من الإغلاق النظري إلى نواة الفتح التنفيذي

MOUSSAIFXT Kernel Deployment · AL RAZZI HyperEngineering · Computer & Browser Use · Polyglot SWE · Linux/PowerShell Control · Fattah Cognitive Layer

تنتقل هذه الورقة من الإغلاق النظري إلى **الإغلاق التنفيذي**: كيف تتحول MUT/MUG وMCD/QMCD/MCAI/HAKIM وHyperKnowledge وHyperSkill/Ability/Loop وEVCF/MCAS وWorldEffect إلى نواة تشغيل فعلية داخل **MOUSSAIFXT** و**AL RAZZI OS**، وكيف يصبح التحكم في الحاسوب والمتصفح، والهندسة البرمجية متعددة اللغات، وPowerShell/Linux، والأدوات المفتوحة المصدر، جزءاً من منظومة واحدة ذات حاكمية وأدلة واسترجاع وتطور ذاتي محكوم. الغاية هي بناء **Agentic-AGI-oriented engineering kernel** قابل للاختبار، لا إعلان تحقق AGI.

السلسلة
Paper IV - Applied Closure & Kernel Deployment - September 2026

المؤلف / البرنامج
Zouhair Moussaif - زهير مساعف

الهدف
Production-grade kernel blueprint + falsifiable deployment program

التنسيق
A4 · Arabic RTL · Amiri · Dense engineering research

منهج الادعاء: «ذكاء فتّاح» و«Agentic AGI» هنا أسماء لبرنامج هندسي واختبارات تكيفية. لا تُرفع أي طبقة إلى مرتبة القدرة المثبتة قبل recovery، ablation، benchmark، tests وverified world effects.

الملخص التنفيذي: الإغلاق التنفيذي فوق الإغلاق النظري

Executive abstract and applied synthesis

أغلقت الورقة الثالثة القاموس النظري حول مبدأ **الإغلاق الحاكمي السببي**: الذكاء الوكيلي لا يكتمل عند reasoning أو tool call، بل عند غلق المسافة بين حالة مطلوبة وحالة مرصودة عبر معرفة وقدرة وفعل ودليل وتحقيق وذاكرة. هذه الورقة تضيف الحلقة الناقصة: **ما شكل النواة التي تنفّذ ذلك يومياً على Windows وLinux والويب والمستودعات البرمجية؟**

المقترح هو اعتبار **AL RAZZI** harness سيادياً يحفظ Mission/World/Evidence state، و**MOUSSAIFXT** fabric للتنفيذ العابر للمضيفين، و**Kernel Deployment Plane** طبقة تربط النظريات بعقود عملية: VerificationAdapter، LanguageAdapter، ComputerEnvironment، ToolContract، HostAdapter، ProviderAdapter، SecureExecutionPlane. لا يصبح أي إطار خارجي - Hermes أو Prime Agent أو OpenCode أو OpenHands أو Browser Use - هو النظام؛ بل يتحول إلى provider أو plugin أو مصدر patterns تحت سيادة الحالة الموحدة.

$$\text{Verified Intelligence Gain} = \text{Useful World } \Delta \times \text{Evidence} \times \text{Transfer} \times \text{Recoverability} / (\text{Cost} + \text{Risk} + \text{Coordination})$$

المساهمة المركزية: تعريف «HyperEngineering Kernel» لا كحزمة tools، بل كجهاز يختار أقل تمثيل كافٍ للمشكلة، وأفضل أداة/لغة/مضيف، وينفذ داخل حدود قابلة للعكس، ثم يقرأ الحالة بعد الفعل ويحول الخبرة إلى HyperKnowledge أو HyperSkill Candidate.

العقد التنفيذي	نمط الفشل	اختبار الإغلاق
تحويل الملخص التنفيذي: الإغلاق التنفيذي فوق الإغلاق النظري إلى state/schema/event قابل للرصد؛ لا يبقى المفهوم نصاً فقط.	تجاوز طبقة أعلى، state خفية، أو ادعاء اكتمال من دون runtime path.	Ablation مقابل baseline أبسط + restart reconstruction + trace consistency.

موضع الورقة في السلسلة: من Survey إلى Deployment

Paper I-IV closure chain

الجزء	سؤاله	المخرج
Paper I	ماذا تعلمت حالة الفن عن harnesses والذاكرة والتحقق؟	paper frontier map + baselines-25
Paper II	كيف تتحول MUT/MUG إلى mechanisms؟	Applied theory + GTAC/DWR/EVCF/WorldEffect
Paper III	ما المبدأ الموحد وحدود القاموس؟	Governed Causal Closure + falsifiability
Paper IV	كيف تصبح النظرية kernel تتحكم بالحاسوب وتبني software؟	Deployment contracts + HyperEngineering + Fattah cognition

الإغلاق الجديد ليس نظرياً فقط. كل مفهوم في corpus يجب أن يهبط إلى واحد من ستة أشكال تنفيذية: **capability contract، routing policy، state object، verifier/court، execution adapter، أو evolution candidate.** إذا لم يمكن إنزاله إلى واحد منها أو إلى benchmark صريح، يبقى في طبقة البحث ولا يدخل kernel production.

Theory/Corpus → Typed Contract → Kernel Service → Tool/Host → World Readback → Evidence/Court → Reusable Experience

> قاعدة الإثراء: لا نضيف «طبقة» لأن الاسم جذاب؛ نضيفها فقط إذا أصلحت failure class لا تمثله العقود الحالية. وهذا يحمي النظام من التضخم النظري ويحافظ على قابلية التنفيذ.

العقد التنفيذي	نمط الفشل	اختبار الإغلاق
تحويل موضع الورقة في السلسلة: من Survey إلى Deployment إلى state/schema/event قابل للرصد؛ لا يبقى المفهوم نصاً فقط.	تجاوز طبقة أعلى، state خفية، أو ادعاء اكتمال من دون runtime path.	Ablation مقابل baseline أبسط + restart reconstruction + trace consistency.

