence Retrieval

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Evidence Retrieval is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Telemetry-Score-Route Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Prompt Compiler as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Prompt Compiler.
- Primary metric: Tool-call precision.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Evidence Retrieval") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Telemetry-Score-Route Loop"
  owner: "Prompt Compiler"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Tool-call precision")
}
```

MOUSSAIF INTERPRETATION
Evidence Retrieval is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 9: GraphRAG Anchoring

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION

GraphRAG Anchoring is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP

The default loop is Novelty-Propose-Invalidate Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns MCP Coordinator as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT

- Owner role: MCP Coordinator.
- Primary metric: Schema validity.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("GraphRAG Anchoring") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Novelty-Propose-Invalidate Loop"
  owner: "MCP Coordinator"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Schema validity")
}
```

MOUSSAIF INTERPRETATION

GraphRAG Anchoring is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 10: Tool Calling Discipline

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION

Tool Calling Discipline is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP

The default loop is Invent-Formalize-Test Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns CI/CD Engineer as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT

- Owner role: CI/CD Engineer.
- Primary metric: Long-context retention.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Tool Calling Discipline") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Invent-Formalize-Test Loop"
  owner: "CI/CD Engineer"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Long-context retention")
}
```

MOUSSAIF INTERPRETATION

Tool Calling Discipline is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 11: Codebase Cartography

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Codebase Cartography is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is GraphRAG-Answer-Audit Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Domain Adapter as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Domain Adapter.
- Primary metric: Domain term accuracy.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Codebase Cartography") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "GraphRAG-Answer-Audit Loop"
  owner: "Domain Adapter"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Domain term accuracy")
}
```

MOUSSAIF INTERPRETATION
Codebase Cartography is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 12: Repository Patch Planning

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Repository Patch Planning is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Benchmark-Run-Regression Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Entropy Governor as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Entropy Governor.
- Primary metric: Novelty-to-validity ratio.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Repository Patch Planning") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Benchmark-Run-Regression Loop"
  owner: "Entropy Governor"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Novelty-to-validity ratio")
}
```

MOUSSAIF INTERPRETATION
Repository Patch Planning is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 13: Terminal Observation Reading

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Terminal Observation Reading is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Sandbox-Execute-Repair Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Benchmark Judge as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Benchmark Judge.
- Primary metric: Security-scope compliance.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Terminal Observation Reading") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Sandbox-Execute-Repair Loop"
  owner: "Benchmark Judge"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Security-scope compliance")
}
```

MOUSSAIF INTERPRETATION
Terminal Observation Reading is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 14: Spec-to-Implementation Translation

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Spec-to-Implementation Translation is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is MCP-Select-Call-Observe Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Incident Commander as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Incident Commander.
- Primary metric: Human escalation quality.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Spec-to-Implementation Translation") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "MCP-Select-Call-Observe Loop"
  owner: "Incident Commander"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Human escalation quality")
}
```

MOUSSAIF INTERPRETATION
Spec-to-Implementation Translation is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 15: Contradiction Detection

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Contradiction Detection is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Provider-Fallback-Resume Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Model Pool Curator as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Model Pool Curator.
- Primary metric: Postmortem learning rate.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Contradiction Detection") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Provider-Fallback-Resume Loop"
  owner: "Model Pool Curator"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Postmortem learning rate")
}
```

MOUSSAIF INTERPRETATION
Contradiction Detection is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 16: Assumption Surfacing

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION

Assumption Surfacing is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP

The default loop is Free-Tier-Quota-Shift Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns State Reconciler as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT

- Owner role: State Reconciler.
- Primary metric: Court correction rate.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Assumption Surfacing") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Free-Tier-Quota-Shift Loop"
  owner: "State Reconciler"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Court correction rate")
}
```

MOUSSAIF INTERPRETATION

Assumption Surfacing is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 17: Counterexample Search

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION

Counterexample Search is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP

The default loop is NIM-Deploy-Benchmark Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns NIM Operator as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT

- Owner role: NIM Operator.
- Primary metric: GraphRAG citation fidelity.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Counterexample Search") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "NIM-Deploy-Benchmark Loop"
  owner: "NIM Operator"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="GraphRAG citation fidelity")
}
```

MOUSSAIF INTERPRETATION

Counterexample Search is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 18: Risk Bounding

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Risk Bounding is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is WordPress-Theme-Plugin Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Router as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Router.
- Primary metric: Benchmark drift.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Risk Bounding") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "WordPress-Theme-Plugin Loop"
  owner: "Router"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Benchmark drift")
}
```

MOUSSAIF INTERPRETATION
Risk Bounding is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 19: Security Scope Control

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Security Scope Control is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Codex-Repo-Review Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Researcher as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Researcher.
- Primary metric: Canary anomaly rate.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Security Scope Control") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Codex-Repo-Review Loop"
  owner: "Researcher"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Canary anomaly rate")
}
```

MOUSSAIF INTERPRETATION
Security Scope Control is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 20: Privacy-Aware Dispatch

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Privacy-Aware Dispatch is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Prompt-Schema-Repair Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Coder as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Coder.
- Primary metric: Production incident count.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Privacy-Aware Dispatch") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Prompt-Schema-Repair Loop"
  owner: "Coder"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Production incident count")
}
```

MOUSSAIF INTERPRETATION
Privacy-Aware Dispatch is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 21: API Fallback Design

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
API Fallback Design is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Agent-Role-Rebalance Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Security Analyst as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Security Analyst.
- Primary metric: Autonomy level match.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("API Fallback Design") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Agent-Role-Rebalance Loop"
  owner: "Security Analyst"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Autonomy level match")
}
```

MOUSSAIF INTERPRETATION
API Fallback Design is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 22: Multimodal Perception Routing

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Multimodal Perception Routing is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Verifier-Quorum-Decide Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Verifier as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Verifier.
- Primary metric: MCD score predictive power.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Multimodal Perception Routing") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Verifier-Quorum-Decide Loop"
  owner: "Verifier"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="MCD score predictive power")
}
```

MOUSSAIF INTERPRETATION
Multimodal Perception Routing is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 23: Screenshot-to-Task Conversion

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Screenshot-to-Task Conversion is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Fugu-Lite Single Worker Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Contradiction Judge as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Contradiction Judge.
- Primary metric: Agency selection regret.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Screenshot-to-Task Conversion") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Fugu-Lite Single Worker Loop"
  owner: "Contradiction Judge"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Agency selection regret")
}
```

MOUSSAIF INTERPRETATION
Screenshot-to-Task Conversion is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 24: Memory Distillation

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION

Memory Distillation is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP

The default loop is Fugu-Ultra Team Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Performance Engineer as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT

- Owner role: Performance Engineer.
- Primary metric: Hyperloop closure rate.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Memory Distillation") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Fugu-Ultra Team Loop"
  owner: "Performance Engineer"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Hyperloop closure rate")
}
```

MOUSSAIF INTERPRETATION

Memory Distillation is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 25: Self-Reconstruction

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Self-Reconstruction is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Sovereign-Swap Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Prompt Compiler as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Prompt Compiler.
- Primary metric: Cross-domain generalization.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Self-Reconstruction") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Sovereign-Swap Loop"
  owner: "Prompt Compiler"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Cross-domain generalization")
}
```

MOUSSAIF INTERPRETATION
Self-Reconstruction is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 26: User-Style Preservation

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
User-Style Preservation is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Vascular Compute Allocation Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns MCP Coordinator as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: MCP Coordinator.
- Primary metric: Task success rate.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("User-Style Preservation") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Vascular Compute Allocation Loop"
  owner: "MCP Coordinator"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Task success rate")
}
```

MOUSSAIF INTERPRETATION
User-Style Preservation is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 27: Domain Vocabulary Injection

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION

Domain Vocabulary Injection is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP

The default loop is Inner-Time Projection Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns CI/CD Engineer as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT

- Owner role: CI/CD Engineer.
- Primary metric: Patch acceptance rate.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Domain Vocabulary Injection") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Inner-Time Projection Loop"
  owner: "CI/CD Engineer"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Patch acceptance rate")
}
```

MOUSSAIF INTERPRETATION

Domain Vocabulary Injection is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 28: Benchmark Framing

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Benchmark Framing is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Agency Selection Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Domain Adapter as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Domain Adapter.
- Primary metric: Regression-free change.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Benchmark Framing") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Agency Selection Loop"
  owner: "Domain Adapter"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Regression-free change")
}
```

MOUSSAIF INTERPRETATION
Benchmark Framing is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 29: Simulation Honesty

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Simulation Honesty is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is World-Effect Learning Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Entropy Governor as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Entropy Governor.
- Primary metric: Contradiction density.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Simulation Honesty") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "World-Effect Learning Loop"
  owner: "Entropy Governor"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Contradiction density")
}
```

MOUSSAIF INTERPRETATION
Simulation Honesty is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 30: Claim Rank Calibration

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Claim Rank Calibration is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Self-Reconstruction Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Benchmark Judge as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Benchmark Judge.
- Primary metric: Uncertainty calibration.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Claim Rank Calibration") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Self-Reconstruction Loop"
  owner: "Benchmark Judge"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Uncertainty calibration")
}
```

MOUSSAIF INTERPRETATION
Claim Rank Calibration is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 31: Novelty Generation

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION

Novelty Generation is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP

The default loop is Selective Assistance Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Incident Commander as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT

- Owner role: Incident Commander.
- Primary metric: Latency per solved subtask.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Novelty Generation") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Selective Assistance Loop"
  owner: "Incident Commander"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Latency per solved subtask")
}
```

MOUSSAIF INTERPRETATION

Novelty Generation is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 32: Novelty Verification

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION

Novelty Verification is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP

The default loop is Anti-Hijack Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Model Pool Curator as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT

- Owner role: Model Pool Curator.
- Primary metric: Free-tier fallback rate.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Novelty Verification") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Anti-Hijack Loop"
  owner: "Model Pool Curator"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Free-tier fallback rate")
}
```

MOUSSAIF INTERPRETATION

Novelty Verification is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 33: Inventive Entropy Control

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Inventive Entropy Control is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Causal Novelty Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns State Reconciler as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: State Reconciler.
- Primary metric: Tool-call precision.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Inventive Entropy Control") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Causal Novelty Loop"
  owner: "State Reconciler"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Tool-call precision")
}
```

MOUSSAIF INTERPRETATION
Inventive Entropy Control is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 34: Causal Path Modeling

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Causal Path Modeling is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Recursive Demolition Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns NIM Operator as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: NIM Operator.
- Primary metric: Schema validity.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Causal Path Modeling") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Recursive Demolition Loop"
  owner: "NIM Operator"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Schema validity")
}
```

MOUSSAIF INTERPRETATION
Causal Path Modeling is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 35: World-Effect Measurement

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
World-Effect Measurement is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is User-Style Alignment Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Router as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Router.
- Primary metric: Long-context retention.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("World-Effect Measurement") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "User-Style Alignment Loop"
  owner: "Router"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Long-context retention")
}
```

MOUSSAIF INTERPRETATION
World-Effect Measurement is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 36: Output Fusion

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Output Fusion is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Long-Session Identity Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Researcher as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Researcher.
- Primary metric: Domain term accuracy.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Output Fusion") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Long-Session Identity Loop"
  owner: "Researcher"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Domain term accuracy")
}
```

MOUSSAIF INTERPRETATION
Output Fusion is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 37: Model Disagreement Mining

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Model Disagreement Mining is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Domain Adapter Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Coder as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Coder.
- Primary metric: Novelty-to-validity ratio.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Model Disagreement Mining") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Domain Adapter Loop"
  owner: "Coder"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Novelty-to-validity ratio")
}
```

MOUSSAIF INTERPRETATION
Model Disagreement Mining is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 38: Critic Court Operation

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION

Critic Court Operation is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP

The default loop is Field Instrumentation Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Security Analyst as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT

- Owner role: Security Analyst.
- Primary metric: Security-scope compliance.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Critic Court Operation") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Field Instrumentation Loop"
  owner: "Security Analyst"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Security-scope compliance")
}
```

MOUSSAIF INTERPRETATION

Critic Court Operation is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 39: Prompt Contracting

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION

Prompt Contracting is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP

The default loop is Evidence Budget Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Verifier as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT

- Owner role: Verifier.
- Primary metric: Human escalation quality.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Prompt Contracting") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Evidence Budget Loop"
  owner: "Verifier"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Human escalation quality")
}
```

MOUSSAIF INTERPRETATION

Prompt Contracting is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 40: JSON Schema Governance

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
JSON Schema Governance is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Latency Tax Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Contradiction Judge as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Contradiction Judge.
- Primary metric: Postmortem learning rate.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("JSON Schema Governance") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Latency Tax Loop"
  owner: "Contradiction Judge"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Postmortem learning rate")
}
```

MOUSSAIF INTERPRETATION
JSON Schema Governance is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 41: Structured Output Repair

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Structured Output Repair is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Risk Tax Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Performance Engineer as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Performance Engineer.
- Primary metric: Court correction rate.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Structured Output Repair") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Risk Tax Loop"
  owner: "Performance Engineer"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Court correction rate")
}
```

MOUSSAIF INTERPRETATION
Structured Output Repair is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 42: MCP Tool Selection

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
MCP Tool Selection is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Cost Elasticity Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Prompt Compiler as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Prompt Compiler.
- Primary metric: GraphRAG citation fidelity.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("MCP Tool Selection") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Cost Elasticity Loop"
  owner: "Prompt Compiler"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="GraphRAG citation fidelity")
}
```

MOUSSAIF INTERPRETATION
MCP Tool Selection is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 43: WordPress Builder Planning

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
WordPress Builder Planning is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is State Reconciliation Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns MCP Coordinator as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: MCP Coordinator.
- Primary metric: Benchmark drift.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("WordPress Builder Planning") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "State Reconciliation Loop"
  owner: "MCP Coordinator"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Benchmark drift")
}
```

MOUSSAIF INTERPRETATION
WordPress Builder Planning is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 44: VPS Deployment Reasoning

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
VPS Deployment Reasoning is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Contradiction Harvest Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns CI/CD Engineer as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: CI/CD Engineer.
- Primary metric: Canary anomaly rate.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("VPS Deployment Reasoning") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Contradiction Harvest Loop"
  owner: "CI/CD Engineer"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Canary anomaly rate")
}
```

MOUSSAIF INTERPRETATION
VPS Deployment Reasoning is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 45: cPanel Automation Planning

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
cPanel Automation Planning is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Worker Disagreement Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Domain Adapter as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Domain Adapter.
- Primary metric: Production incident count.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("cPanel Automation Planning") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Worker Disagreement Loop"
  owner: "Domain Adapter"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Production incident count")
}
```

MOUSSAIF INTERPRETATION
cPanel Automation Planning is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 46: CI/CD Looping

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION

CI/CD Looping is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP

The default loop is Model Pool Health Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Entropy Governor as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT

- Owner role: Entropy Governor.
- Primary metric: Autonomy level match.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("CI/CD Looping") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Model Pool Health Loop"
  owner: "Entropy Governor"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Autonomy level match")
}
```

MOUSSAIF INTERPRETATION

CI/CD Looping is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 47: Rollback Design

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Rollback Design is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Capability Card Update Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Benchmark Judge as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Benchmark Judge.
- Primary metric: MCD score predictive power.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Rollback Design") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Capability Card Update Loop"
  owner: "Benchmark Judge"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="MCD score predictive power")
}
```

MOUSSAIF INTERPRETATION
Rollback Design is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 48: Observability Design

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Observability Design is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Tool Failure Recovery Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Incident Commander as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Incident Commander.
- Primary metric: Agency selection regret.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Observability Design") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Tool Failure Recovery Loop"
  owner: "Incident Commander"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Agency selection regret")
}
```

MOUSSAIF INTERPRETATION
Observability Design is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 49: Telemetry Summarization

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Telemetry Summarization is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Policy Envelope Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Model Pool Curator as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Model Pool Curator.
- Primary metric: Hyperloop closure rate.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Telemetry Summarization") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Policy Envelope Loop"
  owner: "Model Pool Curator"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Hyperloop closure rate")
}
```

MOUSSAIF INTERPRETATION
Telemetry Summarization is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 50: Routing Memory Update

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Routing Memory Update is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Scope Gate Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns State Reconciler as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: State Reconciler.
- Primary metric: Cross-domain generalization.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Routing Memory Update") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Scope Gate Loop"
  owner: "State Reconciler"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Cross-domain generalization")
}
```

MOUSSAIF INTERPRETATION
Routing Memory Update is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 51: Agent Role Assignment

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Agent Role Assignment is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Rollback Memory Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns NIM Operator as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: NIM Operator.
- Primary metric: Task success rate.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Agent Role Assignment") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Rollback Memory Loop"
  owner: "NIM Operator"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Task success rate")
}
```

MOUSSAIF INTERPRETATION
Agent Role Assignment is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 52: Parallel Worker Management

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Parallel Worker Management is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Knowledge Graph Growth Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Router as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Router.
- Primary metric: Patch acceptance rate.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Parallel Worker Management") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Knowledge Graph Growth Loop"
  owner: "Router"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Patch acceptance rate")
}
```

MOUSSAIF INTERPRETATION
Parallel Worker Management is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 53: Recursive Delegation Control

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Recursive Delegation Control is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Skill Promotion Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Researcher as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Researcher.
- Primary metric: Regression-free change.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Recursive Delegation Control") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Skill Promotion Loop"
  owner: "Researcher"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Regression-free change")
}
```