العقد الكلي: من MUT إلى Kernel Deploy

Theory-to-kernel closure contract

يُعاد تعريف العلاقة بين طبقات المشروع كسلسلة compilation لا كسلسلة وثائق. **MUT** يحدد النظرية والآليات والحدود، **MUG** يحولها إلى governance/runtime policies، **MOUSSAIF AGI OS** يملك الحالة والمعرفة والقدرات والمحاكم، **MOUSSAIFXT** يترجم الدلالة إلى مضيفين وأدوات، **AL RAZZI** يدير mission lifecycle ويُظهر CLI/API، ثم **Kernel Deploy** يضمن أن هذه السلسلة قابلة للبدء والاسترجاع والاختبار.

```
MUT Mechanism
  -> MUG Policy
  -> Capability Contract
  -> Runtime State
  -> MOUSSAIXT Adapter
  -> External Effect
  -> Readback
  -> Evidence
  -> Court
  -> Memory/Evolution
```

$$\text{Deployability}(T) = \text{Mechanism}(T) \times \text{Contract}(T) \times \text{Executability}(T) \times \text{Verifiability}(T) \times \text{Recoverability}(T)$$

علامة الضرب هنا weakest-link heuristic: نظرية عظيمة بلا contract تبقى نظرية، contract بلا executor يبقى spec، executor بلا readback قد ينتج false success، automation بلا rollback تتحول إلى مخاطرة. لذلك معيار النضج في هذا الجزء ليس عدد الملفات بل **نسبة السلاسل المغلقة فعلاً من النظرية إلى الأثر.**

العقد التنفيذي	نمط الفشل	اختبار الإغلاق
تحويل العقد الكلي: من MUT إلى Kernel Deploy إلى state/schema/event قابل للرصد، لا يبقى المفهوم نصاً فقط.	تجاوز طبقة أعلى، state خفية، أو ادعاء اكتمال من دون runtime path.	Ablation مقابل baseline أبسط + trace + restart reconstruction + consistency.

المعمارية التنفيذية الكبرى: خمس طائرات مترابطة

Five-plane executable architecture

Plane	المسؤولية	العناصر
Cognitive Plane	فهم الفجوة واختيار التمثيل والجهد	Q, Δ, MCD/QMCD, HAKIM, hypotheses
Knowledge/Context Plane	معرفة قابلة للعنونة والpaging	HyperKB, EVCF/MCAS, PEK, repo/world models
Capability Plane	تجميع الذكاء في عقود قابلة للاختبار	HyperSkill, HyperAbility, HyperLoop, HyperAgent
Execution Plane	الفعل على الأنظمة الحقيقية	MOUSSAIXT, tools, browser, computer, shells, languages
Assurance/Evolution Plane	التحقق والتعلم المحكوم	Evidence, Courts, WorldEffect, Recovery, HarnessGenome

الفصل بين الطائرات لا يعني عزلاً صلباً. الفعل قد يولد observation يعيد صياغة Δ؛ فشل compiler قد يستدعي HyperKnowledge مختلفاً؛ readback من Linux systemd قد يغير world model؛ Court قد يرجع Mission DAG إلى مرحلة earlier. لكن **الملكية** تبقى واضحة: tool لا يملك model ،mission لا يملك world state، وplugin لا يملك constitution.

> **مبدأ السيادة:** external harnesses are replaceable execution substrates; canonical operational truth يبقى داخل Moussaif/AL RAZZI.

العقد التنفيذي	نمط الفشل	اختبار الإغلاق
تحويل المعمارية التنفيذية الكبرى: خمس طائرات مترابطة إلى state/schema/event قابل للرصد؛ لا يبقى المفهوم نصاً فقط.	تجاوز طبقة أعلى، state خفية، أو ادعاء اكتمال من دون runtime path.	Ablation مقابل baseline أبسط + trace + restart reconstruction consistency.

انضباط الادعاء: «Agentic AGI» هدف اختبار لا ملصق

Claim discipline and maturity

تسمية النظام Agentic-AGI-oriented تعني أننا نستهدف خصائص مثل tool، cross-domain transfer، long-horizon closure، novel-task adaptation، compositional capability، persistent world state، self-correction، generalization، وcontinual learning. لكنها لا تعني أن النظام حقق ذكاء عاماً بمجرد امتلاكه agents كثيرة أو القدرة على التحكم في سطح المكتب.

المرتبة	المعنى	مثال
E2 Artifact	الشفرة/المجلد موجود	ComputerUseAdapter موجود
E3 Specified	العقد واضح وقابل للاختبار	WorldEffect schema + tests
E4 Local Verified	يمر اختباراً حقيقياً	يعدل repo ويصلح test
E5 Comparative	يتفوق أو يضيف قيمة عبر baseline	EVCF أفضل من RAG في held-out tasks
E6 Repeated/Transfer	يتكرر عبر domains/models	HyperSkill ينتقل بين مشاريع

هذه الورقة تستهدف دفع طبقات kernel إلى E3/E4 ووضع برنامج E5. «ذكاء فتّاح» في القسم الأخير يعامل كفرضية هندسية: قدرة النظام على كشف openings عالية الرافعة في فضاء المشكلة والأدوات والفرضيات، ثم اختبارها لا مجرد توليدها.

العقد التنفيذي	نمط الفشل	اختبار الإغلاق
تحويل انضباط الادعاء: «Agentic AGI» هدف اختبار لا ملصق إلى state/schema/event قابل للرصد؛ لا يبقى المفهوم نصاً فقط.	تجاوز طبقة أعلى، state خفية، أو ادعاء اكتمال من دون runtime path.	Ablation مقابل baseline أبسط + restart reconstruction + trace consistency.

Boot Kernel: بدء موثوق قبل أي استقلالية

Deterministic boot and self-test

Kernel قوي لا يبدأ باستدعاء نموذج قبل معرفة البيئة. تسلسل boot المقترح: fingerprint للنظام والمشروع، فتح state store مع migrations، تحميل constitution والregistries، فحص trust، اكتشاف providers/models/tools، اختبار sandboxes، استعادة missions، تهيئة EVCF indexes، فحص Git/worktrees، ثم نشر RuntimeReady event. أي optional subsystem يفشل يجب أن يخفض الحالة إلى DEGRADED لا أن يمنع كل النظام.

```
BOOT
1 environment fingerprint
2 state/migrations
3 constitution + registries
4 project trust
5 provider/model/tool health
6 EVCF/HyperKB health
7 host/sandbox probes
8 restore missions
9 scheduler/event bus
10 RuntimeReady
```

يجب أن يكون boot قابلاً للتكرار والقياس. لا يحمل corpus كاملاً إلى الذاكرة ولا يبدأ agents في الخلفية بلا mission. وتُحفظ نتائج probes مع freshness حتى لا نكرر الفحوص المكلفة بلا حاجة.