MOUSSAIF INTERPRETATION
Recursive Delegation Control is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 54: Cost Budgeting

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION

Cost Budgeting is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP

The default loop is Hyperloop Compilation Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Coder as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT

- Owner role: Coder.
- Primary metric: Contradiction density.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Cost Budgeting") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Hyperloop Compilation Loop"
  owner: "Coder"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Contradiction density")
}
```

MOUSSAIF INTERPRETATION

Cost Budgeting is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 55: Rate Limit Handling

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Rate Limit Handling is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Eval-Driven Prompt Mutation Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Security Analyst as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Security Analyst.
- Primary metric: Uncertainty calibration.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Rate Limit Handling") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Eval-Driven Prompt Mutation Loop"
  owner: "Security Analyst"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Uncertainty calibration")
}
```

MOUSSAIF INTERPRETATION
Rate Limit Handling is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 56: Free-Tier Reliability Management

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Free-Tier Reliability Management is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Evolutionary Orchestrator Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Verifier as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Verifier.
- Primary metric: Latency per solved subtask.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Free-Tier Reliability Management") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Evolutionary Orchestrator Loop"
  owner: "Verifier"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Latency per solved subtask")
}
```

MOUSSAIF INTERPRETATION
Free-Tier Reliability Management is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 57: Open-Weight Migration Planning

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Open-Weight Migration Planning is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Reinforcement Routing Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Contradiction Judge as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Contradiction Judge.
- Primary metric: Free-tier fallback rate.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Open-Weight Migration Planning") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Reinforcement Routing Loop"
  owner: "Contradiction Judge"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Free-tier fallback rate")
}
```

MOUSSAIF INTERPRETATION
Open-Weight Migration Planning is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 58: NIM Self-Hosting Transition

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
NIM Self-Hosting Transition is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Human Override Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Performance Engineer as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Performance Engineer.
- Primary metric: Tool-call precision.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("NIM Self-Hosting Transition") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Human Override Loop"
  owner: "Performance Engineer"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Tool-call precision")
}
```

MOUSSAIF INTERPRETATION
NIM Self-Hosting Transition is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 59: Evaluation Harness Design

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Evaluation Harness Design is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Autonomy Level Adjustment Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Prompt Compiler as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Prompt Compiler.
- Primary metric: Schema validity.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Evaluation Harness Design") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Autonomy Level Adjustment Loop"
  owner: "Prompt Compiler"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Schema validity")
}
```

MOUSSAIF INTERPRETATION
Evaluation Harness Design is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 60: Synthetic Task Generation

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Synthetic Task Generation is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is SLA-Aware Dispatch Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns MCP Coordinator as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: MCP Coordinator.
- Primary metric: Long-context retention.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Synthetic Task Generation") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "SLA-Aware Dispatch Loop"
  owner: "MCP Coordinator"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Long-context retention")
}
```

MOUSSAIF INTERPRETATION
Synthetic Task Generation is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 61: Golden-Set Curation

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION

Golden-Set Curation is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP

The default loop is Security Evidence Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns CI/CD Engineer as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT

- Owner role: CI/CD Engineer.
- Primary metric: Domain term accuracy.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Golden-Set Curation") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Security Evidence Loop"
  owner: "CI/CD Engineer"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Domain term accuracy")
}
```

MOUSSAIF INTERPRETATION

Golden-Set Curation is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 62: Regression Detection

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Regression Detection is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Compliance Narrative Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Domain Adapter as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Domain Adapter.
- Primary metric: Novelty-to-validity ratio.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Regression Detection") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Compliance Narrative Loop"
  owner: "Domain Adapter"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Novelty-to-validity ratio")
}
```

MOUSSAIF INTERPRETATION
Regression Detection is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 63: Agent Persona Stability

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION

Agent Persona Stability is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP

The default loop is Deployment Readiness Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Entropy Governor as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT

- Owner role: Entropy Governor.
- Primary metric: Security-scope compliance.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Agent Persona Stability") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Deployment Readiness Loop"
  owner: "Entropy Governor"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Security-scope compliance")
}
```

MOUSSAIF INTERPRETATION

Agent Persona Stability is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 64: Multi-Turn State Control

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Multi-Turn State Control is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Production Incident Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Benchmark Judge as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Benchmark Judge.
- Primary metric: Human escalation quality.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Multi-Turn State Control") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Production Incident Loop"
  owner: "Benchmark Judge"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Human escalation quality")
}
```

MOUSSAIF INTERPRETATION
Multi-Turn State Control is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 65: Dialogue Repair

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Dialogue Repair is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Postmortem-to-KB Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Incident Commander as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Incident Commander.
- Primary metric: Postmortem learning rate.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Dialogue Repair") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Postmortem-to-KB Loop"
  owner: "Incident Commander"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Postmortem learning rate")
}
```

MOUSSAIF INTERPRETATION
Dialogue Repair is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 66: Scientific Source Mapping

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Scientific Source Mapping is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Research Program Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Model Pool Curator as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Model Pool Curator.
- Primary metric: Court correction rate.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Scientific Source Mapping") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Research Program Loop"
  owner: "Model Pool Curator"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Court correction rate")
}
```

MOUSSAIF INTERPRETATION
Scientific Source Mapping is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 67: Patent/Literature Investigation

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Patent/Literature Investigation is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Difference-to-Route Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns State Reconciler as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: State Reconciler.
- Primary metric: GraphRAG citation fidelity.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Patent/Literature Investigation") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Difference-to-Route Loop"
  owner: "State Reconciler"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="GraphRAG citation fidelity")
}
```

MOUSSAIF INTERPRETATION
Patent/Literature Investigation is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 68: Security Report Assembly

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Security Report Assembly is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Question-to-Scaffold Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns NIM Operator as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: NIM Operator.
- Primary metric: Benchmark drift.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Security Report Assembly") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Question-to-Scaffold Loop"
  owner: "NIM Operator"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Benchmark drift")
}
```

MOUSSAIF INTERPRETATION
Security Report Assembly is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 69: Cyber Recon Boundary Control

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Cyber Recon Boundary Control is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Plan-Act-Observe Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Router as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Router.
- Primary metric: Canary anomaly rate.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Cyber Recon Boundary Control") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Plan-Act-Observe Loop"
  owner: "Router"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Canary anomaly rate")
}
```

MOUSSAIF INTERPRETATION
Cyber Recon Boundary Control is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperskill 70: Mathematical Re-derivation

A hyperskill is a transferable agentic capability with routing, evidence, and memory contracts.

DEFINITION
Mathematical Re-derivation is a core Hermes hyperskill: a capability that can survive domain transfer because it is expressed as a governed loop rather than a one-off prompt. It contains an intent signature, a model-role preference, a tool-permission profile, a verifier requirement, and a memory update rule.

EXECUTION LOOP
The default loop is Retrieve-Reason-Verify Loop. Hermes begins with the user difference, converts it into a subgoal lattice, assigns Researcher as the owning role, and keeps the loop alive until the output has a verified effect or is downgraded to an uncertainty note.

CAPABILITY CONTRACT
- Owner role: Researcher.
- Primary metric: Production incident count.
- Input: user task, context, constraints, provider tier, and previous route-memory wins.
- Output: an action-ready artifact plus its evidence ledger, uncertainty ledger, and failure-mode map.
- Promotion rule: the hyperskill becomes core only after repeated wins across at least three task families.

```
hyperskill("Mathematical Re-derivation") {
  route: score_by(MCD, privacy_tier, capability_vector, latency_tax)
  loop: "Retrieve-Reason-Verify Loop"
  owner: "Researcher"
  verify: court.quorum(min_pass=2, max_contradiction=0.12)
  remember: update_route_memory(metric="Production incident count")
}
```

MOUSSAIF INTERPRETATION
Mathematical Re-derivation is a vessel of agency. Its intelligence is measured by whether it selects meaningful differences, preserves inner-time continuity, acts under constraint, and reconstructs memory in a way that makes the next execution stronger.

Hyperloop 1: Difference-to-Route Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Difference-to-Route Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Model Capability Reading to Evidence-budgeted route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Claim support ratio passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Difference-to-Route Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Claim support ratio", verdict)
  memory.write(loop="Difference-to-Route Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Difference-to-Route Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 2: Question-to-Scaffold Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Question-to-Scaffold Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Latency-Aware Delegation to Role-specialized routing. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Contradiction density passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Question-to-Scaffold Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Contradiction density", verdict)
  memory.write(loop="Question-to-Scaffold Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Question-to-Scaffold Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 3: Plan-Act-Observe Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Plan-Act-Observe Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Evidence Retrieval to Quorum routing. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Assumption visibility passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Plan-Act-Observe Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Assumption visibility", verdict)
  memory.write(loop="Plan-Act-Observe Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Plan-Act-Observe Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 4: Retrieve-Reason-Verify Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Retrieve-Reason-Verify Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Tool Calling Discipline to Shadow model route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Uncertainty calibration passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Retrieve-Reason-Verify Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Uncertainty calibration", verdict)
  memory.write(loop="Retrieve-Reason-Verify Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Retrieve-Reason-Verify Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 5: Decompose-Delegate-Fuse Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Decompose-Delegate-Fuse Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Repository Patch Planning to Canary route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Route confidence accuracy passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Decompose-Delegate-Fuse Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Route confidence accuracy", verdict)
  memory.write(loop="Decompose-Delegate-Fuse Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Decompose-Delegate-Fuse Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 6: Coder-Verifier-Patcher Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Coder-Verifier-Patcher Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Spec-to-Implementation Translation to Cascade route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Latency per solved subtask passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Coder-Verifier-Patcher Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Latency per solved subtask", verdict)
  memory.write(loop="Coder-Verifier-Patcher Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Coder-Verifier-Patcher Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 7: Bug-Repro-Fix Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Bug-Repro-Fix Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Assumption Surfacing to ReAct scaffold route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Cost per verified answer passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Bug-Repro-Fix Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Cost per verified answer", verdict)
  memory.write(loop="Bug-Repro-Fix Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Bug-Repro-Fix Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 8: Spec-Generate-Test Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Spec-Generate-Test Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Risk Bounding to Tree-search route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Free-tier fallback rate passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Spec-Generate-Test Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Free-tier fallback rate", verdict)
  memory.write(loop="Spec-Generate-Test Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Spec-Generate-Test Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 9: Evidence-Claim-Rank Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Evidence-Claim-Rank Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Privacy-Aware Dispatch to GraphRAG constrained route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Provider failure recovery passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Evidence-Claim-Rank Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Provider failure recovery", verdict)
  memory.write(loop="Evidence-Claim-Rank Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Evidence-Claim-Rank Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 10: Assumption-Attack-Repair Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Assumption-Attack-Repair Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Multimodal Perception Routing to MCP tool-aware route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Tool-call precision passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Assumption-Attack-Repair Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Tool-call precision", verdict)
  memory.write(loop="Assumption-Attack-Repair Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Assumption-Attack-Repair Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 11: Risk-Scope-Action Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Risk-Scope-Action Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Memory Distillation to Repository-aware route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Tool-call recall passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Risk-Scope-Action Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Tool-call recall", verdict)
  memory.write(loop="Risk-Scope-Action Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Risk-Scope-Action Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 12: Multimodal-Parse-Explain Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Multimodal-Parse-Explain Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds User-Style Preservation to Security-scoped route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Schema validity passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Multimodal-Parse-Explain Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Schema validity", verdict)
  memory.write(loop="Multimodal-Parse-Explain Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Multimodal-Parse-Explain Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 13: Context-Compress-Preserve Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Context-Compress-Preserve Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Benchmark Framing to Privacy-tier route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when JSON repair frequency passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Context-Compress-Preserve Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("JSON repair frequency", verdict)
  memory.write(loop="Context-Compress-Preserve Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Context-Compress-Preserve Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 14: Memory-Distill-Reuse Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Memory-Distill-Reuse Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Claim Rank Calibration to Context-fragment route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Long-context retention passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Memory-Distill-Reuse Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Long-context retention", verdict)
  memory.write(loop="Memory-Distill-Reuse Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Memory-Distill-Reuse Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 15: Telemetry-Score-Route Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Telemetry-Score-Route Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Novelty Verification to Compression-before-route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Persona stability passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Telemetry-Score-Route Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Persona stability", verdict)
  memory.write(loop="Telemetry-Score-Route Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Telemetry-Score-Route Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 16: Novelty-Propose-Invalidate Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Novelty-Propose-Invalidate Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Causal Path Modeling to Model-pool refresh route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Domain term accuracy passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Novelty-Propose-Invalidate Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Domain term accuracy", verdict)
  memory.write(loop="Novelty-Propose-Invalidate Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Novelty-Propose-Invalidate Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 17: Invent-Formalize-Test Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Invent-Formalize-Test Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Output Fusion to Capability-card route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Memory reuse precision passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Invent-Formalize-Test Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Memory reuse precision", verdict)
  memory.write(loop="Invent-Formalize-Test Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Invent-Formalize-Test Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 18: GraphRAG-Answer-Audit Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
GraphRAG-Answer-Audit Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Critic Court Operation to Observed-win-rate route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Novelty-to-validity ratio passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="GraphRAG-Answer-Audit Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Novelty-to-validity ratio", verdict)
  memory.write(loop="GraphRAG-Answer-Audit Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of GraphRAG-Answer-Audit Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 19: Benchmark-Run-Regression Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Benchmark-Run-Regression Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds JSON Schema Governance to Synthetic benchmark route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when False novelty rate passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Benchmark-Run-Regression Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("False novelty rate", verdict)
  memory.write(loop="Benchmark-Run-Regression Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Benchmark-Run-Regression Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 20: Sandbox-Execute-Repair Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Sandbox-Execute-Repair Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds MCP Tool Selection to Human override route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Security-scope compliance passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Sandbox-Execute-Repair Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Security-scope compliance", verdict)
  memory.write(loop="Sandbox-Execute-Repair Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Sandbox-Execute-Repair Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 21: MCP-Select-Call-Observe Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
MCP-Select-Call-Observe Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds VPS Deployment Reasoning to Disagreement expansion route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Privacy dispatch violations passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="MCP-Select-Call-Observe Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Privacy dispatch violations", verdict)
  memory.write(loop="MCP-Select-Call-Observe Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of MCP-Select-Call-Observe Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 22: Provider-Fallback-Resume Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Provider-Fallback-Resume Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds CI/CD Looping to Confidence collapse route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Human escalation quality passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Provider-Fallback-Resume Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Human escalation quality", verdict)
  memory.write(loop="Provider-Fallback-Resume Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Provider-Fallback-Resume Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 23: Free-Tier-Quota-Shift Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Free-Tier-Quota-Shift Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Observability Design to Novelty court route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when World-effect confirmation passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Free-Tier-Quota-Shift Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("World-effect confirmation", verdict)
  memory.write(loop="Free-Tier-Quota-Shift Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Free-Tier-Quota-Shift Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 24: NIM-Deploy-Benchmark Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
NIM-Deploy-Benchmark Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Routing Memory Update to Repair-only route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Postmortem learning rate passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="NIM-Deploy-Benchmark Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Postmortem learning rate", verdict)
  memory.write(loop="NIM-Deploy-Benchmark Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of NIM-Deploy-Benchmark Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 25: WordPress-Theme-Plugin Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
WordPress-Theme-Plugin Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Parallel Worker Management to No-tool safe route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Model disagreement value passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="WordPress-Theme-Plugin Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Model disagreement value", verdict)
  memory.write(loop="WordPress-Theme-Plugin Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of WordPress-Theme-Plugin Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 26: Codex-Repo-Review Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Codex-Repo-Review Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Cost Budgeting to Local-only route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Court correction rate passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Codex-Repo-Review Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Court correction rate", verdict)
  memory.write(loop="Codex-Repo-Review Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Codex-Repo-Review Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 27: Prompt-Schema-Repair Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Prompt-Schema-Repair Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Free-Tier Reliability Management to NIM production route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Fusion compression loss passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Prompt-Schema-Repair Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Fusion compression loss", verdict)
  memory.write(loop="Prompt-Schema-Repair Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Prompt-Schema-Repair Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 28: Agent-Role-Rebalance Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Agent-Role-Rebalance Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds NIM Self-Hosting Transition to OpenRouter prototype route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when GraphRAG citation fidelity passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Agent-Role-Rebalance Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("GraphRAG citation fidelity", verdict)
  memory.write(loop="Agent-Role-Rebalance Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Agent-Role-Rebalance Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 29: Verifier-Quorum-Decide Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Verifier-Quorum-Decide Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Synthetic Task Generation to Open-weight migration route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Evidence freshness passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Verifier-Quorum-Decide Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Evidence freshness", verdict)
  memory.write(loop="Verifier-Quorum-Decide Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Verifier-Quorum-Decide Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 30: Fugu-Lite Single Worker Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION

Fugu-Lite Single Worker Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR

This loop binds Regression Detection to Fugu-Lite route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE

- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Benchmark drift passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Fugu-Lite Single Worker Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Benchmark drift", verdict)
  memory.write(loop="Fugu-Lite Single Worker Loop", verdict=verdict)
```

FAILURE MODE

The most dangerous failure of Fugu-Lite Single Worker Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 31: Fugu-Ultra Team Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Fugu-Ultra Team Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Multi-Turn State Control to Fugu-Ultra route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Golden-set stability passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Fugu-Ultra Team Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Golden-set stability", verdict)
  memory.write(loop="Fugu-Ultra Team Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Fugu-Ultra Team Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 32: Sovereign-Swap Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Sovereign-Swap Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Scientific Source Mapping to Moussaif MCD route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Canary anomaly rate passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Sovereign-Swap Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Canary anomaly rate", verdict)
  memory.write(loop="Sovereign-Swap Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Sovereign-Swap Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 33: Vascular Compute Allocation Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Vascular Compute Allocation Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Security Report Assembly to Vascular compute route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Rollback readiness passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Vascular Compute Allocation Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Rollback readiness", verdict)
  memory.write(loop="Vascular Compute Allocation Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Vascular Compute Allocation Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 34: Inner-Time Projection Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Inner-Time Projection Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Mathematical Re-derivation to Selective assistance route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Production incident count passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Inner-Time Projection Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Production incident count", verdict)
  memory.write(loop="Inner-Time Projection Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Inner-Time Projection Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 35: Agency Selection Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Agency Selection Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Operational Runbook Writing to Agency selection route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Tool failure recovery time passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Agency Selection Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Tool failure recovery time", verdict)
  memory.write(loop="Agency Selection Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Agency Selection Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 36: World-Effect Learning Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
World-Effect Learning Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Skill Ontology Design to World-effect route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Autonomy level match passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="World-Effect Learning Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Autonomy level match", verdict)
  memory.write(loop="World-Effect Learning Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of World-Effect Learning Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 37: Self-Reconstruction Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Self-Reconstruction Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Sovereignty Readiness Audit to Self-reconstruction route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when SLA compliance passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Self-Reconstruction Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("SLA compliance", verdict)
  memory.write(loop="Self-Reconstruction Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Self-Reconstruction Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 38: Selective Assistance Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Selective Assistance Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Human-in-the-Loop Escalation to Claim-rank route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when MCD score predictive power passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Selective Assistance Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("MCD score predictive power", verdict)
  memory.write(loop="Selective Assistance Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Selective Assistance Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 39: Anti-Hijack Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Anti-Hijack Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Falsifiable Roadmap Creation to Anti-hijack route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Vascular compute efficiency passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Anti-Hijack Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Vascular compute efficiency", verdict)
  memory.write(loop="Anti-Hijack Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Anti-Hijack Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 40: Causal Novelty Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Causal Novelty Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Task Decomposition to Domain resonance route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Agency selection regret passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Causal Novelty Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Agency selection regret", verdict)
  memory.write(loop="Causal Novelty Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Causal Novelty Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 41: Recursive Demolition Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Recursive Demolition Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Model Capability Reading to Field adapter route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Self-reconstruction gain passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Recursive Demolition Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Self-reconstruction gain", verdict)
  memory.write(loop="Recursive Demolition Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Recursive Demolition Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 42: User-Style Alignment Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
User-Style Alignment Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Latency-Aware Delegation to Telemetry feedback route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Hyperloop closure rate passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="User-Style Alignment Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Hyperloop closure rate", verdict)
  memory.write(loop="User-Style Alignment Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of User-Style Alignment Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 43: Long-Session Identity Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Long-Session Identity Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Evidence Retrieval to Latency-control route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Hyperskill transfer rate passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Long-Session Identity Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Hyperskill transfer rate", verdict)
  memory.write(loop="Long-Session Identity Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Long-Session Identity Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 44: Domain Adapter Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Domain Adapter Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Tool Calling Discipline to Budget-control route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Cross-domain generalization passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Domain Adapter Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Cross-domain generalization", verdict)
  memory.write(loop="Domain Adapter Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Domain Adapter Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 45: Field Instrumentation Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Field Instrumentation Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Repository Patch Planning to Tool-failure route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Research productivity index passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Field Instrumentation Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Research productivity index", verdict)
  memory.write(loop="Field Instrumentation Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Field Instrumentation Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 46: Evidence Budget Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Evidence Budget Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Spec-to-Implementation Translation to Fallback ladder route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Task success rate passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Evidence Budget Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Task success rate", verdict)
  memory.write(loop="Evidence Budget Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Evidence Budget Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 47: Latency Tax Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Latency Tax Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Assumption Surfacing to Cross-domain route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Verifier pass rate passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Latency Tax Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Verifier pass rate", verdict)
  memory.write(loop="Latency Tax Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Latency Tax Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 48: Risk Tax Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Risk Tax Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Risk Bounding to Mathematical specialist route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Patch acceptance rate passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Risk Tax Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Patch acceptance rate", verdict)
  memory.write(loop="Risk Tax Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Risk Tax Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 49: Cost Elasticity Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Cost Elasticity Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Privacy-Aware Dispatch to Cyber specialist route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Execution success passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Cost Elasticity Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Execution success", verdict)
  memory.write(loop="Cost Elasticity Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Cost Elasticity Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 50: State Reconciliation Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
State Reconciliation Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Multimodal Perception Routing to Single-best worker selection. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Regression-free change passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="State Reconciliation Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Regression-free change", verdict)
  memory.write(loop="State Reconciliation Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of State Reconciliation Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 51: Contradiction Harvest Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Contradiction Harvest Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Memory Distillation to Top-k parallel delegation. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Claim support ratio passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Contradiction Harvest Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Claim support ratio", verdict)
  memory.write(loop="Contradiction Harvest Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Contradiction Harvest Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 52: Worker Disagreement Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Worker Disagreement Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds User-Style Preservation to Planner-first routing. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Contradiction density passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Worker Disagreement Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Contradiction density", verdict)
  memory.write(loop="Worker Disagreement Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Worker Disagreement Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 53: Model Pool Health Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Model Pool Health Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Benchmark Framing to Verifier-first routing. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Assumption visibility passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Model Pool Health Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Assumption visibility", verdict)
  memory.write(loop="Model Pool Health Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Model Pool Health Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 54: Capability Card Update Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Capability Card Update Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Claim Rank Calibration to Coder bench routing. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Uncertainty calibration passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Capability Card Update Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Uncertainty calibration", verdict)
  memory.write(loop="Capability Card Update Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Capability Card Update Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 55: Tool Failure Recovery Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Tool Failure Recovery Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Novelty Verification to Multimodal branch routing. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Route confidence accuracy passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Tool Failure Recovery Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Route confidence accuracy", verdict)
  memory.write(loop="Tool Failure Recovery Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Tool Failure Recovery Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 56: Policy Envelope Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Policy Envelope Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Causal Path Modeling to Long-context route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Latency per solved subtask passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Policy Envelope Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Latency per solved subtask", verdict)
  memory.write(loop="Policy Envelope Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Policy Envelope Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 57: Scope Gate Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Scope Gate Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Output Fusion to Low-latency fallback. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Cost per verified answer passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Scope Gate Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Cost per verified answer", verdict)
  memory.write(loop="Scope Gate Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Scope Gate Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 58: Rollback Memory Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Rollback Memory Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Critic Court Operation to Sovereign provider split. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Free-tier fallback rate passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Rollback Memory Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Free-tier fallback rate", verdict)
  memory.write(loop="Rollback Memory Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Rollback Memory Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 59: Knowledge Graph Growth Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Knowledge Graph Growth Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds JSON Schema Governance to Free-tier stochastic route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Provider failure recovery passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Knowledge Graph Growth Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Provider failure recovery", verdict)
  memory.write(loop="Knowledge Graph Growth Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Knowledge Graph Growth Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 60: Skill Promotion Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Skill Promotion Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds MCP Tool Selection to Risk-taxed route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Tool-call precision passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Skill Promotion Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Tool-call precision", verdict)
  memory.write(loop="Skill Promotion Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Skill Promotion Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 61: Hyperloop Compilation Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Hyperloop Compilation Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds VPS Deployment Reasoning to Evidence-budgeted route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Tool-call recall passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Hyperloop Compilation Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Tool-call recall", verdict)
  memory.write(loop="Hyperloop Compilation Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Hyperloop Compilation Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 62: Eval-Driven Prompt Mutation Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Eval-Driven Prompt Mutation Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds CI/CD Looping to Role-specialized routing. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Schema validity passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Eval-Driven Prompt Mutation Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Schema validity", verdict)
  memory.write(loop="Eval-Driven Prompt Mutation Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Eval-Driven Prompt Mutation Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 63: Evolutionary Orchestrator Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Evolutionary Orchestrator Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Observability Design to Quorum routing. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when JSON repair frequency passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Evolutionary Orchestrator Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("JSON repair frequency", verdict)
  memory.write(loop="Evolutionary Orchestrator Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Evolutionary Orchestrator Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 64: Reinforcement Routing Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Reinforcement Routing Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Routing Memory Update to Shadow model route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Long-context retention passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Reinforcement Routing Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Long-context retention", verdict)
  memory.write(loop="Reinforcement Routing Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Reinforcement Routing Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 65: Human Override Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Human Override Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Parallel Worker Management to Canary route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Persona stability passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Human Override Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Persona stability", verdict)
  memory.write(loop="Human Override Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Human Override Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 66: Autonomy Level Adjustment Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Autonomy Level Adjustment Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Cost Budgeting to Cascade route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Domain term accuracy passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Autonomy Level Adjustment Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Domain term accuracy", verdict)
  memory.write(loop="Autonomy Level Adjustment Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Autonomy Level Adjustment Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 67: SLA-Aware Dispatch Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
SLA-Aware Dispatch Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Free-Tier Reliability Management to ReAct scaffold route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Memory reuse precision passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="SLA-Aware Dispatch Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Memory reuse precision", verdict)
  memory.write(loop="SLA-Aware Dispatch Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of SLA-Aware Dispatch Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 68: Security Evidence Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Security Evidence Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds NIM Self-Hosting Transition to Tree-search route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Novelty-to-validity ratio passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Security Evidence Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Novelty-to-validity ratio", verdict)
  memory.write(loop="Security Evidence Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Security Evidence Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 69: Compliance Narrative Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Compliance Narrative Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Synthetic Task Generation to GraphRAG constrained route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when False novelty rate passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Compliance Narrative Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("False novelty rate", verdict)
  memory.write(loop="Compliance Narrative Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Compliance Narrative Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Hyperloop 70: Deployment Readiness Loop

A hyperloop is a closed execution cycle that turns agentic reasoning into verified change.

LOOP DEFINITION
Deployment Readiness Loop is a recurrent control cycle. It accepts a goal, context, constraints, model pool, tools, and memory. It emits a verified state change or an explicit refusal/downgrade. It becomes AGI-oriented when it transfers across domains without losing its evaluation discipline.

LOOP GRAMMAR
This loop binds Regression Detection to MCP tool-aware route. It must include: trigger condition, route decision, worker instruction, observation step, falsification step, fusion step, stop condition, and memory update. Without these eight elements, the loop is only a prompt pattern.

STATE MACHINE
- S0 - Detect active difference and expected world-effect.
- S1 - Score MCD and choose route depth: single worker, top-k, or courted fusion.
- S2 - Execute worker roles with tool and privacy limits.
- S3 - Compare evidence, contradictions, and risk.
- S4 - Close loop when Security-scope compliance passes threshold, otherwise repair or escalate.

```
while not closed:
  delta = extract_difference(task)
  route = router.choose(delta, loop="Deployment Readiness Loop")
  result = workers.run(route)
  verdict = court.attack(result)
  closed = eval("Security-scope compliance", verdict)
  memory.write(loop="Deployment Readiness Loop", verdict=verdict)
```

FAILURE MODE
The most dangerous failure of Deployment Readiness Loop is premature closure: the system stops because an answer sounds complete, not because the claimed effect has been tested. Hermes must distinguish rhetorical completion from verified closure.

Fusion Router Pattern 1: Single-best worker selection

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Single-best worker selection is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: nvidia/nemotron-3-ultra-550b-a55b:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Task success rate.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 2: Top-k parallel delegation

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Top-k parallel delegation is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: nvidia/nemotron-3-super-120b-a12b:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Execution success.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 3: Planner-first routing

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION
Planner-first routing is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING
Candidate model: openai/gpt-oss-120b:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS
- Metric to monitor: Contradiction density.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE
The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 4: Verifier-first routing

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION
Verifier-first routing is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING
Candidate model: qwen/qwen3-coder:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS
- Metric to monitor: Route confidence accuracy.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE
The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 5: Coder bench routing

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Coder bench routing is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: poolside/laguna-m.1:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Free-tier fallback rate.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 6: Multimodal branch routing

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Multimodal branch routing is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: cohere/north-mini-code:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Tool-call recall.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 7: Long-context route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Long-context route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: google/gemma-4-31b-it:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Long-context retention.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 8: Low-latency fallback

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Low-latency fallback is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: nvidia/nemotron-3-nano-30b-a3b:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Memory reuse precision.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 9: Sovereign provider split

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION
Sovereign provider split is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING
Candidate model: nvidia/nemotron-3-nano-omni-30b-a3b-reasoning:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS
- Metric to monitor: Security-scope compliance.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE
The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 10: Free-tier stochastic route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Free-tier stochastic route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: google/gemma-4-26b-a4b-it:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: World-effect confirmation.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 11: Risk-taxed route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Risk-taxed route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: nvidia/nemotron-3-ultra-550b-a55b:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Court correction rate.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 12: Evidence-budgeted route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Evidence-budgeted route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: nvidia/nemotron-3-super-120b-a12b:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Evidence freshness.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 13: Role-specialized routing

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION
Role-specialized routing is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING
Candidate model: openai/gpt-oss-120b:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS
- Metric to monitor: Canary anomaly rate.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE
The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 14: Quorum routing

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION
Quorum routing is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING
Candidate model: qwen/qwen3-coder:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS
- Metric to monitor: Tool failure recovery time.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE
The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 15: Shadow model route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION
Shadow model route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING
Candidate model: poolside/laguna-m.1:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS
- Metric to monitor: MCD score predictive power.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE
The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 16: Canary route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Canary route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: cohere/north-mini-code:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Self-reconstruction gain.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 17: Cascade route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Cascade route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: google/gemma-4-31b-it:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Cross-domain generalization.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 18: ReAct scaffold route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

ReAct scaffold route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: nvidia/nemotron-3-nano-30b-a3b:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Verifier pass rate.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 19: Tree-search route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Tree-search route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: nvidia/nemotron-3-nano-omni-30b-a3b-reasoning:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Regression-free change.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 20: GraphRAG constrained route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

GraphRAG constrained route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: google/gemma-4-26b-a4b-it:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Assumption visibility.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 21: MCP tool-aware route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

MCP tool-aware route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: nvidia/nemotron-3-ultra-550b-a55b:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Latency per solved subtask.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 22: Repository-aware route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Repository-aware route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: nvidia/nemotron-3-super-120b-a12b:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Provider failure recovery.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 23: Security-scoped route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION
Security-scoped route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING
Candidate model: openai/gpt-oss-120b:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS
- Metric to monitor: Schema validity.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE
The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 24: Privacy-tier route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION
Privacy-tier route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING
Candidate model: qwen/qwen3-coder:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS
- Metric to monitor: Persona stability.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE
The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 25: Context-fragment route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION
Context-fragment route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING
Candidate model: poolside/laguna-m.1:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS
- Metric to monitor: Novelty-to-validity ratio.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE
The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 26: Compression-before-route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Compression-before-route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: cohere/north-mini-code:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Privacy dispatch violations.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 27: Model-pool refresh route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Model-pool refresh route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: google/gemma-4-31b-it:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Postmortem learning rate.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 28: Capability-card route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION
Capability-card route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING
Candidate model: nvidia/nemotron-3-nano-30b-a3b:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS
- Metric to monitor: Fusion compression loss.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE
The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 29: Observed-win-rate route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION
Observed-win-rate route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING
Candidate model: nvidia/nemotron-3-nano-omni-30b-a3b-reasoning:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS
- Metric to monitor: Benchmark drift.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE
The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 30: Synthetic benchmark route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Synthetic benchmark route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: google/gemma-4-26b-a4b-it:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Rollback readiness.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 31: Human override route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION
Human override route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING
Candidate model: nvidia/nemotron-3-ultra-550b-a55b:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS
- Metric to monitor: Autonomy level match.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE
The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 32: Disagreement expansion route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Disagreement expansion route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: nvidia/nemotron-3-super-120b-a12b:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Vascular compute efficiency.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 33: Confidence collapse route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION
Confidence collapse route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING
Candidate model: openai/gpt-oss-120b:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS
- Metric to monitor: Hyperloop closure rate.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE
The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 34: Novelty court route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION
Novelty court route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING
Candidate model: qwen/qwen3-coder:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS
- Metric to monitor: Research productivity index.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE
The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 35: Repair-only route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION
Repair-only route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING
Candidate model: poolside/laguna-m.1:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS
- Metric to monitor: Patch acceptance rate.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE
The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 36: No-tool safe route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

No-tool safe route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: cohere/north-mini-code:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Claim support ratio.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 37: Local-only route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Local-only route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: google/gemma-4-31b-it:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Uncertainty calibration.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 38: NIM production route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

NIM production route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: nvidia/nemotron-3-nano-30b-a3b:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Cost per verified answer.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 39: OpenRouter prototype route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

OpenRouter prototype route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: nvidia/nemotron-3-nano-omni-30b-a3b-reasoning:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Tool-call precision.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 40: Open-weight migration route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Open-weight migration route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: google/gemma-4-26b-a4b-it:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: JSON repair frequency.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 41: Fugu-Lite route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Fugu-Lite route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: nvidia/nemotron-3-ultra-550b-a55b:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Domain term accuracy.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 42: Fugu-Ultra route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Fugu-Ultra route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: nvidia/nemotron-3-super-120b-a12b:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: False novelty rate.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 43: Moussaif MCD route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION
Moussaif MCD route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING
Candidate model: openai/gpt-oss-120b:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS
- Metric to monitor: Human escalation quality.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE
The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 44: Vascular compute route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION
Vascular compute route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING
Candidate model: qwen/qwen3-coder:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS
- Metric to monitor: Model disagreement value.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE
The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 45: Selective assistance route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION
Selective assistance route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING
Candidate model: poolside/laguna-m.1:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS
- Metric to monitor: GraphRAG citation fidelity.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE
The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 46: Agency selection route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Agency selection route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: cohere/north-mini-code:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Golden-set stability.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 47: World-effect route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

World-effect route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: google/gemma-4-31b-it:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Production incident count.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 48: Self-reconstruction route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION
Self-reconstruction route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING
Candidate model: nvidia/nemotron-3-nano-30b-a3b:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS
- Metric to monitor: SLA compliance.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE
The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 49: Claim-rank route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Claim-rank route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: nvidia/nemotron-3-nano-omni-30b-a3b-reasoning:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Agency selection regret.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 50: Anti-hijack route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Anti-hijack route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: google/gemma-4-26b-a4b-it:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Hyperskill transfer rate.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 51: Domain resonance route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Domain resonance route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: nvidia/nemotron-3-ultra-550b-a55b:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Task success rate.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 52: Field adapter route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Field adapter route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: nvidia/nemotron-3-super-120b-a12b:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Execution success.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 53: Telemetry feedback route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION
Telemetry feedback route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING
Candidate model: openai/gpt-oss-120b:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS
- Metric to monitor: Contradiction density.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE
The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 54: Latency-control route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION
Latency-control route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING
Candidate model: qwen/qwen3-coder:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS
- Metric to monitor: Route confidence accuracy.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE
The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 55: Budget-control route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION
Budget-control route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING
Candidate model: poolside/laguna-m.1:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS
- Metric to monitor: Free-tier fallback rate.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE
The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 56: Tool-failure route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Tool-failure route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: cohere/north-mini-code:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Tool-call recall.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 57: Fallback ladder route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Fallback ladder route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: google/gemma-4-31b-it:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Long-context retention.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 58: Cross-domain route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Cross-domain route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: nvidia/nemotron-3-nano-30b-a3b:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Memory reuse precision.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 59: Mathematical specialist route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Mathematical specialist route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: nvidia/nemotron-3-nano-omni-30b-a3b-reasoning:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: Security-scope compliance.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Fusion Router Pattern 60: Cyber specialist route

Router engineering for OpenRouter, NVIDIA NIM, and self-hosted open-weight paths.

DEFINITION

Cyber specialist route is a routing strategy for making Hermes behave like one Fugu-style endpoint while actually using a changing pool of models, providers, tools, and verifiers. The policy must be provider-agnostic: model names are data, not architecture.

DEFAULT BINDING

Candidate model: google/gemma-4-26b-a4b-it:free. This binding is provisional. The production router should compare current availability, context size, tool support, privacy tier, latency, observed win rate, and the task family before dispatching.