العقد التنفيذي	نمط الفشل	اختبار الإغلاق
تحويل Boot Kernel: بدء موثوق قبل أي استقلالية إلى state/schema/event قابل للرصد؛ لا يبقى المفهوم نصاً فقط.	تجاوز طبقة أعلى، state خفية، أو ادعاء اكتمال من دون runtime path.	Ablation مقابل baseline أبسط + restart reconstruction + trace consistency.

Shared Runtime Schema: حالة واحدة فوق نماذج ومضيفين متعددين

Canonical state ownership

الحالة لا تختزل في transcript. النواة تستخدم objects نمطية: WorldModelDelta، CandidateSet، ActiveDifferenceSet، GoverningQuestion، MissionContract، EvolutionCandidate، EvidenceGraph، RemainingGapRegister، ToolInventory، AuthorizationContext. هذه objects تتشاركها /Hermes/Codex/Claude OpenCode عبر adapters من دون أن تُنسخ الحقيقة إلى كل harness.

Object	مالك الحقيقة	ما لا يجوز
MissionContract	AL RAZZI Runtime	أن يغيّره host بصمت
WorldModel	Moussaif state store	اعتبار output النموذج state مرصودة
Authorization	MUG/Authority Engine	اشتقاقها من capability
Evidence	Evidence Ledger	استبدالها بثقة model
Context	EVCF/PEK	اعتبار context window الذاكرة الكاملة

> ميزة هذا الفصل أن تبديل model أو host لا يقطع هوية المهمة؛ يمكن إنهاء node في Hermes واستكمال verify node في Codex/Native بينما يبقى causal trace واحداً.

العقد التنفيذي	نمط الفشل	اختبار الإغلاق
تحويل Shared Runtime Schema: حالة واحدة فوق نماذج ومضيفين متعددين إلى state/schema/event قابل للرصد؛ لا يبقى المفهوم نصاً فقط.	تجاوز طبقة أعلى، state خفية، أو ادعاء اكتمال من دون runtime path.	Ablation مقابل baseline أبسط + restart reconstruction + trace consistency.

Runtime Governance Compiler کے MUG

Constitution, authority and executable policy

في kernel deployment يجب أن تتحول MUG من نصوص إلى قرارات machine-evaluable. كل ActionIntent يحمل sensitivity، reversibility، effect class، target، blast radius، evidence requirement، credential scope. Policy Engine يعيد ALLOW أو DRY_RUN أو REQUIRE_APPROVAL أو BLOCK، ثم ToolGateway يطبق القرار.

Capability ≠ Authority

نقترح بقاء مستويات T0-T4: T0 reasoning/read-public؛ T1 approved reads/simulation/local artifacts؛ T2 تغييرات محلية قابلة للتراجع مع checkpoint/diff/test؛ T3 public/production/message/credential/privacy effects بموافقة صريحة حسب السياسة؛ T4 محظور. الهدف ليس بيروقراطية بل منع انتقال الذكاء التقني إلى سلطة تلقائية.

قاعدة إنتاجية: security boundary يجب أن تكون أسفل agent loop. Prompt safety وحدها لا تكفي؛ حيث أمكن تُستخدم sandbox/network/filesystem policies مستقلة عن model.

العقد التنفيذي	نمط الفشل	اختبار الإغلاق
تحويل MUG كـ Runtime Governance Compiler إلى state/schema/event قابل للرصد؛ لا يبقى المفهوم نصاً فقط.	تجاوز طبقة أعلى، state خفية، أو ادعاء اكتمال من دون runtime path.	Ablation مقابل baseline أبسط + restart reconstruction + trace consistency.

ذكاء فتّاح: تعريف تشغيلي أولي

Fattah intelligence as opening discovery

نعرف **Fattah / Opening Intelligence** هنا كقدرة النظام على اكتشاف فتحات ذات قيمة في فضاء مغلق ظاهرياً: سؤال حاكم أفضل، representation أرخص، أداة حاسمة، hypothesis قابلة للفصل، reuse لخبرة قديمة، أو تغيير صغير يفتح سلسلة كاملة من الحل. الفتح ليس novelty لغوية؛ هو **فتح قابل للإثبات في فضاء الإمكانات.**

$$\text{OpeningValue}(o) \approx \text{Leverage} \times \text{InformationGain} \times \text{Transfer} \times \text{Verifiability} \times \text{Reversibility} / \text{Cost}$$

تعمل FattahLayer فوق MCD/QMCD: MCD يحدد أين تستحق الموارد؛ QMCD يحافظ على بدائل؛ Fattah يبحث عن transformations تغيّر topology للمشكلة، مثل الانتقال من screenshot automation إلى API، من قراءة repo كاملة إلى symbol graph، من debugging يدوي إلى failing-test minimization، أو من reasoning طويل إلى experiment حاسم.

الفرضية القابلة للاختبار: إذا كانت طبقة الفتح مفيدة، يجب أن تخفض actions-per-verified-delta أو تقلل وقت الوصول إلى discriminating evidence مقارنةً بـbaseline planner.

العقد التنفيذي	نمط الفشل	اختبار الإغلاق
تحويل ذكاءٍ فتّاح: تعريف تشغيلي أولي إلى state/schema/event قابل للرصد؛ لا يبقى المفهوم نصاً فقط.	تجاوز طبقة أعلى، state خفية، أو ادعاء اكتمال من دون runtime path.	Ablation مقابل baseline أبسط + trace + restart reconstruction consistency.

السؤال الحاكم كـ Interrupt معرفي

Governing Question in live runtime

Governing Question ليس سؤالاً يكتب مرة في بداية mission. هو object قابل للتغير بعد observations. إذا فشل build، قد يتحول السؤال من «كيف ننفذ feature؟» إلى «هل contract بين A/B module هو سبب regression؟». إذا ظهر provider outage يتحول إلى «أي host بديل يحافظ على semantics؟».