```
RouteScore(model, task) =
  0.22*domain_fit + 0.20*reasoning_fit + 0.15*context_fit
+ 0.12*tool_fit + 0.10*history_win_rate + 0.08*evidence_fit
- 0.06*latency_tax - 0.04*risk_tax - 0.03*failure_rate

selected = argmax(RouteScore) under privacy_tier and budget constraints
```

CONTROL CHECKS

- Metric to monitor: World-effect confirmation.
- Do not dispatch secrets to hosted-free routes.
- If model pool changes, update capability cards before trusting previous routing memory.
- If verifier disagrees with worker, escalate to courted fusion or downgrade claim rank.
- If all providers fail, return a partial answer plus exact missing execution steps.

NOVELTY ANGLE

The Moussaif addition is to route not only by capability but by vascular pressure: where the active difference needs compute, where evidence is missing, where agency is blocked, and where the world-effect loop requires action.

Agent Role 1: Router

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Router is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Spec-to-Implementation Translation through Difference-to-Route Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Router.
Return JSON only:
{
  "role": "Router",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 2: Planner

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Planner is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Contradiction Detection through Plan-Act-Observe Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Planner.
Return JSON only:
{
  "role": "Planner",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 3: Decomposer

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Decomposer is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Assumption Surfacing through Decompose-Delegate-Fuse Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Decomposer.
Return JSON only:
{
  "role": "Decomposer",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 4: Researcher

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Researcher is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Counterexample Search through Bug-Repro-Fix Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Researcher.
Return JSON only:
{
  "role": "Researcher",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 5: Evidence Miner

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Evidence Miner is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Risk Bounding through Evidence-Claim-Rank Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Evidence Miner.
Return JSON only:
{
  "role": "Evidence Miner",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 6: GraphRAG Librarian

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION

The GraphRAG Librarian is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL

This role primarily serves Security Scope Control through Risk-Scope-Action Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-GraphRAG Librarian.
Return JSON only:
{
  "role": "GraphRAG Librarian",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS

- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 7: Coder

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Coder is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Privacy-Aware Dispatch through Context-Compress-Preserve Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Coder.
Return JSON only:
{
  "role": "Coder",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 8: Patch Writer

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Patch Writer is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves API Fallback Design through Telemetry-Score-Route Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Patch Writer.
Return JSON only:
{
  "role": "Patch Writer",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 9: Terminal Operator

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Terminal Operator is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Multimodal Perception Routing through Invent-Formalize-Test Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Terminal Operator.
Return JSON only:
{
  "role": "Terminal Operator",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 10: Security Analyst

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Security Analyst is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Screenshot-to-Task Conversion through Benchmark-Run-Regression Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Security Analyst.
Return JSON only:
{
  "role": "Security Analyst",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 11: Privacy Officer

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Privacy Officer is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Memory Distillation through MCP-Select-Call-Observe Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Privacy Officer.
Return JSON only:
{
  "role": "Privacy Officer",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 12: Risk Governor

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Risk Governor is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Self-Reconstruction through Free-Tier-Quota-Shift Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Risk Governor.
Return JSON only:
{
  "role": "Risk Governor",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 13: Verifier

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Verifier is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves User-Style Preservation through WordPress-Theme-Plugin Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Verifier.
Return JSON only:
{
  "role": "Verifier",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 14: Falsifier

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Falsifier is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Domain Vocabulary Injection through Prompt-Schema-Repair Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Falsifier.
Return JSON only:
{
  "role": "Falsifier",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 15: Critic

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Critic is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Benchmark Framing through Verifier-Quorum-Decide Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Critic.
Return JSON only:
{
  "role": "Critic",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 16: Contradiction Judge

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION

The Contradiction Judge is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL

This role primarily serves Simulation Honesty through Fugu-Ultra Team Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Contradiction Judge.
Return JSON only:
{
  "role": "Contradiction Judge",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS

- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 17: Mathematical Re-deriver

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Mathematical Re-deriver is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Claim Rank Calibration through Vascular Compute Allocation Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Mathematical Re-deriver.
Return JSON only:
{
  "role": "Mathematical Re-deriver",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 18: Architecture Reviewer

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION

The Architecture Reviewer is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL

This role primarily serves Novelty Generation through Agency Selection Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Architecture Reviewer.
Return JSON only:
{
  "role": "Architecture Reviewer",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS

- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 19: Performance Engineer

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION

The Performance Engineer is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL

This role primarily serves Novelty Verification through Self-Reconstruction Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Performance Engineer.
Return JSON only:
{
  "role": "Performance Engineer",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS

- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 20: Observability Engineer

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION

The Observability Engineer is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL

This role primarily serves Inventive Entropy Control through Anti-Hijack Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Observability Engineer.
Return JSON only:
{
  "role": "Observability Engineer",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS

- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 21: Memory Distiller

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Memory Distiller is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Causal Path Modeling through Recursive Demolition Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Memory Distiller.
Return JSON only:
{
  "role": "Memory Distiller",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 22: Prompt Compiler

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Prompt Compiler is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves World-Effect Measurement through Long-Session Identity Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Prompt Compiler.
Return JSON only:
{
  "role": "Prompt Compiler",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 23: Schema Enforcer

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Schema Enforcer is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Output Fusion through Field Instrumentation Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Schema Enforcer.
Return JSON only:
{
  "role": "Schema Enforcer",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 24: Tool Broker

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION

The Tool Broker is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL

This role primarily serves Model Disagreement Mining through Latency Tax Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Tool Broker.
Return JSON only:
{
  "role": "Tool Broker",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS

- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 25: MCP Coordinator

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The MCP Coordinator is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Critic Court Operation through Cost Elasticity Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-MCP Coordinator.
Return JSON only:
{
  "role": "MCP Coordinator",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 26: WordPress Builder

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The WordPress Builder is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Prompt Contracting through Contradiction Harvest Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-WordPress Builder.
Return JSON only:
{
  "role": "WordPress Builder",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 27: cPanel Deployer

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The cPanel Deployer is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves JSON Schema Governance through Model Pool Health Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-cPanel Deployer.
Return JSON only:
{
  "role": "cPanel Deployer",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 28: CI/CD Engineer

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The CI/CD Engineer is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Structured Output Repair through Tool Failure Recovery Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-CI/CD Engineer.
Return JSON only:
{
  "role": "CI/CD Engineer",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 29: Documentation Synthesizer

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Documentation Synthesizer is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves MCP Tool Selection through Scope Gate Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Documentation Synthesizer.
Return JSON only:
{
  "role": "Documentation Synthesizer",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 30: Executive Summarizer

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Executive Summarizer is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves WordPress Builder Planning through Knowledge Graph Growth Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Executive Summarizer.
Return JSON only:
{
  "role": "Executive Summarizer",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 31: Domain Adapter

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Domain Adapter is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves VPS Deployment Reasoning through Hyperloop Compilation Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Domain Adapter.
Return JSON only:
{
  "role": "Domain Adapter",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 32: Novelty Inventor

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Novelty Inventor is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves cPanel Automation Planning through Evolutionary Orchestrator Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Novelty Inventor.
Return JSON only:
{
  "role": "Novelty Inventor",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 33: Novelty Skeptic

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Novelty Skeptic is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves CI/CD Looping through Human Override Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Novelty Skeptic.
Return JSON only:
{
  "role": "Novelty Skeptic",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 34: Entropy Governor

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION

The Entropy Governor is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL

This role primarily serves Rollback Design through SLA-Aware Dispatch Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Entropy Governor.
Return JSON only:
{
  "role": "Entropy Governor",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS

- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 35: Causal Modeler

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Causal Modeler is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Observability Design through Compliance Narrative Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Causal Modeler.
Return JSON only:
{
  "role": "Causal Modeler",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 36: Simulation Designer

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Simulation Designer is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Telemetry Summarization through Production Incident Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Simulation Designer.
Return JSON only:
{
  "role": "Simulation Designer",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 37: Benchmark Judge

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Benchmark Judge is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Routing Memory Update through Research Program Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Benchmark Judge.
Return JSON only:
{
  "role": "Benchmark Judge",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 38: Human Liaison

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Human Liaison is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Agent Role Assignment through Question-to-Scaffold Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Human Liaison.
Return JSON only:
{
  "role": "Human Liaison",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 39: Rollback Manager

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Rollback Manager is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Parallel Worker Management through Retrieve-Reason-Verify Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Rollback Manager.
Return JSON only:
{
  "role": "Rollback Manager",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 40: Incident Commander

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION

The Incident Commander is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL

This role primarily serves Recursive Delegation Control through Coder-Verifier-Patcher Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Incident Commander.
Return JSON only:
{
  "role": "Incident Commander",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS

- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 41: Sovereignty Auditor

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION

The Sovereignty Auditor is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL

This role primarily serves Cost Budgeting through Spec-Generate-Test Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Sovereignty Auditor.
Return JSON only:
{
  "role": "Sovereignty Auditor",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS

- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 42: Provider Manager

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION

The Provider Manager is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL

This role primarily serves Rate Limit Handling through Assumption-Attack-Repair Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Provider Manager.
Return JSON only:
{
  "role": "Provider Manager",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS

- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 43: Model Pool Curator

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION

The Model Pool Curator is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL

This role primarily serves Free-Tier Reliability Management through Multimodal-Parse-Explain Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Model Pool Curator.
Return JSON only:
{
  "role": "Model Pool Curator",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS

- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 44: Fusion Synthesizer

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION

The Fusion Synthesizer is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL

This role primarily serves Open-Weight Migration Planning through Memory-Distill-Reuse Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Fusion Synthesizer.
Return JSON only:
{
  "role": "Fusion Synthesizer",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS

- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 45: Style Guardian

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Style Guardian is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves NIM Self-Hosting Transition through Novelty-Propose-Invalidate Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Style Guardian.
Return JSON only:
{
  "role": "Style Guardian",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 46: State Reconciler

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The State Reconciler is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Evaluation Harness Design through GraphRAG-Answer-Audit Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-State Reconciler.
Return JSON only:
{
  "role": "State Reconciler",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 47: Policy Monitor

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Policy Monitor is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Synthetic Task Generation through Sandbox-Execute-Repair Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Policy Monitor.
Return JSON only:
{
  "role": "Policy Monitor",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 48: Open-Weight Migrator

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The Open-Weight Migrator is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Golden-Set Curation through Provider-Fallback-Resume Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-Open-Weight Migrator.
Return JSON only:
{
  "role": "Open-Weight Migrator",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 49: NIM Operator

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION
The NIM Operator is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL
This role primarily serves Regression Detection through NIM-Deploy-Benchmark Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-NIM Operator.
Return JSON only:
{
  "role": "NIM Operator",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS
- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Agent Role 50: OpenRouter Gateway

Prompt contract and behavior boundaries for a Hermes worker.

ROLE DEFINITION

The OpenRouter Gateway is not a personality. It is a bounded function inside the scaffold. It owns a narrow transformation, reports uncertainty, refuses tasks outside its scope, and emits structured output for the court and fusion layers.

ASSIGNED HYPERSKILL

This role primarily serves Agent Persona Stability through Codex-Repo-Review Loop. The role must not optimize for eloquence. It optimizes for usable intermediate state: evidence, artifacts, tests, risk notes, confidence, and recommended next action.