Trigger	إعادة صياغة السؤال
دليل يناقض الخطة	ما الفرضية التي أصبحت أضعف؟
تكلفة تتضخم	ما أقل experiment يفصل البدائل؟
تعطل أداة	هل نحتاج tool آخر أم representation آخر؟
false success	ما readback الذي كان مفقوداً؟
stagnation	أي opening لم نجربه؟

تسجل النواة transitions الحاسمة للسؤال لا chain-of-thought السرية. هذا يكفي لإعادة بناء قرار orchestration ويجعل Fattah layer قابلة للرصد.

العقد التنفيذي	نمط الفشل	مقياس/اختبار
ربط السؤال الحاكم كـ Interrupt معرفي بـContext/Router objects وقرارات قابلة للتفسير.	context bloat، collapse مبكر، أو صرف compute بلا Information Gain.	Useful-context ratio، router regret، remaining-gap reduction، fixed-model ablation.

الفرق النشط: وحدة الدمج بين النظرية والهندسة

Active Difference as universal work unit

Δ هو ما يجعل corpus الكبير قابلاً للاستخدام. في software يمكن أن يكون test failing أو contract missing؛ في browser يمكن أن يكون form field غير مملوء؛ في Linux يمكن أن يكون service state مختلفاً عن desired state؛ في المعرفة يمكن أن يكون contradiction أو stale source. كل Δ له expected، dependencies، severity، class، verifier.

```
ActiveDifference {
 id, observed, desired, type, severity, uncertainty,
 dependencies, candidate_actions, verifier, evidence_refs
}
```

بدلاً من أن يعمل agent على «كل المشروع»، يُنشأ DifferenceGraph ويُفعّل أقل مسار مهارات وأدوات يغطي الفروق المركزية. هذه هي الصيغة التطبيقية لـDifference Before Volume، وتصبح أيضاً أساس progress الحقيقي: التقدم هو انخفاض weighted residual gap لا عدد الرسائل أو tokens.

العقد التنفيذي	نمط الفشل	مقياس/اختبار
ربط الفرق النشط: وحدة الدمج بين النظرية والهندسة بـContext/Router objects وقرارات قابلة للتفسير.	collapse، context bloat مبكر، أو صرف compute بلا Information Gain.	remaining-gap reduction، router regret، Useful-context ratio، fixed-model ablation.

MCD كـ Resource Scheduler لا كمقياس غامض

Calibrated compute allocation

في kernel تُستخدم MCD لتحديد cognitive mode والـcontext budget والـverification depth والـagent population. يجب أن تكون الأبعاد versioned ومقيسة حسب domain، مع router regret: لو اختار route مكلفاً وفشل بينما route أبسط نجح في replay، تسجل النواة ذلك لتحسين المعايرة.

Signal	أثره المحتمل
Uncertainty مرتفع	زيادة retrieval/candidate diversity
Risk مرتفع	زيادة courts/checkpoints
Irreversibility مرتفع	human gate + dry-run
Info gain مرتفع	تشغيل experiment مبكر
Reuse مرتفع	تفعيل HyperSkill مثبتة
Context cost مرتفع	compute-to-context / summarization

$$\text{RouteScore} = \Delta + U + \text{Risk} + \text{InfoGain} + \text{Reuse} + \text{Effect} - \text{ContextCost} - \text{ToolCost}$$

العقد التنفيذي	نمط الفشل	مقياس/اختبار
ربط MCD كـ Resource Scheduler لا كمقياس غامض بـContext/Router objects وقرارات قابلة للتفسير.	context bloat، collapse مبكر، أو صرف compute بلا Information Gain.	Useful-context ratio، router regret، remaining-gap reduction، fixed-model ablation.

QMCD: حفظ التراكب حتى ينهار بالدليل

Candidate superposition and discriminating tests

عندما لا يكفي evidence، لا يحسم النظام بين branch A/B بمجرد majority vote. QMCD يحتفظ بـCandidateSet ولكل مرشح prediction وcheap discriminator. في software: هل bug من race أم schema؟ في Linux: هل latency من disk أم network؟ في browser: هل element hidden أم page stale؟



الهدف هو نقل «التراكب» من metaphor إلى data structure. Candidate يبقى live طالما لم توجد تجربة تفصله. MCD يقرر كم candidate يستحق التكلفة. Court يمكن أن يرفض collapse إذا كانت evidence متساوية أو غير مستقلة.

> هذا يخلق engineering equivalent لـplural reasoning: الخلاف غير المحسوم يصبح state لا نصاً ضائعاً في المحادثة.

العقد التنفيذي	نمط الفشل	مقياس/اختبار
ربط QMCD: حفظ التراكب حتى ينهار بالدليل بـContext/Router objects وقرارات قابلة للتفسير.	collapse مبكر، أو صرف compute بلا context bloat، Information Gain.	Useful-context ratio، router regret، remaining-gap reduction، fixed-model ablation.

MCAI وHAKIM-MCD: من المرشح إلى الوكالة

Meta-cognitive agency allocation

MCAI يقرر أين يُرسل الفرق: external harness، agent، tool، HyperSkill، model reasoning أو HAKIM-MCD. human. يغلق المسار: Difference → compute → memory → WorldEffect → agency → retrieval/tools → falsification → candidates → route. في التنفيذ، كل مرحلة لها typed input/output ويمكن Court أن يعيد السلسلة للخلف.

```
Difference
 -> MCD/QMCD
 -> CandidateSet
 -> MCAI allocation
 -> HAKIM pipeline
 -> Action
 -> Readback
 -> Evidence
 -> Memory
```

هذا التصميم يسمح بأن يكون «الذكاء» أحياناً عدم استخدام LLM. إذا exact grep أو LSP reference search أو systemctl status يعطي الإجابة، MCAI يجب أن يختار deterministic action. الذكاء الفتاح هنا يكتشف الطريق الأقل تكلفة الذي يحتفظ بالإثبات.

العقد التنفيذي	نمط الفشل	مقياس/اختبار
ربط MCAI وHAKIM-MCD: من المرشح إلى الوكالة بـContext/Router objects وقرارات قابلة للتفسير.	collapse، context bloat مبكر، أو صرف compute بلا Information Gain.	Useful-context ratio، router regret، remaining-gap reduction، fixed-model ablation.

الحوسبة الوعائية: Router على مستوى النظام كله

Sparse vascular compute and routing

Router ليس اختيار model فقط. هو multiplexor فوق knowledge/context/skill/ability/loop/agent/host/tool/verifier. تمثل الحوسبة الوعائية الفكرة أن الموارد تتدفق إلى مناطق ذات centrality وuncertainty وleverage وrisk، بينما nodes الباردة تبقى addressable. هذا ضروري مع HyperKB كبيرة وعدة لغات وأنظمة تشغيل.

نوع التوجيه	أمثلة
Knowledge	AST symbol vs KB theory vs previous trace
Capability	HyperSkill vs dynamic plan
Execution	Native shell vs Hermes vs Codex
Environment	Windows vs WSL vs remote Linux
Verification	unit test vs browser readback vs system metric

ينتج RouterDecision object يحمل alternatives والسبب والكلفة المتوقعة. لاحقاً يمكن RuntimePerformanceRegistry قياس regret والنجاح المتحقق بدلاً من تحسين heuristics عبر الانطباع.

العقد التنفيذي	نمط الفشل	مقياس/اختبار
ربط الحوسبة الوعائية: Router على مستوى النظام كله بـContext/Router objects وقرارات قابلة للتفسير.	collapse ،context bloat مبكر، أو صرف compute بلا Information Gain.	Useful-context ratio، router regret، remaining-gap reduction، fixed-model ablation.

EVCF/MCAS: سياق يتجاوز نافذة النموذج دون ادعاء اللانهاية

Extensible virtual context and cognitive address space

يظل context window مجرد resident set. EVCF يدير ملفات ومستودعات وجداول وlogs وصوراً وbrowser state وtraces خارج النافذة؛ MCAS يضيف objects إدراكية: candidate plans، world entities، gaps، hypotheses، claims. كل مورد له URI وhash وfreshness وauthority وmaterialization policy.

```
ctx://repo/core/runtime.py
ctx://symbol/RouterDecision
ctx://db/telemetry/provider_runs
cog://gap/G-142
cog://hypothesis/H-9
cog://world/service/auth-api
```

العملية الثورية ليست storage أكبر، بل **compute-to-context + paging**: يرسل model query إلى resource، تنفذ SQL/LSP/ripgrep/Python ثم يُعاد view صغير مع evidence handle. إذا resource كبير من logs، لا نلخصه كله؛ نبحث time slice وerror signatures. إذا repo ضخم، نستدعي symbol graph لا الملفات كاملة.

> معيار EVCF الحقيقي: context faults recovered، useful-context ratio، tokens avoided، وmission success على موارد أكبر من model context.

العقد التنفيذي	نمط الفشل	مقياس/اختبار
ربط EVCF/MCAS: سياق يتجاوز نافذة النموذج دون ادعاء اللانهاية بـContext/Router objects وقرارات قابلة للتفسير.	collapse، context bloat مبكر، أو صرف compute بلا Information Gain.	remaining-gap reduction، router regret، Useful-context ratio، fixed-model ablation.

materialized view کے PEK Compiler: prompt

Prompt-embedded knowledge runtime compilation

PEK يبني كل model call من طبقات: Tool schemas، Active Ability/Skills/Loop، relevant HyperKB، World State، Q/Δ، Mission، Constitution، Output contract، Memory، Evidence. يجب أن يكون fingerprint قابل لإعادة البناء وأن يسجل لماذا دخلت كل وحدة.

طبقة	سياسة
Constitution	Pinned؛ لا eviction
Mission/Acceptance	Pinned
Gaps/Failed tests	Pinned حتى الحل
HyperKnowledge	Sparse retrieval
Tool results	Artifact handles عند الكبر
History	Compaction مع decisions/errors
Computer screenshots	آخر state + handle للتاريخ

من telemetry PEK يمكن إنشاء ContextPolicyCandidate: وحدات knowledge التي لم تؤثر تُخفّض؛ وحدات حاسمة مفقودة تُرفع. لكن السياسة الجديدة تمر held-out eval، لأن context optimization قد تحذف شرطاً نادراً لكنه أمني.

العقد التنفيذي	نمط الفشل	مقياس/اختبار
ربط PEK Compiler: prompt کـ materialized view بـ Context/Router objects وقرارات قابلة للتفسير.	context bloat، collapse مبكر، أو صرف compute بلا Information Gain.	Useful-context ratio، router regret، remaining-gap reduction، fixed-model ablation.

HyperKnowledge Deploy: من corpus إلى خلايا تشغيل

Operationalizing the corpus

HyperKBالمستهدفة لا تُقاس بالMB فقط. كل unit يجب أن تجيب: ما trigger؟ ما mechanism؟ ما scope؟ ما evidence rank؟ ما failure boundary؟ ما skill/ability المرتبطة؟ ما freshness؟ ويمكن لـKnowledgeDistiller ربط duplicates وsupersedes وcontradictions بلا حذف الأصل.

```
HKUnit {
 id, domain, trigger, mechanism, scope,
 evidence_rank, provenance, freshness,
 relations, capability_refs, tests, failure_boundary
}
```

في kernel deploy نستخدم knowledge classes: Descriptive، Constraint، Procedural، Evaluative، Historical، Speculative. speculative unit يمكن أن يولد hypothesis لكنه لا يحصل على authority. Constraint unit قد يدخل PEK تلقائياً إذا scope يطابق mission. Evaluative unit قد يصبح Court rule.

> الهدف النهائي: refined Knowledge → Experience → Effect → Capability → Knowledge. عندها تصبح المعرفة OS لا أرشيفاً.

العقد التنفيذي	نمط الفشل	مقياس/اختبار
تعريف HyperKnowledge Deploy: من corpus إلى خلايا تشغيل كقدرة/حالة ذات pre/postconditions وEvidence/Recovery.	نجاح لفظي، tool return بلا readback، أو تعميم خبرة one-off.	Verified success، false-closure rate، transfer gain، recovery/rollback test.

HyperSkill Runtime: القدرة الذرية القابلة للنقل

Executable atomic capability

HyperSkill هي أقل procedure نريد إعادة استخدامها عبر models/hosts. في software مثالها «تشخيص failing test مع minimal reproduction»؛ في Linux «تحقق من service degradation بلا تعديل»؛ في browser «ملء form مع verification قبل submit». كل Skill تحمل pre/postconditions وpermissions وtests/recovery.