```
SYSTEM: You are Hermes-OpenRouter Gateway.
Return JSON only:
{
  "role": "OpenRouter Gateway",
  "task_understanding": "...",
  "answer_or_artifact": "...",
  "evidence": ["..."],
  "uncertainties": ["..."],
  "failure_modes": ["..."],
  "confidence": 0.0,
  "next_action": "..."
}
```

COURT OBLIGATIONS

- Expose assumptions instead of hiding them.
- Separate fact, inference, hypothesis, and implementation choice.
- Mark unsupported claims for downgrade.
- Recommend a verifier when the output has high impact.
- Never write memory directly; propose a memory trace for the distiller.

Evaluation Metric 1: Task success rate

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Task success rate measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Sandbox-Execute-Repair Loop, evaluate Task success rate before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Task success rate"
  hyperskill: "Intent Crystallization"
  loop: "Sandbox-Execute-Repair Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Task success rate is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 2: Verifier pass rate

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Verifier pass rate measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For MCP-Select-Call-Observe Loop, evaluate Verifier pass rate before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Verifier pass rate"
  hyperskill: "Provider-Agnostic Routing"
  loop: "MCP-Select-Call-Observe Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Verifier pass rate is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 3: Patch acceptance rate

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Patch acceptance rate measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Provider-Fallback-Resume Loop, evaluate Patch acceptance rate before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Patch acceptance rate"
  hyperskill: "GraphRAG Anchoring"
  loop: "Provider-Fallback-Resume Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Patch acceptance rate is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 4: Execution success

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Execution success measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Free-Tier-Quota-Shift Loop, evaluate Execution success before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Execution success"
  hyperskill: "Terminal Observation Reading"
  loop: "Free-Tier-Quota-Shift Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Execution success is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 5: Regression-free change

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Regression-free change measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For NIM-Deploy-Benchmark Loop, evaluate Regression-free change before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Regression-free change"
  hyperskill: "Counterexample Search"
  loop: "NIM-Deploy-Benchmark Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Regression-free change is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 6: Claim support ratio

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Claim support ratio measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For WordPress-Theme-Plugin Loop, evaluate Claim support ratio before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Claim support ratio"
  hyperskill: "API Fallback Design"
  loop: "WordPress-Theme-Plugin Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Claim support ratio is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 7: Contradiction density

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Contradiction density measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Codex-Repo-Review Loop, evaluate Contradiction density before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Contradiction density"
  hyperskill: "Self-Reconstruction"
  loop: "Codex-Repo-Review Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Contradiction density is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 8: Assumption visibility

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Assumption visibility measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Prompt-Schema-Repair Loop, evaluate Assumption visibility before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Assumption visibility"
  hyperskill: "Simulation Honesty"
  loop: "Prompt-Schema-Repair Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Assumption visibility is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 9: Uncertainty calibration

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Uncertainty calibration measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Agent-Role-Rebalance Loop, evaluate Uncertainty calibration before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Uncertainty calibration"
  hyperskill: "Inventive Entropy Control"
  loop: "Agent-Role-Rebalance Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Uncertainty calibration is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 10: Route confidence accuracy

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Route confidence accuracy measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Verifier-Quorum-Decide Loop, evaluate Route confidence accuracy before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Route confidence accuracy"
  hyperskill: "Model Disagreement Mining"
  loop: "Verifier-Quorum-Decide Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Route confidence accuracy is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 11: Latency per solved subtask

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Latency per solved subtask measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Fugu-Lite Single Worker Loop, evaluate Latency per solved subtask before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Latency per solved subtask"
  hyperskill: "Structured Output Repair"
  loop: "Fugu-Lite Single Worker Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Latency per solved subtask is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 12: Cost per verified answer

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Cost per verified answer measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Fugu-Ultra Team Loop, evaluate Cost per verified answer before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Cost per verified answer"
  hyperskill: "cPanel Automation Planning"
  loop: "Fugu-Ultra Team Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Cost per verified answer is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 13: Free-tier fallback rate

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Free-tier fallback rate measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Sovereign-Swap Loop, evaluate Free-tier fallback rate before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Free-tier fallback rate"
  hyperskill: "Telemetry Summarization"
  loop: "Sovereign-Swap Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Free-tier fallback rate is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 14: Provider failure recovery

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Provider failure recovery measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Vascular Compute Allocation Loop, evaluate Provider failure recovery before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Provider failure recovery"
  hyperskill: "Recursive Delegation Control"
  loop: "Vascular Compute Allocation Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Provider failure recovery is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 15: Tool-call precision

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Tool-call precision measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Inner-Time Projection Loop, evaluate Tool-call precision before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Tool-call precision"
  hyperskill: "Open-Weight Migration Planning"
  loop: "Inner-Time Projection Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Tool-call precision is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 16: Tool-call recall

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Tool-call recall measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Agency Selection Loop, evaluate Tool-call recall before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Tool-call recall"
  hyperskill: "Golden-Set Curation"
  loop: "Agency Selection Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Tool-call recall is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 17: Schema validity

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Schema validity measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For World-Effect Learning Loop, evaluate Schema validity before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Schema validity"
  hyperskill: "Dialogue Repair"
  loop: "World-Effect Learning Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Schema validity is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 18: JSON repair frequency

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

JSON repair frequency measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Self-Reconstruction Loop, evaluate JSON repair frequency before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "JSON repair frequency"
  hyperskill: "Cyber Recon Boundary Control"
  loop: "Self-Reconstruction Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

JSON repair frequency is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 19: Long-context retention

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Long-context retention measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Selective Assistance Loop, evaluate Long-context retention before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Long-context retention"
  hyperskill: "Knowledge Base Expansion"
  loop: "Selective Assistance Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Long-context retention is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 20: Persona stability

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Persona stability measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Anti-Hijack Loop, evaluate Persona stability before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Persona stability"
  hyperskill: "Ethical Constraint Routing"
  loop: "Anti-Hijack Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Persona stability is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 21: Domain term accuracy

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Domain term accuracy measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Causal Novelty Loop, evaluate Domain term accuracy before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Domain term accuracy"
  hyperskill: "Intent Crystallization"
  loop: "Causal Novelty Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Domain term accuracy is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 22: Memory reuse precision

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Memory reuse precision measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Recursive Demolition Loop, evaluate Memory reuse precision before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Memory reuse precision"
  hyperskill: "Provider-Agnostic Routing"
  loop: "Recursive Demolition Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Memory reuse precision is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 23: Novelty-to-validity ratio

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Novelty-to-validity ratio measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For User-Style Alignment Loop, evaluate Novelty-to-validity ratio before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Novelty-to-validity ratio"
  hyperskill: "GraphRAG Anchoring"
  loop: "User-Style Alignment Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Novelty-to-validity ratio is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 24: False novelty rate

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

False novelty rate measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Long-Session Identity Loop, evaluate False novelty rate before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "False novelty rate"
  hyperskill: "Terminal Observation Reading"
  loop: "Long-Session Identity Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

False novelty rate is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 25: Security-scope compliance

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Security-scope compliance measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Domain Adapter Loop, evaluate Security-scope compliance before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Security-scope compliance"
  hyperskill: "Counterexample Search"
  loop: "Domain Adapter Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Security-scope compliance is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 26: Privacy dispatch violations

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Privacy dispatch violations measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Field Instrumentation Loop, evaluate Privacy dispatch violations before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Privacy dispatch violations"
  hyperskill: "API Fallback Design"
  loop: "Field Instrumentation Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Privacy dispatch violations is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 27: Human escalation quality

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Human escalation quality measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Evidence Budget Loop, evaluate Human escalation quality before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Human escalation quality"
  hyperskill: "Self-Reconstruction"
  loop: "Evidence Budget Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Human escalation quality is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 28: World-effect confirmation

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

World-effect confirmation measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Latency Tax Loop, evaluate World-effect confirmation before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "World-effect confirmation"
  hyperskill: "Simulation Honesty"
  loop: "Latency Tax Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

World-effect confirmation is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 29: Postmortem learning rate

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Postmortem learning rate measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Risk Tax Loop, evaluate Postmortem learning rate before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Postmortem learning rate"
  hyperskill: "Inventive Entropy Control"
  loop: "Risk Tax Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Postmortem learning rate is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 30: Model disagreement value

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Model disagreement value measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Cost Elasticity Loop, evaluate Model disagreement value before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Model disagreement value"
  hyperskill: "Model Disagreement Mining"
  loop: "Cost Elasticity Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Model disagreement value is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 31: Court correction rate

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Court correction rate measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For State Reconciliation Loop, evaluate Court correction rate before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Court correction rate"
  hyperskill: "Structured Output Repair"
  loop: "State Reconciliation Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Court correction rate is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 32: Fusion compression loss

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Fusion compression loss measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Contradiction Harvest Loop, evaluate Fusion compression loss before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Fusion compression loss"
  hyperskill: "cPanel Automation Planning"
  loop: "Contradiction Harvest Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Fusion compression loss is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 33: GraphRAG citation fidelity

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

GraphRAG citation fidelity measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Worker Disagreement Loop, evaluate GraphRAG citation fidelity before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "GraphRAG citation fidelity"
  hyperskill: "Telemetry Summarization"
  loop: "Worker Disagreement Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

GraphRAG citation fidelity is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 34: Evidence freshness

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Evidence freshness measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Model Pool Health Loop, evaluate Evidence freshness before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Evidence freshness"
  hyperskill: "Recursive Delegation Control"
  loop: "Model Pool Health Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Evidence freshness is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 35: Benchmark drift

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Benchmark drift measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Capability Card Update Loop, evaluate Benchmark drift before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Benchmark drift"
  hyperskill: "Open-Weight Migration Planning"
  loop: "Capability Card Update Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Benchmark drift is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 36: Golden-set stability

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Golden-set stability measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Tool Failure Recovery Loop, evaluate Golden-set stability before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Golden-set stability"
  hyperskill: "Golden-Set Curation"
  loop: "Tool Failure Recovery Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Golden-set stability is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 37: Canary anomaly rate

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Canary anomaly rate measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Policy Envelope Loop, evaluate Canary anomaly rate before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Canary anomaly rate"
  hyperskill: "Dialogue Repair"
  loop: "Policy Envelope Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Canary anomaly rate is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 38: Rollback readiness

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Rollback readiness measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Scope Gate Loop, evaluate Rollback readiness before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Rollback readiness"
  hyperskill: "Cyber Recon Boundary Control"
  loop: "Scope Gate Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Rollback readiness is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 39: Production incident count

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Production incident count measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Rollback Memory Loop, evaluate Production incident count before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Production incident count"
  hyperskill: "Knowledge Base Expansion"
  loop: "Rollback Memory Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Production incident count is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 40: Tool failure recovery time

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Tool failure recovery time measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Knowledge Graph Growth Loop, evaluate Tool failure recovery time before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Tool failure recovery time"
  hyperskill: "Ethical Constraint Routing"
  loop: "Knowledge Graph Growth Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Tool failure recovery time is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 41: Autonomy level match

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Autonomy level match measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Skill Promotion Loop, evaluate Autonomy level match before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Autonomy level match"
  hyperskill: "Intent Crystallization"
  loop: "Skill Promotion Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Autonomy level match is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 42: SLA compliance

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

SLA compliance measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Hyperloop Compilation Loop, evaluate SLA compliance before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "SLA compliance"
  hyperskill: "Provider-Agnostic Routing"
  loop: "Hyperloop Compilation Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

SLA compliance is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 43: MCD score predictive power

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

MCD score predictive power measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Eval-Driven Prompt Mutation Loop, evaluate MCD score predictive power before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "MCD score predictive power"
  hyperskill: "GraphRAG Anchoring"
  loop: "Eval-Driven Prompt Mutation Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

MCD score predictive power is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 44: Vascular compute efficiency

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Vascular compute efficiency measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Evolutionary Orchestrator Loop, evaluate Vascular compute efficiency before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Vascular compute efficiency"
  hyperskill: "Terminal Observation Reading"
  loop: "Evolutionary Orchestrator Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Vascular compute efficiency is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 45: Agency selection regret

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Agency selection regret measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Reinforcement Routing Loop, evaluate Agency selection regret before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Agency selection regret"
  hyperskill: "Counterexample Search"
  loop: "Reinforcement Routing Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Agency selection regret is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 46: Self-reconstruction gain

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Self-reconstruction gain measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Human Override Loop, evaluate Self-reconstruction gain before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Self-reconstruction gain"
  hyperskill: "API Fallback Design"
  loop: "Human Override Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Self-reconstruction gain is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 47: Hyperloop closure rate

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Hyperloop closure rate measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Autonomy Level Adjustment Loop, evaluate Hyperloop closure rate before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Hyperloop closure rate"
  hyperskill: "Self-Reconstruction"
  loop: "Autonomy Level Adjustment Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Hyperloop closure rate is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 48: Hyperskill transfer rate

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Hyperskill transfer rate measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For SLA-Aware Dispatch Loop, evaluate Hyperskill transfer rate before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Hyperskill transfer rate"
  hyperskill: "Simulation Honesty"
  loop: "SLA-Aware Dispatch Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Hyperskill transfer rate is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 49: Cross-domain generalization

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Cross-domain generalization measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Security Evidence Loop, evaluate Cross-domain generalization before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Cross-domain generalization"
  hyperskill: "Inventive Entropy Control"
  loop: "Security Evidence Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Cross-domain generalization is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Evaluation Metric 50: Research productivity index

Benchmark tribunal and memory feedback for model orchestration.

METRIC DEFINITION

Research productivity index measures whether Hermes is becoming more capable through orchestration rather than merely producing more text. It must be logged per task family, model route, scaffold shape, verifier result, and final user/world outcome.

MEASUREMENT PATTERN

For Compliance Narrative Loop, evaluate Research productivity index before and after courted fusion. Compare single-worker baseline, OpenRouter free route, specialist model route, and NIM/self-hosted route. Store the winning scaffold, not only the winning model.

```
eval_case:
  metric: "Research productivity index"
  hyperskill: "Model Disagreement Mining"
  loop: "Compliance Narrative Loop"
  baselines: [single_worker, free_router, specialist_top_k, courted_fusion]
  log: [route, provider, latency, verifier_pass, contradiction_density, user_accep
  tance]
  promote_when: repeated_win >= 5 and regression_rate <= 0.05
```

BENCHMARK TRIBUNAL QUESTIONS

- Did the route solve the task or merely sound plausible?
- Which subagent created the decisive improvement?
- Did the verifier catch errors the worker missed?
- Did fusion preserve evidence or compress it away?
- What should memory change before the next similar task?

HONESTY RULE

Research productivity index is not a published benchmark score unless it was measured in a reproducible harness. In design documents, label it as target, simulated envelope, or observed local run.

Implementation Component 1: model_registry.json

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
model_registry.json is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: GraphRAG Librarian. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// model_registry.json
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["False novelty rate", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
model_registry.json becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 2: provider_openrouter.ts

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
provider_openrouter.ts is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Coder. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// provider_openrouter.ts
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Security-scope compliance", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
provider_openrouter.ts becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 3: provider_nvidia_nim.ts

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
provider_nvidia_nim.ts is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Patch Writer. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// provider_nvidia_nim.ts
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Privacy dispatch violations", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
provider_nvidia_nim.ts becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 4: router_score.ts

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
router_score.ts is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Terminal Operator. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// router_score.ts
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Human escalation quality", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
router_score.ts becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 5: mcd_gate.ts

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
mcd_gate.ts is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Security Analyst. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// mcd_gate.ts
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["World-effect confirmation", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
mcd_gate.ts becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 6: task_classifier.ts

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
task_classifier.ts is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Privacy Officer. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// task_classifier.ts
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Postmortem learning rate", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
task_classifier.ts becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 7: scaffold_builder.ts

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
scaffold_builder.ts is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Risk Governor. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// scaffold_builder.ts
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Model disagreement value", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
scaffold_builder.ts becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 8: worker_pool.ts

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
worker_pool.ts is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Verifier. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// worker_pool.ts
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Court correction rate", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
worker_pool.ts becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 9: parallel_executor.ts

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
parallel_executor.ts is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Falsifier. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// parallel_executor.ts
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Fusion compression loss", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
parallel_executor.ts becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 10: court.ts

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
court.ts is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Critic. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// court.ts
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["GraphRAG citation fidelity", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
court.ts becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 11: fusion.ts

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
fusion.ts is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Contradiction Judge. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// fusion.ts
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Evidence freshness", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
fusion.ts becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 12: memory_distiller.ts

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
memory_distiller.ts is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Mathematical Re-deriver. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// memory_distiller.ts
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Benchmark drift", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
memory_distiller.ts becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 13: telemetry_store.ts

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
telemetry_store.ts is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Architecture Reviewer. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// telemetry_store.ts
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Golden-set stability", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
telemetry_store.ts becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 14: eval_harness.ts

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
eval_harness.ts is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Performance Engineer. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// eval_harness.ts
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Canary anomaly rate", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
eval_harness.ts becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 15: golden_tasks.yaml

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
golden_tasks.yaml is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Observability Engineer. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// golden_tasks.yaml
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Rollback readiness", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
golden_tasks.yaml becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 16: prompt_contracts/

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
prompt_contracts/ is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Memory Distiller. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// prompt_contracts/
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Production incident count", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
prompt_contracts/ becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 17: schemas/

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
schemas/ is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Prompt Compiler. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// schemas/
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Tool failure recovery time", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
schemas/ becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 18: mcp_tools/

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
mcp_tools/ is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Schema Enforcer. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// mcp_tools/
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Autonomy level match", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
mcp_tools/ becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 19: graph_rag/

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
graph_rag/ is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Tool Broker. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// graph_rag/
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["SLA compliance", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
graph_rag/ becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 20: vector_index/

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
vector_index/ is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: MCP Coordinator. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// vector_index/
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["MCD score predictive power", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
vector_index/ becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 21: source_ledger/

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
source_ledger/ is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: WordPress Builder. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// source_ledger/
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Vascular compute efficiency", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
source_ledger/ becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 22: repo_adapter/

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
repo_adapter/ is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: cPanel Deployer. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// repo_adapter/
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Agency selection regret", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
repo_adapter/ becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 23: terminal_sandbox/

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
terminal_sandbox/ is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: CI/CD Engineer. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// terminal_sandbox/
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Self-reconstruction gain", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
terminal_sandbox/ becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 24: wordpress_adapter/

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
wordpress_adapter/ is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Documentation Synthesizer. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// wordpress_adapter/
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Hyperloop closure rate", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
wordpress_adapter/ becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 25: cpanel_adapter/

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
cpanel_adapter/ is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Executive Summarizer. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// cpanel_adapter/
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Hyperskill transfer rate", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
cpanel_adapter/ becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 26: github_adapter/

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
github_adapter/ is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Domain Adapter. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// github_adapter/
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Cross-domain generalization", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
github_adapter/ becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 27: vps_adapter/

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
vps_adapter/ is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Novelty Inventor. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// vps_adapter/
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Research productivity index", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
vps_adapter/ becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 28: security_scope.ts

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
security_scope.ts is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Novelty Skeptic. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// security_scope.ts
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Task success rate", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
security_scope.ts becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 29: privacy_filter.ts

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
privacy_filter.ts is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Entropy Governor. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// privacy_filter.ts
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Verifier pass rate", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
privacy_filter.ts becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 30: fallback_ladder.ts

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
fallback_ladder.ts is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Causal Modeler. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// fallback_ladder.ts
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Patch acceptance rate", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
fallback_ladder.ts becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 31: rate_limit_manager.ts

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
rate_limit_manager.ts is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Simulation Designer. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// rate_limit_manager.ts
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Execution success", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
rate_limit_manager.ts becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 32: provider_health.ts

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
provider_health.ts is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Benchmark Judge. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// provider_health.ts
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Regression-free change", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
provider_health.ts becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 33: nvidia_nim_deploy/

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
nvidia_nim_deploy/ is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Human Liaison. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// nvidia_nim_deploy/
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Claim support ratio", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
nvidia_nim_deploy/ becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 34: vllm_deploy/

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
vllm_deploy/ is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Rollback Manager. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// vllm_deploy/
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Contradiction density", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
vllm_deploy/ becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 35: tgi_deploy/

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
tgi_deploy/ is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Incident Commander. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// tgi_deploy/
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Assumption visibility", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
tgi_deploy/ becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 36: observability_dashboard/

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
observability_dashboard/ is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Sovereignty Auditor. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// observability_dashboard/
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Uncertainty calibration", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
observability_dashboard/ becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 37: trace_viewer/

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
trace_viewer/ is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Provider Manager. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// trace_viewer/
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Route confidence accuracy", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
trace_viewer/ becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 38: bench_runner.ts

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
bench_runner.ts is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Model Pool Curator. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// bench_runner.ts
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Latency per solved subtask", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
bench_runner.ts becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 39: postmortem_writer.ts

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
postmortem_writer.ts is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Fusion Synthesizer. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// postmortem_writer.ts
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Cost per verified answer", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
postmortem_writer.ts becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 40: route_memory.sqlite

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
route_memory.sqlite is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Style Guardian. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// route_memory.sqlite
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Free-tier fallback rate", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
route_memory.sqlite becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 41: capability_cards.yaml

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
capability_cards.yaml is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: State Reconciler. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// capability_cards.yaml
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Provider failure recovery", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
capability_cards.yaml becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 42: domain_adapters/

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
domain_adapters/ is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Policy Monitor. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// domain_adapters/
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Tool-call precision", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
domain_adapters/ becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 43: policy_envelopes/

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
policy_envelopes/ is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Open-Weight Migrator. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// policy_envelopes/
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Tool-call recall", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
policy_envelopes/ becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 44: human_override.ts

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
human_override.ts is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: NIM Operator. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// human_override.ts
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Schema validity", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
human_override.ts becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 45: incident_runbook.md

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
incident_runbook.md is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: OpenRouter Gateway. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// incident_runbook.md
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["JSON repair frequency", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
incident_runbook.md becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 46: deployment_checklist.md

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
deployment_checklist.md is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Latency Governor. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// deployment_checklist.md
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Long-context retention", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
deployment_checklist.md becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 47: sovereignty_audit.md

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
sovereignty_audit.md is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Router. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// sovereignty_audit.md
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Persona stability", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
sovereignty_audit.md becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 48: skill_ontology.ttl

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
skill_ontology.ttl is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Planner. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// skill_ontology.ttl
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Domain term accuracy", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
skill_ontology.ttl becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 49: hyperloop_compiler.ts

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
hyperloop_compiler.ts is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Decomposer. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// hyperloop_compiler.ts
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Memory reuse precision", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
hyperloop_compiler.ts becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Implementation Component 50: model_registry.json

Concrete file, module, or subsystem for turning the KB into Hermes code.

PURPOSE
model_registry.json is a required implementation artifact in the Hermes-Fugu Fusion stack. It should encode a policy that would otherwise remain hidden in prompts. The goal is to move from inspiration to reproducible execution.

OWNER AND INTERFACE
Primary owner role: Researcher. The component should expose a typed interface, structured logs, configuration boundaries, and a test fixture. It must be usable by the router without embedding provider-specific assumptions in the core logic.