عنصر العقد	مثال
Trigger	pytest failure signature
Input	repo + failing test
Tools	rg, tree-sitter, pytest
Authority	read + reversible local edit
Postcondition	test passes with no regression
Evidence	test output + diff
Recovery	rollback + alternate hypothesis

Invocation تسجل causal ledger. Skill التي تكرر النجاح عبر held-out repos ترتفع quality؛ التي تفشل أو تسبب side effects تُquarantine. هذه telemetry أساس SkillFactory لاحقاً.

العقد التنفيذي	نمط الفشل	مقياس/اختبار
تعريف HyperSkill Runtime: القدرة الذرية القابلة للنقل كقدرة/حالة ذات pre/postconditions وEvidence/Recovery.	نجاح لفظي، tool return بلا readback، أو تعميم خبرة one-off.	Verified success، false-closure rate، transfer gain، recovery/rollback test.

HyperAbility: هندسة قدرة مركبة بدل persona

Composed capabilities

HyperAbility تمثل مساراً مغلقاً من skills والloops والأدوات والمحاكم. مثال **HA_PRODUCTION_FEATURE**: Discovery → Contract → Implement → Test → Security. يمكن route كل node إلى host مختلف لكن ability semantics تبقى ثابتة. Verify → Deploy Dry-Run



القدرة لا تعتبر closed إذا احتوت effect دون verifier أو إذا أمكن تجاوز DynamicHyperAbility authority. يمكن توليدها لمهمة واحدة، لا تصبح canonical إلا بعد ظهور reuse pattern وGeneralization Court. هذا يمنع registry من التضخم بقدرات one-off.

العقد التنفيذي	نمط الفشل	مقياس/اختبار
تعريف HyperAbility: هندسة قدرة مركبة بدل persona كقدرة/حالة ذات pre/postconditions وEvidence/Recovery.	نجاح لفظي، tool return بلا readback، أو تعميم خبرة one-off.	Verified success، false-closure rate، transfer gain، recovery/rollback test.

HyperLoop: recurrence محدودة بالدليل والميزانية

Bounded recurrent control

HyperLoop هي state machine وليست prompt «استمر حتى تنتهي». Engineering loop: Observe → Difference → Retrieve → Plan → Act → Readback → Verify → Repair/Replan/Close. لكل loop budgets للوقت والtokens والأدوات وعدد المحاولات، watchdog وcheckpoint policy.

```
HL_IMPLEMENT_VERIFY_REPAIR
ENTRY: accepted plan
STOP: acceptance && verified_delta && no_critical_gap
MAX_ITER: 8
ON_FAIL: classify -> repair | replan | escalate
CHECKPOINT: before mutable actions
```

StagnationDetector يراقب تكرار نفس error أو نفس plan أو عدم تغير RemainingGap. عندها لا يزيد loop عدد المحاولات فقط؛ يستدعي Fattah opening search أو QMCD alternative.

العقد التنفيذي	نمط الفشل	مقياس/اختبار
تعريف HyperLoop: recurrence محدودة بالدليل والميزانية كقدرة/حالة ذات pre/postconditions وEvidence/Recovery.	نجاح لفظي، tool return بلا readback، أو تعميم خبرة one-off.	Verified success، false-closure rate، transfer gain، recovery/rollback test.

HyperAgent وAgent Mesh: سكان محدودون لا شخصيات

Bounded agents and adaptive population

HyperAgent هو capability bundle له memory، tools، model policy، authority ceiling، knowledge scope، goal scope، وAgent Mesh evidence contract. لا يعني 30 agent دائمين؛ AgentPopulationController يقدر هل parallelism أو independent verification يبرر coordination cost.

Spawn if ExpectedParallelGain + IndependenceGain > ContextCost + CoordinationCost + ToolContention

دور وظيفي	متى يظهر
RepositoryExplorer	عند غموض topology
ImplementationWorker	عند plan واضح
LinuxSRE	عند service/network diagnostics
BrowserWorker	عند DOM/browser task
IndependentVerifier	عند medium/high risk
FattahExplorer	عند stagnation أو unknown environment

Delegation يعيد structured result + evidence + artifacts + unresolved gaps، ولا يعيد full child transcript افتراضياً. هذا يحمي parent context ويسمح ببقاء agent retained إذا كان يحتاج state طويلة.

العقد التنفيذي	نمط الفشل	مقياس/اختبار
تعريف HyperAgent وAgent Mesh: سكان محدودون لا شخصيات كقدرة/حالة ذات pre/postconditions وEvidence/Recovery.	نجاح لفظي، tool return بلا readback، أو تعميم خبرة one-off.	Verified success، false-closure rate، transfer gain، recovery/rollback test.

Mission DAG وHyperBranch: هندسة التوازي القابل للدمج

Task graph and isolated mutation

Mission DAG يحول goal إلى nodes ذات dependencies/acceptance/budget. في software nodes التي تعدل نفس repo تُعزل بـGit worktrees أو sandboxes. HyperBranchManager يسجل merge candidate، tests، diff، owner، dirty state، base commit وcleanup.

حالة node	المعنى
PENDING	dependencies لم تكتمل
READY	قابل للتشغيل
RUNNING	execution active
WAITING	tool/human/agent
PASSED	postcondition verified
FAILED	failure classified
ROLLED_BACK	effect reversed

التوازي لا يُسمح إذا nodes تتصارع على shared mutable resource من دون transaction semantics. Git isolation مفيد للكود؛ database/environment actions تحتاج locks أو compensation أو sequencing.

العقد التنفيذي	نمط الفشل	مقياس/اختبار
تعريف Mission DAG وHyperBranch: هندسة التوازي القابل للدمج كقدرة/حالة ذات pre/postconditions وEvidence/Recovery.	نجاح لفظي، tool return بلا readback، أو تعميم خبرة one-off.	Verified success، false-closure rate، transfer gain، recovery/rollback test.

WorldEffect Transaction: الإغلاق الحقيقي للفعل

Evidence-carrying action and closure

كل action مهم يحمل EvidencePolicy، Reversibility، ExpectedDelta، Authority، EffectClass، Preconditions، Target، Intent. بعد التنفيذ لا نثق في return code وحده؛ نقرأ state المستهدفة ونقارن observed delta. هكذا يصبح world effect transaction هو atom للنجاح.

```
WorldEffect {
 before, intended_delta, action_trace,
 observed_after, observed_delta, evidence, verification
}
```

$$\text{MissionSuccess} = \text{AcceptanceCriteria} \wedge \text{VerifiedObservedDelta} \wedge \text{NoCriticalGap}$$

في البرمجة: build + tests + diff. في browser: page state بعد submit. في Windows: service/process/registry state. في Linux/: systemd/unit/network filesystem state. إذا لا يمكن readback، النتيجة UNVERIFIED أو PARTIAL لا SUCCESS.

العقد التنفيذي	نمط الفشل	مقياس/اختبار
تعريف WorldEffect Transaction: الإغلاق الحقيقي للفعل كقدرة/حالة ذات pre/postconditions وEvidence/Recovery.	نجاح لفظي، return tool بلا readback، أو تعميم خبرة one-off.	Verified success، false-closure rate، transfer gain، recovery/rollback test.

Self-Reconstruction Memory: الاستمرار بأقل حالة كافية

Durable continuity and minimal writeback

الذاكرة لا تحفظ كل token. بعد mission أو ExperienceCompiler، checkpoint يستخرج unresolved، world deltas، evidence، failed approaches، decisions، gaps وSelf-Reconstruction Memory يجب أن تسمح بإعادة بدء session بعد انقطاع terminal من دون إعادة قراءة كل التاريخ. reusable patterns.

نوع الذاكرة	المحتوى
Working	temporary active objects
Mission	plan/gaps/evidence/checkpoints
Project	verified decisions/world state
Procedural	skills/abilities/loops
Error	signatures + recoveries
Epistemic	claims + evidence/counterevidence

أي تعلم قابل للتعميم يتحول candidate لا حقيقة production مباشرة. هذه قاعدة مهمة لأن agent قد يتعلم shortcut خاص بمشروع واحد أو سلوكاً يرفع benchmark دون القدرة الحقيقية.

العقد التنفيذي	نمط الفشل	مقياس/اختبار
تعريف Self-Reconstruction Memory: الاستمرار بأقل حالة كافية كمقدرة/حالة ذات pre/postconditions وEvidence/Recovery.	نجاح لفظي، tool return بلا readback، أو تعميم خبرة one-off.	Verified success، false-closure rate، transfer gain، recovery/rollback test.

Computer Use: من النقر إلى هندسة حالة النظام

State-grounded computer control

طبقة ComputerUseRuntime يجب أن تفضّل structured state على pixels. ترتيب الفعل: API → app CLI → filesystem/data → DOM → accessibility/UI → vision clicking. Automation. هذا يرفع reliability ويجعل evidence أوضح. screenshot-only يبقى fallback عندما لا توجد واجهة بنيوية.

ComputerObservation	مصدر
DOM/accessibility	browser/app structured tree
Window/process	OS APIs
Screen	screenshot/vision
Clipboard metadata	scoped handle
Action history	runtime trace

ComputerAction يحمل intent وtarget وexpected effect وrisk وverification. بعد click نعيد observe؛ إذا ظهر dialog غير متوقع، تتغير state ويعاد التخطيط. Human takeover يصبح WAITING_FOR_HUMAN مع resume checkpoint لا فشل غامض.

> التحكم الحاسوبي في AL RAZZI ليس امتيازاً للنموذج؛ هو Capability خلف ToolGateway وAuthorityEngine.

العقد التنفيذي	نمط الفشل	اختبار ميداني
وضع Computer Use: من النقر إلى هندسة حالة النظام خلف ToolGateway + Adapter + Authority مع Readback صريح.	تنفيذ مباشر غير محكوم، parsing هش، privilege زائد أو state drift.	Permission denial، timeout، induced drift، restart، post-action state verification.

Browser HyperControl: DOM أولاً، رؤية عند الحاجة

Browser engineering and verification

للمهام الويب ندمج Playwright كطبقة deterministic عالية الثقة، وBrowser Use كطبقة agentic أعلى عند المواقع المتغيرة، مع إمكانية Playwright MCP/CLI. يوفر Chromium/Firefox/WebKit وعزل واختبارات E2E؛ Browser Use يوفر agentic browser control وCLI/MCP، لكن لا يجب أن يُستخدم لطلب HTTP بسيط يمكن حله بـAPI/curl.

المستوى	أداة/أسلوب	الاستخدام
HTTP/API	httpx/curl	بيانات مباشرة بلا UI
DOM/Test	Playwright	selectors, assertions, E2E
Agentic browser	Browser Use	dynamic multistep browsing
Visual fallback	ComputerUse vision	canvas/legacy/opaque UI

BrowserWorldModel يحفظ pending navigation، form state، page entities، auth state handle، URL، وlast verified action. لا تحفظ secrets أو cookies في HyperKB؛ تبقى handles في secret/session store.

> مراجع مفتوحة المصدر: https://github.com/microsoft/playwright و https://github.com/browser-use/browser-use

العقد التنفيذي	نمط الفشل	اختبار ميداني
وضع Browser HyperControl: DOM أولاً، رؤية عند الحاجة خلف Adapter + ToolGateway + Authority مع Readback صريح.	تنفيذ مباشر غير محكوم، parsing هش، privilege زائد أو state drift.	Permission denial، timeout، induced drift، restart، post-action state verification.

Windows Control Plane: واجهة نظام لا مجموعة أوامر

Windows native engineering plane

على Windows نحتاج تجمع abstraction networking، Scheduled Tasks، Registry، Event Logs، Service Control Manager، WMI/CIM، PowerShell 7، files، processes وWindows UI Automation. الهدف أن mission تقول desired state لا سلسلة أوامر ثابتة؛ adapter يختار أفضل interface ويعيد structured observations.

مجال	واجهات محتملة
Services	Get-Service / CIM / sc.exe
Processes	Get-Process / WMI
Registry	PowerShell Registry provider
Events	Get-WinEvent
Network	*Get-NetTCPConnection / netsh / PowerShell Net
Tasks	ScheduledTasks module
UI	UIAutomation/pywinauto
Packages	winget / scoop/choco when policy permits

كل mutation تسجل before-state قدر الإمكان. Registry/service changes تُعامل REVERSIBLE_WRITE إن أمكن install/uninstall. capture old value/state أو firewall changes تحتاج أعلى verification وربما approval حسب scope.