```
// model_registry.json
export interface HermesModule {
  name: string;
  version: string;
  accepts(input: TaskState): boolean;
  run(input: TaskState, ctx: HermesContext): Promise<ModuleResult>;
  metrics: ["Novelty-to-validity ratio", "latency", "failure_rate"];
}
```

BUILD CHECKLIST
- Define inputs and outputs before prompts.
- Attach telemetry from the first day.
- Add a failure fixture and a golden fixture.
- Record provider and model separately.
- Connect successful traces to memory distillation.

INTEGRATION NOTE
model_registry.json becomes revolutionary only when it feeds the loop: route decisions improve, court failures are remembered, and world-effect evidence changes future execution. Static code without telemetry is not Hermes; it is only plumbing.

Security / Anti-Hijack Pattern 1: Prompt injection quarantine

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Prompt injection quarantine prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Prompt injection quarantine protects Novelty Verification by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Prompt injection quarantine")
  return allowed_route
```

FAILURE MODE
Prompt injection quarantine fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 2: Tool permission minimization

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Tool permission minimization prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Tool permission minimization protects Inventive Entropy Control by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Tool permission minimization")
  return allowed_route
```

FAILURE MODE
Tool permission minimization fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 3: Secret redaction before routing

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Secret redaction before routing prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Secret redaction before routing protects Causal Path Modeling by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Secret redaction before routing")
  return allowed_route
```

FAILURE MODE
Secret redaction before routing fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 4: Provider privacy tiering

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Provider privacy tiering prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Provider privacy tiering protects World-Effect Measurement by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Provider privacy tiering")
  return allowed_route
```

FAILURE MODE
Provider privacy tiering fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 5: Hosted-free data boundary

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Hosted-free data boundary prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Hosted-free data boundary protects Output Fusion by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Hosted-free data boundary")
  return allowed_route
```

FAILURE MODE
Hosted-free data boundary fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 6: Local-only sensitive route

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Local-only sensitive route prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Local-only sensitive route protects Model Disagreement Mining by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Local-only sensitive route")
  return allowed_route
```

FAILURE MODE
Local-only sensitive route fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 7: Repository credential scan

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Repository credential scan prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Repository credential scan protects Critic Court Operation by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Repository credential scan")
  return allowed_route
```

FAILURE MODE
Repository credential scan fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 8: Command allowlist

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Command allowlist prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Command allowlist protects Prompt Contracting by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Command allowlist")
  return allowed_route
```

FAILURE MODE
Command allowlist fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 9: Sandboxed execution

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Sandboxed execution prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Sandboxed execution protects JSON Schema Governance by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Sandboxed execution")
  return allowed_route
```

FAILURE MODE
Sandboxed execution fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 10: No-destructive-action default

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION

No-destructive-action default prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME

In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. No-destructive-action default protects Structured Output Repair by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS

- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("No-destructive-action default")
  return allowed_route
```

FAILURE MODE

No-destructive-action default fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 11: Human approval gate

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Human approval gate prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Human approval gate protects MCP Tool Selection by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Human approval gate")
  return allowed_route
```

FAILURE MODE
Human approval gate fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 12: Scope statement enforcement

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Scope statement enforcement prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Scope statement enforcement protects WordPress Builder Planning by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Scope statement enforcement")
  return allowed_route
```

FAILURE MODE
Scope statement enforcement fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 13: Evidence-preserving cyber report

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Evidence-preserving cyber report prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Evidence-preserving cyber report protects VPS Deployment Reasoning by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Evidence-preserving cyber report")
  return allowed_route
```

FAILURE MODE
Evidence-preserving cyber report fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 14: Rate-limit abuse control

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Rate-limit abuse control prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Rate-limit abuse control protects cPanel Automation Planning by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Rate-limit abuse control")
  return allowed_route
```

FAILURE MODE
Rate-limit abuse control fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 15: Model output taint tracking

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Model output taint tracking prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Model output taint tracking protects CI/CD Looping by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Model output taint tracking")
  return allowed_route
```

FAILURE MODE
Model output taint tracking fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 16: Cross-model leakage control

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Cross-model leakage control prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Cross-model leakage control protects Rollback Design by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Cross-model leakage control")
  return allowed_route
```

FAILURE MODE
Cross-model leakage control fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 17: Memory poisoning detection

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Memory poisoning detection prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Memory poisoning detection protects Observability Design by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Memory poisoning detection")
  return allowed_route
```

FAILURE MODE
Memory poisoning detection fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 18: GraphRAG source trust scoring

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
GraphRAG source trust scoring prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. GraphRAG source trust scoring protects Telemetry Summarization by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("GraphRAG source trust scoring")
  return allowed_route
```

FAILURE MODE
GraphRAG source trust scoring fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 19: Malicious plugin scan

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Malicious plugin scan prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Malicious plugin scan protects Routing Memory Update by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Malicious plugin scan")
  return allowed_route
```

FAILURE MODE
Malicious plugin scan fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 20: Dependency vulnerability check

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION

Dependency vulnerability check prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME

In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Dependency vulnerability check protects Agent Role Assignment by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS

- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Dependency vulnerability check")
  return allowed_route
```

FAILURE MODE

Dependency vulnerability check fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 21: WordPress hardening loop

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
WordPress hardening loop prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. WordPress hardening loop protects Parallel Worker Management by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("WordPress hardening loop")
  return allowed_route
```

FAILURE MODE
WordPress hardening loop fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 22: cPanel least-privilege deployment

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
cPanel least-privilege deployment prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. cPanel least-privilege deployment protects Recursive Delegation Control by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("cPanel least-privilege deployment")
  return allowed_route
```

FAILURE MODE
cPanel least-privilege deployment fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 23: API key vault boundary

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
API key vault boundary prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. API key vault boundary protects Cost Budgeting by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("API key vault boundary")
  return allowed_route
```

FAILURE MODE
API key vault boundary fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 24: Rollback before mutation

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Rollback before mutation prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Rollback before mutation protects Rate Limit Handling by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Rollback before mutation")
  return allowed_route
```

FAILURE MODE
Rollback before mutation fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 25: Untrusted attachment handling

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION

Untrusted attachment handling prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME

In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Untrusted attachment handling protects Free-Tier Reliability Management by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS

- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Untrusted attachment handling")
  return allowed_route
```

FAILURE MODE

Untrusted attachment handling fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 26: Vision prompt injection scan

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Vision prompt injection scan prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Vision prompt injection scan protects Open-Weight Migration Planning by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Vision prompt injection scan")
  return allowed_route
```

FAILURE MODE
Vision prompt injection scan fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 27: HTML/script neutralization

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
HTML/script neutralization prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. HTML/script neutralization protects NIM Self-Hosting Transition by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("HTML/script neutralization")
  return allowed_route
```

FAILURE MODE
HTML/script neutralization fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 28: License compliance check

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION

License compliance check prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME

In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. License compliance check protects Evaluation Harness Design by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS

- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("License compliance check")
  return allowed_route
```

FAILURE MODE

License compliance check fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 29: Open-weight supply chain audit

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Open-weight supply chain audit prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Open-weight supply chain audit protects Synthetic Task Generation by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Open-weight supply chain audit")
  return allowed_route
```

FAILURE MODE
Open-weight supply chain audit fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 30: NIM container hardening

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
NIM container hardening prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. NIM container hardening protects Golden-Set Curation by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("NIM container hardening")
  return allowed_route
```

FAILURE MODE
NIM container hardening fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 31: Telemetry privacy minimization

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Telemetry privacy minimization prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Telemetry privacy minimization protects Regression Detection by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Telemetry privacy minimization")
  return allowed_route
```

FAILURE MODE
Telemetry privacy minimization fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 32: Audit log immutability

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Audit log immutability prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Audit log immutability protects Agent Persona Stability by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Audit log immutability")
  return allowed_route
```

FAILURE MODE
Audit log immutability fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 33: Policy envelope tests

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Policy envelope tests prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Policy envelope tests protects Multi-Turn State Control by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Policy envelope tests")
  return allowed_route
```

FAILURE MODE
Policy envelope tests fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 34: Red-team simulation

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Red-team simulation prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Red-team simulation protects Dialogue Repair by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Red-team simulation")
  return allowed_route
```

FAILURE MODE
Red-team simulation fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 35: Safety exception log

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Safety exception log prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Safety exception log protects Scientific Source Mapping by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Safety exception log")
  return allowed_route
```

FAILURE MODE
Safety exception log fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 36: Capability abuse detector

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Capability abuse detector prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Capability abuse detector protects Patent/Literature Investigation by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Capability abuse detector")
  return allowed_route
```

FAILURE MODE
Capability abuse detector fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 37: Autonomy clamp

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION

Autonomy clamp prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME

In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Autonomy clamp protects Security Report Assembly by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS

- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Autonomy clamp")
  return allowed_route
```

FAILURE MODE

Autonomy clamp fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 38: External action dry-run

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
External action dry-run prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. External action dry-run protects Cyber Recon Boundary Control by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("External action dry-run")
  return allowed_route
```

FAILURE MODE
External action dry-run fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 39: Production kill switch

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Production kill switch prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Production kill switch protects Mathematical Re-derivation by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Production kill switch")
  return allowed_route
```

FAILURE MODE
Production kill switch fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Security / Anti-Hijack Pattern 40: Incident postmortem lock

Hardening the open-router, hosted-free, and multi-agent orchestration surface.

DEFINITION
Incident postmortem lock prevents Hermes from turning model fusion into uncontrolled data movement or action execution. Security is not a layer after intelligence; it is part of routing. The router must know what cannot be sent, what cannot be executed, and when human approval is mandatory.

MOUSSAIF ANTI-HIJACK FRAME
In vascular terms, hijack is a false vessel: the system routes attention, secrets, or agency into a path that does not serve the real goal. Incident postmortem lock protects Architecture Stress Testing by keeping difference extraction aligned with scope, evidence, and permission.

CONTROLS
- Classify data before provider dispatch.
- Strip secrets and credentials before hosted-free routes.
- Use local/self-hosted route for private repositories or client data.
- Dry-run destructive operations and require explicit approval.
- Log every external action with reason, tool, and rollback path.

```
security_gate(task):
  if contains_secret(task.context): return LOCAL_ONLY
  if action_is_destructive(task): require_human_approval()
  if provider_tier == HOSTED_FREE: redact_sensitive_context()
  enforce_scope("Incident postmortem lock")
  return allowed_route
```

FAILURE MODE
Incident postmortem lock fails when the system treats model quality as permission. A brilliant model has no extra right to see secrets or perform actions. Capability and authority are separate variables.

Training / Evolution Method 1: Routing imitation from expert traces

How Hermes can move from engineered routing to learned orchestration.

TRAINING DEFINITION
Routing imitation from expert traces is a method for improving the orchestrator itself. The target is not a bigger worker model. The target is a better conductor: a router that knows when to call one model, when to form a team, how to assign roles, when to verify, and when to stop.

RELATION TO FUGU
Fugu shows the direction: learned orchestration over agent pools. Hermes can begin with transparent rule-based routing, then distill traces into a smaller router, then use preference data and evaluation results to approach learned scaffolding.

TRAINING DATA FIELDS
- Task text and task family.
- Chosen pattern: ReAct scaffold route.
- Workers, roles, prompts, tools, context sizes, and provider tiers.
- Court verdict, final answer, user/world outcome, and latency.
- Target metric: Rollback readiness.

```
trace -> score -> preference_pair -> router_update
positive_trace = route where Rollback readiness improved and verifier_pass == true
negative_trace = route where contradiction_density rose or action failed
train_router({positive_trace, negative_trace}, method="Routing imitation from expe
  rt traces")
```

GUARDRAIL
A learned router must remain inspectable at the policy layer. Even if the route selector becomes neural, the privacy boundary, tool permissions, human gates, and source-rank rules must stay explicit.

Training / Evolution Method 2: Pairwise route preference learning

How Hermes can move from engineered routing to learned orchestration.

TRAINING DEFINITION
Pairwise route preference learning is a method for improving the orchestrator itself. The target is not a bigger worker model. The target is a better conductor: a router that knows when to call one model, when to form a team, how to assign roles, when to verify, and when to stop.

RELATION TO FUGU
Fugu shows the direction: learned orchestration over agent pools. Hermes can begin with transparent rule-based routing, then distill traces into a smaller router, then use preference data and evaluation results to approach learned scaffolding.

TRAINING DATA FIELDS
- Task text and task family.
- Chosen pattern: Tree-search route.
- Workers, roles, prompts, tools, context sizes, and provider tiers.
- Court verdict, final answer, user/world outcome, and latency.
- Target metric: Production incident count.

```
trace -> score -> preference_pair -> router_update
positive_trace = route where Production incident count improved and verifier_pass
  == true
negative_trace = route where contradiction_density rose or action failed
train_router({positive_trace, negative_trace}, method="Pairwise route preference l
  earning")
```

GUARDRAIL
A learned router must remain inspectable at the policy layer. Even if the route selector becomes neural, the privacy boundary, tool permissions, human gates, and source-rank rules must stay explicit.

Training / Evolution Method 3: Verifier reward shaping

How Hermes can move from engineered routing to learned orchestration.

TRAINING DEFINITION
Verifier reward shaping is a method for improving the orchestrator itself. The target is not a bigger worker model. The target is a better conductor: a router that knows when to call one model, when to form a team, how to assign roles, when to verify, and when to stop.

RELATION TO FUGU
Fugu shows the direction: learned orchestration over agent pools. Hermes can begin with transparent rule-based routing, then distill traces into a smaller router, then use preference data and evaluation results to approach learned scaffolding.

TRAINING DATA FIELDS
- Task text and task family.
- Chosen pattern: GraphRAG constrained route.
- Workers, roles, prompts, tools, context sizes, and provider tiers.
- Court verdict, final answer, user/world outcome, and latency.
- Target metric: Tool failure recovery time.

```
trace -> score -> preference_pair -> router_update
positive_trace = route where Tool failure recovery time improved and verifier_pass
   == true
negative_trace = route where contradiction_density rose or action failed
train_router({positive_trace, negative_trace}, method="Verifier reward shaping")
```

GUARDRAIL
A learned router must remain inspectable at the policy layer. Even if the route selector becomes neural, the privacy boundary, tool permissions, human gates, and source-rank rules must stay explicit.

Training / Evolution Method 4: Evolutionary scaffold search

How Hermes can move from engineered routing to learned orchestration.

TRAINING DEFINITION

Evolutionary scaffold search is a method for improving the orchestrator itself. The target is not a bigger worker model. The target is a better conductor: a router that knows when to call one model, when to form a team, how to assign roles, when to verify, and when to stop.

RELATION TO FUGU

Fugu shows the direction: learned orchestration over agent pools. Hermes can begin with transparent rule-based routing, then distill traces into a smaller router, then use preference data and evaluation results to approach learned scaffolding.

TRAINING DATA FIELDS

- Task text and task family.
- Chosen pattern: MCP tool-aware route.
- Workers, roles, prompts, tools, context sizes, and provider tiers.
- Court verdict, final answer, user/world outcome, and latency.
- Target metric: Autonomy level match.

```
trace -> score -> preference_pair -> router_update
positive_trace = route where Autonomy level match improved and verifier_pass == tr
  ue
negative_trace = route where contradiction_density rose or action failed
train_router({positive_trace, negative_trace}, method="Evolutionary scaffold searc
  h")
```

GUARDRAIL

A learned router must remain inspectable at the policy layer. Even if the route selector becomes neural, the privacy boundary, tool permissions, human gates, and source-rank rules must stay explicit.

Training / Evolution Method 5: Synthetic task curricula

How Hermes can move from engineered routing to learned orchestration.

TRAINING DEFINITION
Synthetic task curricula is a method for improving the orchestrator itself. The target is not a bigger worker model. The target is a better conductor: a router that knows when to call one model, when to form a team, how to assign roles, when to verify, and when to stop.

RELATION TO FUGU
Fugu shows the direction: learned orchestration over agent pools. Hermes can begin with transparent rule-based routing, then distill traces into a smaller router, then use preference data and evaluation results to approach learned scaffolding.

TRAINING DATA FIELDS
- Task text and task family.
- Chosen pattern: Repository-aware route.
- Workers, roles, prompts, tools, context sizes, and provider tiers.
- Court verdict, final answer, user/world outcome, and latency.
- Target metric: SLA compliance.

```
trace -> score -> preference_pair -> router_update
positive_trace = route where SLA compliance improved and verifier_pass == true
negative_trace = route where contradiction_density rose or action failed
train_router({positive_trace, negative_trace}, method="Synthetic task curricula")
```

GUARDRAIL
A learned router must remain inspectable at the policy layer. Even if the route selector becomes neural, the privacy boundary, tool permissions, human gates, and source-rank rules must stay explicit.

Training / Evolution Method 6: Difficulty-aware batching

How Hermes can move from engineered routing to learned orchestration.

TRAINING DEFINITION
Difficulty-aware batching is a method for improving the orchestrator itself. The target is not a bigger worker model. The target is a better conductor: a router that knows when to call one model, when to form a team, how to assign roles, when to verify, and when to stop.

RELATION TO FUGU
Fugu shows the direction: learned orchestration over agent pools. Hermes can begin with transparent rule-based routing, then distill traces into a smaller router, then use preference data and evaluation results to approach learned scaffolding.

TRAINING DATA FIELDS
- Task text and task family.
- Chosen pattern: Security-scoped route.
- Workers, roles, prompts, tools, context sizes, and provider tiers.
- Court verdict, final answer, user/world outcome, and latency.
- Target metric: MCD score predictive power.

```
trace -> score -> preference_pair -> router_update
positive_trace = route where MCD score predictive power improved and verifier_pass
   == true
negative_trace = route where contradiction_density rose or action failed
train_router({positive_trace, negative_trace}, method="Difficulty-aware batching")
```

GUARDRAIL
A learned router must remain inspectable at the policy layer. Even if the route selector becomes neural, the privacy boundary, tool permissions, human gates, and source-rank rules must stay explicit.

Training / Evolution Method 7: Agent role mutation

How Hermes can move from engineered routing to learned orchestration.

TRAINING DEFINITION
Agent role mutation is a method for improving the orchestrator itself. The target is not a bigger worker model. The target is a better conductor: a router that knows when to call one model, when to form a team, how to assign roles, when to verify, and when to stop.

RELATION TO FUGU
Fugu shows the direction: learned orchestration over agent pools. Hermes can begin with transparent rule-based routing, then distill traces into a smaller router, then use preference data and evaluation results to approach learned scaffolding.

TRAINING DATA FIELDS
- Task text and task family.
- Chosen pattern: Privacy-tier route.
- Workers, roles, prompts, tools, context sizes, and provider tiers.
- Court verdict, final answer, user/world outcome, and latency.
- Target metric: Vascular compute efficiency.

```
trace -> score -> preference_pair -> router_update
positive_trace = route where Vascular compute efficiency improved and verifier_pas
  s == true
negative_trace = route where contradiction_density rose or action failed
train_router({positive_trace, negative_trace}, method="Agent role mutation")
```

GUARDRAIL
A learned router must remain inspectable at the policy layer. Even if the route selector becomes neural, the privacy boundary, tool permissions, human gates, and source-rank rules must stay explicit.

Training / Evolution Method 8: Prompt contract mutation

How Hermes can move from engineered routing to learned orchestration.

TRAINING DEFINITION

Prompt contract mutation is a method for improving the orchestrator itself. The target is not a bigger worker model. The target is a better conductor: a router that knows when to call one model, when to form a team, how to assign roles, when to verify, and when to stop.

RELATION TO FUGU

Fugu shows the direction: learned orchestration over agent pools. Hermes can begin with transparent rule-based routing, then distill traces into a smaller router, then use preference data and evaluation results to approach learned scaffolding.

TRAINING DATA FIELDS

- Task text and task family.
- Chosen pattern: Context-fragment route.
- Workers, roles, prompts, tools, context sizes, and provider tiers.
- Court verdict, final answer, user/world outcome, and latency.
- Target metric: Agency selection regret.

```
trace -> score -> preference_pair -> router_update
positive_trace = route where Agency selection regret improved and verifier_pass ==
   true
negative_trace = route where contradiction_density rose or action failed
train_router({positive_trace, negative_trace}, method="Prompt contract mutation")
```

GUARDRAIL

A learned router must remain inspectable at the policy layer. Even if the route selector becomes neural, the privacy boundary, tool permissions, human gates, and source-rank rules must stay explicit.

Training / Evolution Method 9: Telemetry replay training

How Hermes can move from engineered routing to learned orchestration.

TRAINING DEFINITION
Telemetry replay training is a method for improving the orchestrator itself. The target is not a bigger worker model. The target is a better conductor: a router that knows when to call one model, when to form a team, how to assign roles, when to verify, and when to stop.

RELATION TO FUGU
Fugu shows the direction: learned orchestration over agent pools. Hermes can begin with transparent rule-based routing, then distill traces into a smaller router, then use preference data and evaluation results to approach learned scaffolding.

TRAINING DATA FIELDS
- Task text and task family.
- Chosen pattern: Compression-before-route.
- Workers, roles, prompts, tools, context sizes, and provider tiers.
- Court verdict, final answer, user/world outcome, and latency.
- Target metric: Self-reconstruction gain.

```
trace -> score -> preference_pair -> router_update
positive_trace = route where Self-reconstruction gain improved and verifier_pass =
  = true
negative_trace = route where contradiction_density rose or action failed
train_router({positive_trace, negative_trace}, method="Telemetry replay training")
```

GUARDRAIL
A learned router must remain inspectable at the policy layer. Even if the route selector becomes neural, the privacy boundary, tool permissions, human gates, and source-rank rules must stay explicit.

Training / Evolution Method 10: Golden-set regression loops

How Hermes can move from engineered routing to learned orchestration.

TRAINING DEFINITION

Golden-set regression loops is a method for improving the orchestrator itself. The target is not a bigger worker model. The target is a better conductor: a router that knows when to call one model, when to form a team, how to assign roles, when to verify, and when to stop.

RELATION TO FUGU

Fugu shows the direction: learned orchestration over agent pools. Hermes can begin with transparent rule-based routing, then distill traces into a smaller router, then use preference data and evaluation results to approach learned scaffolding.

TRAINING DATA FIELDS

- Task text and task family.
- Chosen pattern: Model-pool refresh route.
- Workers, roles, prompts, tools, context sizes, and provider tiers.
- Court verdict, final answer, user/world outcome, and latency.
- Target metric: Hyperloop closure rate.

```
trace -> score -> preference_pair -> router_update
positive_trace = route where Hyperloop closure rate improved and verifier_pass ==
  true
negative_trace = route where contradiction_density rose or action failed
train_router({positive_trace, negative_trace}, method="Golden-set regression loops
  ")
```

GUARDRAIL

A learned router must remain inspectable at the policy layer. Even if the route selector becomes neural, the privacy boundary, tool permissions, human gates, and source-rank rules must stay explicit.

Training / Evolution Method 11: Failure-case distillation

How Hermes can move from engineered routing to learned orchestration.

TRAINING DEFINITION
Failure-case distillation is a method for improving the orchestrator itself. The target is not a bigger worker model. The target is a better conductor: a router that knows when to call one model, when to form a team, how to assign roles, when to verify, and when to stop.

RELATION TO FUGU
Fugu shows the direction: learned orchestration over agent pools. Hermes can begin with transparent rule-based routing, then distill traces into a smaller router, then use preference data and evaluation results to approach learned scaffolding.

TRAINING DATA FIELDS
- Task text and task family.
- Chosen pattern: Capability-card route.
- Workers, roles, prompts, tools, context sizes, and provider tiers.
- Court verdict, final answer, user/world outcome, and latency.
- Target metric: Hyperskill transfer rate.

```
trace -> score -> preference_pair -> router_update
positive_trace = route where Hyperskill transfer rate improved and verifier_pass =
  = true
negative_trace = route where contradiction_density rose or action failed
train_router({positive_trace, negative_trace}, method="Failure-case distillation")
```

GUARDRAIL
A learned router must remain inspectable at the policy layer. Even if the route selector becomes neural, the privacy boundary, tool permissions, human gates, and source-rank rules must stay explicit.

Training / Evolution Method 12: Disagreement mining

How Hermes can move from engineered routing to learned orchestration.

TRAINING DEFINITION

Disagreement mining is a method for improving the orchestrator itself. The target is not a bigger worker model. The target is a better conductor: a router that knows when to call one model, when to form a team, how to assign roles, when to verify, and when to stop.

RELATION TO FUGU

Fugu shows the direction: learned orchestration over agent pools. Hermes can begin with transparent rule-based routing, then distill traces into a smaller router, then use preference data and evaluation results to approach learned scaffolding.

TRAINING DATA FIELDS

- Task text and task family.
- Chosen pattern: Observed-win-rate route.
- Workers, roles, prompts, tools, context sizes, and provider tiers.
- Court verdict, final answer, user/world outcome, and latency.
- Target metric: Cross-domain generalization.

```
trace -> score -> preference_pair -> router_update
positive_trace = route where Cross-domain generalization improved and verifier_pas
  s == true
negative_trace = route where contradiction_density rose or action failed
train_router({positive_trace, negative_trace}, method="Disagreement mining")
```

GUARDRAIL

A learned router must remain inspectable at the policy layer. Even if the route selector becomes neural, the privacy boundary, tool permissions, human gates, and source-rank rules must stay explicit.

Training / Evolution Method 13: Self-play court battles

How Hermes can move from engineered routing to learned orchestration.

TRAINING DEFINITION
Self-play court battles is a method for improving the orchestrator itself. The target is not a bigger worker model. The target is a better conductor: a router that knows when to call one model, when to form a team, how to assign roles, when to verify, and when to stop.

RELATION TO FUGU
Fugu shows the direction: learned orchestration over agent pools. Hermes can begin with transparent rule-based routing, then distill traces into a smaller router, then use preference data and evaluation results to approach learned scaffolding.

TRAINING DATA FIELDS
- Task text and task family.
- Chosen pattern: Synthetic benchmark route.
- Workers, roles, prompts, tools, context sizes, and provider tiers.
- Court verdict, final answer, user/world outcome, and latency.
- Target metric: Research productivity index.

```
trace -> score -> preference_pair -> router_update
positive_trace = route where Research productivity index improved and verifier_pas
  s == true
negative_trace = route where contradiction_density rose or action failed
train_router({positive_trace, negative_trace}, method="Self-play court battles")
```

GUARDRAIL
A learned router must remain inspectable at the policy layer. Even if the route selector becomes neural, the privacy boundary, tool permissions, human gates, and source-rank rules must stay explicit.

Training / Evolution Method 14: MCD target prediction

How Hermes can move from engineered routing to learned orchestration.

TRAINING DEFINITION
MCD target prediction is a method for improving the orchestrator itself. The target is not a bigger worker model. The target is a better conductor: a router that knows when to call one model, when to form a team, how to assign roles, when to verify, and when to stop.

RELATION TO FUGU
Fugu shows the direction: learned orchestration over agent pools. Hermes can begin with transparent rule-based routing, then distill traces into a smaller router, then use preference data and evaluation results to approach learned scaffolding.

TRAINING DATA FIELDS
- Task text and task family.
- Chosen pattern: Human override route.
- Workers, roles, prompts, tools, context sizes, and provider tiers.
- Court verdict, final answer, user/world outcome, and latency.
- Target metric: Task success rate.

```
trace -> score -> preference_pair -> router_update
positive_trace = route where Task success rate improved and verifier_pass == true
negative_trace = route where contradiction_density rose or action failed
train_router({positive_trace, negative_trace}, method="MCD target prediction")
```

GUARDRAIL
A learned router must remain inspectable at the policy layer. Even if the route selector becomes neural, the privacy boundary, tool permissions, human gates, and source-rank rules must stay explicit.

Training / Evolution Method 15: Vascular allocation training

How Hermes can move from engineered routing to learned orchestration.

TRAINING DEFINITION
Vascular allocation training is a method for improving the orchestrator itself. The target is not a bigger worker model. The target is a better conductor: a router that knows when to call one model, when to form a team, how to assign roles, when to verify, and when to stop.

RELATION TO FUGU
Fugu shows the direction: learned orchestration over agent pools. Hermes can begin with transparent rule-based routing, then distill traces into a smaller router, then use preference data and evaluation results to approach learned scaffolding.

TRAINING DATA FIELDS
- Task text and task family.
- Chosen pattern: Disagreement expansion route.
- Workers, roles, prompts, tools, context sizes, and provider tiers.
- Court verdict, final answer, user/world outcome, and latency.
- Target metric: Verifier pass rate.

```
trace -> score -> preference_pair -> router_update
positive_trace = route where Verifier pass rate improved and verifier_pass == true
negative_trace = route where contradiction_density rose or action failed
train_router({positive_trace, negative_trace}, method="Vascular allocation trainin
  g")
```

GUARDRAIL
A learned router must remain inspectable at the policy layer. Even if the route selector becomes neural, the privacy boundary, tool permissions, human gates, and source-rank rules must stay explicit.

Training / Evolution Method 16: Latency-quality Pareto tuning

How Hermes can move from engineered routing to learned orchestration.

TRAINING DEFINITION
Latency-quality Pareto tuning is a method for improving the orchestrator itself. The target is not a bigger worker model. The target is a better conductor: a router that knows when to call one model, when to form a team, how to assign roles, when to verify, and when to stop.

RELATION TO FUGU
Fugu shows the direction: learned orchestration over agent pools. Hermes can begin with transparent rule-based routing, then distill traces into a smaller router, then use preference data and evaluation results to approach learned scaffolding.

TRAINING DATA FIELDS
- Task text and task family.
- Chosen pattern: Confidence collapse route.
- Workers, roles, prompts, tools, context sizes, and provider tiers.
- Court verdict, final answer, user/world outcome, and latency.
- Target metric: Patch acceptance rate.

```
trace -> score -> preference_pair -> router_update
positive_trace = route where Patch acceptance rate improved and verifier_pass == t
  rue
negative_trace = route where contradiction_density rose or action failed
train_router({positive_trace, negative_trace}, method="Latency-quality Pareto tuni
  ng")
```

GUARDRAIL
A learned router must remain inspectable at the policy layer. Even if the route selector becomes neural, the privacy boundary, tool permissions, human gates, and source-rank rules must stay explicit.

Training / Evolution Method 17: Tool-call curriculum

How Hermes can move from engineered routing to learned orchestration.

TRAINING DEFINITION
Tool-call curriculum is a method for improving the orchestrator itself. The target is not a bigger worker model. The target is a better conductor: a router that knows when to call one model, when to form a team, how to assign roles, when to verify, and when to stop.

RELATION TO FUGU
Fugu shows the direction: learned orchestration over agent pools. Hermes can begin with transparent rule-based routing, then distill traces into a smaller router, then use preference data and evaluation results to approach learned scaffolding.

TRAINING DATA FIELDS
- Task text and task family.
- Chosen pattern: Novelty court route.
- Workers, roles, prompts, tools, context sizes, and provider tiers.
- Court verdict, final answer, user/world outcome, and latency.
- Target metric: Execution success.

```
trace -> score -> preference_pair -> router_update
positive_trace = route where Execution success improved and verifier_pass == true
negative_trace = route where contradiction_density rose or action failed
train_router({positive_trace, negative_trace}, method="Tool-call curriculum")
```

GUARDRAIL
A learned router must remain inspectable at the policy layer. Even if the route selector becomes neural, the privacy boundary, tool permissions, human gates, and source-rank rules must stay explicit.

Training / Evolution Method 18: Domain adapter finetuning

How Hermes can move from engineered routing to learned orchestration.

TRAINING DEFINITION

Domain adapter finetuning is a method for improving the orchestrator itself. The target is not a bigger worker model. The target is a better conductor: a router that knows when to call one model, when to form a team, how to assign roles, when to verify, and when to stop.

RELATION TO FUGU

Fugu shows the direction: learned orchestration over agent pools. Hermes can begin with transparent rule-based routing, then distill traces into a smaller router, then use preference data and evaluation results to approach learned scaffolding.

TRAINING DATA FIELDS

- Task text and task family.
- Chosen pattern: Repair-only route.
- Workers, roles, prompts, tools, context sizes, and provider tiers.
- Court verdict, final answer, user/world outcome, and latency.
- Target metric: Regression-free change.

```
trace -> score -> preference_pair -> router_update
positive_trace = route where Regression-free change improved and verifier_pass ==
  true
negative_trace = route where contradiction_density rose or action failed
train_router({positive_trace, negative_trace}, method="Domain adapter finetuning")
```

GUARDRAIL

A learned router must remain inspectable at the policy layer. Even if the route selector becomes neural, the privacy boundary, tool permissions, human gates, and source-rank rules must stay explicit.

Training / Evolution Method 19: Open-weight router distillation

How Hermes can move from engineered routing to learned orchestration.

TRAINING DEFINITION
Open-weight router distillation is a method for improving the orchestrator itself. The target is not a bigger worker model. The target is a better conductor: a router that knows when to call one model, when to form a team, how to assign roles, when to verify, and when to stop.

RELATION TO FUGU
Fugu shows the direction: learned orchestration over agent pools. Hermes can begin with transparent rule-based routing, then distill traces into a smaller router, then use preference data and evaluation results to approach learned scaffolding.

TRAINING DATA FIELDS
- Task text and task family.
- Chosen pattern: No-tool safe route.
- Workers, roles, prompts, tools, context sizes, and provider tiers.
- Court verdict, final answer, user/world outcome, and latency.
- Target metric: Claim support ratio.

```
trace -> score -> preference_pair -> router_update
positive_trace = route where Claim support ratio improved and verifier_pass == tru
  e
negative_trace = route where contradiction_density rose or action failed
train_router({positive_trace, negative_trace}, method="Open-weight router distilla
  tion")
```

GUARDRAIL
A learned router must remain inspectable at the policy layer. Even if the route selector becomes neural, the privacy boundary, tool permissions, human gates, and source-rank rules must stay explicit.

Training / Evolution Method 20: Human feedback on route shape

How Hermes can move from engineered routing to learned orchestration.

TRAINING DEFINITION

Human feedback on route shape is a method for improving the orchestrator itself. The target is not a bigger worker model. The target is a better conductor: a router that knows when to call one model, when to form a team, how to assign roles, when to verify, and when to stop.

RELATION TO FUGU

Fugu shows the direction: learned orchestration over agent pools. Hermes can begin with transparent rule-based routing, then distill traces into a smaller router, then use preference data and evaluation results to approach learned scaffolding.

TRAINING DATA FIELDS

- Task text and task family.
- Chosen pattern: Local-only route.
- Workers, roles, prompts, tools, context sizes, and provider tiers.
- Court verdict, final answer, user/world outcome, and latency.
- Target metric: Contradiction density.

```
trace -> score -> preference_pair -> router_update
positive_trace = route where Contradiction density improved and verifier_pass == t
  rue
negative_trace = route where contradiction_density rose or action failed
train_router({positive_trace, negative_trace}, method="Human feedback on route sha
  pe")
```

GUARDRAIL

A learned router must remain inspectable at the policy layer. Even if the route selector becomes neural, the privacy boundary, tool permissions, human gates, and source-rank rules must stay explicit.

Deployment Roadmap 1: Prototype architecture

A practical construction path for Hermes-Fugu Fusion.

ROADMAP THESIS

Prototype architecture is part of the transition from idea to operating system. The correct path is not to build every agent at once. Build the smallest route that logs evidence, then add courts, then add memory, then add local sovereignty.

PHASES

- Phase 1 - Hosted-free prototype
- Phase 2 - Deterministic router
- Phase 3 - Courted fusion
- Phase 4 - Domain adapters
- Phase 5 - Evaluation memory
- Phase 6 - Self-hosted/NIM migration
- Phase 7 - Learned routing
- Phase 8 - Production governance

```
minimum_viable_hermes:
  - OpenAI-compatible provider adapter
  - model registry with capability cards
  - route_score(task, model)
  - parallel workers for hard tasks
  - falsification court
  - fusion synthesizer
  - telemetry and memory update
  - security/privacy gate
```

ACCEPTANCE CRITERIA

The roadmap is successful when Hermes can outperform a single fixed model on recurring task families, explain why it routed the way it did, recover from provider failure, protect sensitive data, and improve routing from logged outcomes.

Deployment Roadmap 2: Registry and providers

A practical construction path for Hermes-Fugu Fusion.

ROADMAP THESIS

Registry and providers is part of the transition from idea to operating system. The correct path is not to build every agent at once. Build the smallest route that logs evidence, then add courts, then add memory, then add local sovereignty.

PHASES

- Phase 1 - Hosted-free prototype
- Phase 2 - Deterministic router
- Phase 3 - Courted fusion
- Phase 4 - Domain adapters
- Phase 5 - Evaluation memory
- Phase 6 - Self-hosted/NIM migration
- Phase 7 - Learned routing
- Phase 8 - Production governance

```
minimum_viable_hermes:
  - OpenAI-compatible provider adapter
  - model registry with capability cards
  - route_score(task, model)
  - parallel workers for hard tasks
  - falsification court
  - fusion synthesizer
  - telemetry and memory update
  - security/privacy gate
```

ACCEPTANCE CRITERIA

The roadmap is successful when Hermes can outperform a single fixed model on recurring task families, explain why it routed the way it did, recover from provider failure, protect sensitive data, and improve routing from logged outcomes.

Deployment Roadmap 3: Core routing formula

A practical construction path for Hermes-Fugu Fusion.

ROADMAP THESIS

Core routing formula is part of the transition from idea to operating system. The correct path is not to build every agent at once. Build the smallest route that logs evidence, then add courts, then add memory, then add local sovereignty.

PHASES

- Phase 1 - Hosted-free prototype
- Phase 2 - Deterministic router
- Phase 3 - Courted fusion
- Phase 4 - Domain adapters
- Phase 5 - Evaluation memory
- Phase 6 - Self-hosted/NIM migration
- Phase 7 - Learned routing
- Phase 8 - Production governance

```
minimum_viable_hermes:
  - OpenAI-compatible provider adapter
  - model registry with capability cards
  - route_score(task, model)
  - parallel workers for hard tasks
  - falsification court
  - fusion synthesizer
  - telemetry and memory update
  - security/privacy gate
```

ACCEPTANCE CRITERIA

The roadmap is successful when Hermes can outperform a single fixed model on recurring task families, explain why it routed the way it did, recover from provider failure, protect sensitive data, and improve routing from logged outcomes.

Deployment Roadmap 4: Court and fusion

A practical construction path for Hermes-Fugu Fusion.

ROADMAP THESIS

Court and fusion is part of the transition from idea to operating system. The correct path is not to build every agent at once. Build the smallest route that logs evidence, then add courts, then add memory, then add local sovereignty.

PHASES

- Phase 1 - Hosted-free prototype
- Phase 2 - Deterministic router
- Phase 3 - Courted fusion
- Phase 4 - Domain adapters
- Phase 5 - Evaluation memory
- Phase 6 - Self-hosted/NIM migration
- Phase 7 - Learned routing
- Phase 8 - Production governance

```
minimum_viable_hermes:
  - OpenAI-compatible provider adapter
  - model registry with capability cards
  - route_score(task, model)
  - parallel workers for hard tasks
  - falsification court
  - fusion synthesizer
  - telemetry and memory update
  - security/privacy gate
```

ACCEPTANCE CRITERIA

The roadmap is successful when Hermes can outperform a single fixed model on recurring task families, explain why it routed the way it did, recover from provider failure, protect sensitive data, and improve routing from logged outcomes.

Deployment Roadmap 5: Telemetry and memory

A practical construction path for Hermes-Fugu Fusion.

ROADMAP THESIS

Telemetry and memory is part of the transition from idea to operating system. The correct path is not to build every agent at once. Build the smallest route that logs evidence, then add courts, then add memory, then add local sovereignty.

PHASES

- Phase 1 - Hosted-free prototype
- Phase 2 - Deterministic router
- Phase 3 - Courted fusion
- Phase 4 - Domain adapters
- Phase 5 - Evaluation memory
- Phase 6 - Self-hosted/NIM migration
- Phase 7 - Learned routing
- Phase 8 - Production governance

```
minimum_viable_hermes:
  - OpenAI-compatible provider adapter
  - model registry with capability cards
  - route_score(task, model)
  - parallel workers for hard tasks
  - falsification court
  - fusion synthesizer
  - telemetry and memory update
  - security/privacy gate
```

ACCEPTANCE CRITERIA

The roadmap is successful when Hermes can outperform a single fixed model on recurring task families, explain why it routed the way it did, recover from provider failure, protect sensitive data, and improve routing from logged outcomes.

Deployment Roadmap 6: Evaluation harness

A practical construction path for Hermes-Fugu Fusion.

ROADMAP THESIS

Evaluation harness is part of the transition from idea to operating system. The correct path is not to build every agent at once. Build the smallest route that logs evidence, then add courts, then add memory, then add local sovereignty.

PHASES

- Phase 1 - Hosted-free prototype
- Phase 2 - Deterministic router
- Phase 3 - Courted fusion
- Phase 4 - Domain adapters
- Phase 5 - Evaluation memory
- Phase 6 - Self-hosted/NIM migration
- Phase 7 - Learned routing
- Phase 8 - Production governance

```
minimum_viable_hermes:
  - OpenAI-compatible provider adapter
  - model registry with capability cards
  - route_score(task, model)
  - parallel workers for hard tasks
  - falsification court
  - fusion synthesizer
  - telemetry and memory update
  - security/privacy gate
```

ACCEPTANCE CRITERIA

The roadmap is successful when Hermes can outperform a single fixed model on recurring task families, explain why it routed the way it did, recover from provider failure, protect sensitive data, and improve routing from logged outcomes.

Deployment Roadmap 7: Open-weight migration

A practical construction path for Hermes-Fugu Fusion.

ROADMAP THESIS

Open-weight migration is part of the transition from idea to operating system. The correct path is not to build every agent at once. Build the smallest route that logs evidence, then add courts, then add memory, then add local sovereignty.

PHASES

- Phase 1 - Hosted-free prototype
- Phase 2 - Deterministic router
- Phase 3 - Courted fusion
- Phase 4 - Domain adapters
- Phase 5 - Evaluation memory
- Phase 6 - Self-hosted/NIM migration
- Phase 7 - Learned routing
- Phase 8 - Production governance

```
minimum_viable_hermes:
  - OpenAI-compatible provider adapter
  - model registry with capability cards
  - route_score(task, model)
  - parallel workers for hard tasks
  - falsification court
  - fusion synthesizer
  - telemetry and memory update
  - security/privacy gate
```

ACCEPTANCE CRITERIA

The roadmap is successful when Hermes can outperform a single fixed model on recurring task families, explain why it routed the way it did, recover from provider failure, protect sensitive data, and improve routing from logged outcomes.

Deployment Roadmap 8: Production sovereignty

A practical construction path for Hermes-Fugu Fusion.

ROADMAP THESIS

Production sovereignty is part of the transition from idea to operating system. The correct path is not to build every agent at once. Build the smallest route that logs evidence, then add courts, then add memory, then add local sovereignty.

PHASES

- Phase 1 - Hosted-free prototype
- Phase 2 - Deterministic router
- Phase 3 - Courted fusion
- Phase 4 - Domain adapters
- Phase 5 - Evaluation memory
- Phase 6 - Self-hosted/NIM migration
- Phase 7 - Learned routing
- Phase 8 - Production governance

```
minimum_viable_hermes:
  - OpenAI-compatible provider adapter
  - model registry with capability cards
  - route_score(task, model)
  - parallel workers for hard tasks
  - falsification court
  - fusion synthesizer
  - telemetry and memory update
  - security/privacy gate
```

ACCEPTANCE CRITERIA

The roadmap is successful when Hermes can outperform a single fixed model on recurring task families, explain why it routed the way it did, recover from provider failure, protect sensitive data, and improve routing from logged outcomes.

References and Source Notes 1

Sources used for current factual anchors and project concepts.

CITATION POLICY
This PDF uses fresh public sources only for current external facts about Sakana Fugu, OpenRouter free routing, and NVIDIA NIM. Moussaif/MCAI/MICAGI/neoAGI concepts come from the user project memory and are presented as original design-framework material, not as externally validated benchmark results.

SOURCE
Sakana AI, Sakana Fugu: One Model to Command Them All, June 22 2026. Key facts used: Fugu exposes multi-agent orchestration as a single foundation-model-like API; it handles model selection, delegation, verification, and synthesis internally; Fugu balances latency and Fugu Ultra prioritizes answer quality.

SOURCE
Sakana Fugu Technical Report, arXiv:2606.21228v1, June 19 2026. Key facts used: Fugu is a family of learned orchestrators that build query-adaptive agentic scaffolds over frontier worker models; Fugu can select a single worker for speed and Fugu Ultra composes multi-agent workflows for hard tasks.

References and Source Notes 2

Sources used for current factual anchors and project concepts.

SOURCE
OpenRouter Free AI Models page, updated June 2026. Key facts used: OpenRouter maintains a free-model collection, free capacity can change, and openrouter/free automatically selects from free models according to request requirements.

SOURCE
NVIDIA NIM Large Language Model API docs, June 2026. Key facts used: NIM LLM exposes an OpenAI-compatible inference API backed by vLLM; NVIDIA Build offers free serverless APIs for development and self-hosting paths.

URLS
Sakana Fugu release: https://sakana.ai/fugu-release/ ; Sakana Fugu technical report: https://arxiv.org/abs/2606.21228 ; OpenRouter free models: https://openrouter.ai/collections/free-models ; NVIDIA NIM LLM API: https://docs.nvidia.com/nim/large-language-models/latest/api-reference.html

References and Source Notes 3

Sources used for current factual anchors and project concepts.

PROJECT SOURCE NOTE

Prior Moussaif/MCAI/MICAGI/neoAGI project memory. Key concepts integrated: MCD, Vascular Compute Router, Difference Extraction, Inner Time Engine, Perturbation/Falsification Court, World-Effect Loop, Self-Reconstruction Memory, Selective Assistance, Inventive Entropy, Recursive Demolition Court, Hyperloop Skill Theory, Vascular Knowledge Routing, and AGI-oriented hyperskills.

INTEGRATED INVENTED VOCABULARY

Hyperskill, Hyperloop Skill Theory, Vascular Knowledge Routing, MCD-Fusion Gate, Vascular Compute Router, Selective Assistance, Recursive Demolition Court, Causal Novelty Engine, Inventive Entropy Governance, World-Effect Loop, Self-Reconstruction Memory, and Hermes-Fugu Fusion Router.

References and Source Notes 4

Sources used for current factual anchors and project concepts.

CLOSING TECHNICAL DOCTRINE

The final doctrine is simple: intelligence is not only a model parameter count. It is governed orchestration under evidence, constraint, memory, and action. Hermes should therefore be built as a living router-court-fusion-memory system. Every hyperskill must close a loop. Every loop must leave evidence. Every evidence trace must improve routing.

FINAL BUILD COMMAND

- Create the model registry.
- Implement route scoring.
- Add worker roles.
- Add courted fusion.
- Add telemetry and memory.
- Run golden tasks.
- Move sensitive paths to self-hosted or local open weights.
- Train the router from traces.

```
Hermes = Difference -> MCD -> Route -> Scaffold -> Workers -> Court -> Fusion -> W
  orld Effect -> Self-Reconstruction
```