العقد التنفيذي	نمط الفشل	اختبار ميداني
وضع Windows Control Plane: واجهة نظام لا مجموعة أوامر خلف Adapter + ToolGateway Authority مع Readback صريح.	تنفيذ مباشر غير محكوم، parsing هش، privilege زائد أو state drift.	Permission denial، timeout، induced drift، restart، post-action state verification.

PowerShell HyperEngineering: لغة تحكم أولى في Windows وعابرة للمنصات

PowerShell as structured automation substrate

PowerShell 7 ليس مجرد shell؛ هو object pipeline، REST/JSON/XML integration وframework للcmdlets ويعمل على Windows/Linux/macOS. لذلك نعرّف PowerShellCapability كأداة structured، مع transcript مُجزأ، WhatIf/Confirm، strict error handling عندما تدعمها cmdlets، وPester للتحقق من scripts/modules.

قدرة	تطبيق
Discovery	Get-Command/Get-Module/Get-Help
Structured state	ConvertTo/From-Json, CIM
Remote	PowerShell Remoting/SSH where configured
Testing	Pester
Safety	WhatIf + scoped policy-
Artifact	PS script/module + tests + manifest

Agent الذي يولد PowerShell يجب أن يميز بين output stream وerror stream، ويتجنب parsing نص إذا object متاح، ويستخدم `ErrorAction Stop-` في العمليات الحساسة مع try/catch. أي script يُرقى HyperSkill يمر Pester + idempotency/replay tests.

> مرجع: https://github.com/PowerShell/PowerShell ؛ Pester: https://github.com/pester/Pester

العقد التنفيذي	نمط الفشل	اختبار ميداني
وضع PowerShell HyperEngineering: لغة تحكم أولى في Windows وعابرة للمنصات خلف Adapter + Authority + ToolGateway مع Readback صريح.	تنفيذ مباشر غير محكوم، parsing هش، privilege زائد أو state drift.	Permission denial، timeout، induced drift، restart، post-action state verification.

Windows Observability & Recovery

Event-driven diagnostics and rollback

قوة التحكم لا تأتي من القدرة على التنفيذ فقط، بل من التشخيص. WindowsAdapter يجب أن يربط Event Logs، process/service metrics، filesystem changes وapplication logs إلى Evidence Ledger. قبل إصلاح service، نسجل observed state؛ بعد restart نتحقق من running state وhealth endpoint/port/logs. restart ناجح تقنياً لا يكفي إذا application لا تستجيب.

نقترح FailureSignatures لنماذج متكررة: execution، service timeout، locked file، path/encoding، missing DLL/runtime، access denied، port conflict. ErrorMemory يربط signature بـdiagnostic sequence وrecovery proven. بهذه الطريقة تتحول الإدارة إلى procedural intelligence لا حفظ أوامر.

Opening pattern: عندما يتكرر فشل Fattah layer، shell تبحث عن interface أعلى أو أدنى: event source، registry state، service API، أو isolated reinstall - لا تكرر الأمر نفسه.

العقد التنفيذي	نمط الفشل	اختبار ميداني
وضع Windows Observability & Recovery خلف ToolGateway + Adapter + Authority مع Readback صريح.	تنفيذ مباشر غير محكوم، parsing هش، privilege زائد أو state drift.	Permission denial، timeout، induced drift، restart، post-action state verification.

Linux HyperEngineering: system state as a first-class world

Linux control and SRE plane

Linux ليس «bash أكبر». النواة تتعامل مع logs، packages، network، storage، mounts، namespaces، cgroups، permissions، users، processes، services. الواجهات الأساسية تشمل /proc، systemd/journalctl، SSH/ و kernel observability كعالم structured. sys، iproute2، ss، nftables، package managers، container runtimes وeBPF tools.

Domain	Read-first probes	Effectful actions
Service	systemctl status/show, journalctl	restart/enable/config
Network	ip, ss, resolvectl	routes/firewall
Storage	lsblk, df, mount	mount/resize
Process	ps, /proc, cgroups	signal/limits
Security	id, getfacl, audit logs	permissions/policy

قاعدة Linux: probe before mutate. command output يتحول إلى structured state حيث أمكن. `systemctl is-active` + health probe + journal delta أقوى من «service restart returned 0».

العقد التنفيذي	نمط الفشل	اختبار ميداني
وضع Linux HyperEngineering: system state as a first-class world خلف Adapter + ToolGateway + Authority مع Readback صريح.	تنفيذ مباشر غير محكوم، parsing هش، privilege زائد أو state drift.	Permission denial، timeout، induced drift، restart، post-action state verification.

Linux Service/Network/Storage Control Loops

Closed-loop operations

نعرّف HyperLoops تخصصية: ServiceRecoveryLoop، NetworkDiagnosisLoop، DiskPressureLoop. كل loop يبدأ بـobservations محددة، يكوّن hypotheses، ينفذ discriminating probes، ثم mutation smallest reversible. مثال latency: لا يقفز إلى restart؛ يقيس CPU/memory/I/O/network، يحدد bottleneck، ثم يجرب change محدود.

Loop	Observation	Verifier
ServiceRecovery	unit state + logs + ports	health + error-rate
NetworkDiagnosis	routes/DNS/socket/latency	connectivity + packet path
DiskPressure	df/du/inodes/I/O	free-space + app health
PermissionRepair	owner/mode/ACL	minimal access test

هذا يحول SRE knowledge إلى HyperSkills وليس prompts. الـautomation يمكن replay على sandbox/VM، وتصبح تغييرات production T3 حسب MUG policy.

العقد التنفيذي	نمط الفشل	اختبار ميداني
وضع Linux Service/Network/Storage Control Loops خلف ToolGateway + Adapter + Authority مع Readback صريح.	تنفيذ مباشر غير محكوم، parsing هش، privilege زائد أو state drift.	Permission denial، timeout، induced drift، restart، post-action state verification.

Linux Deep Observability: eBPF كطبقة فتح تشخيصي

eBPF, BCC and bpftrace

عندما لا تكفي metrics/logs، يمكن أن يكون eBPF opening عالي القيمة لفهم latency وsyscalls وI/O وnetwork بلا تعديل application. BCC يوفر أدوات ومكتبات BPF، وbpftrace لغة tracing عالية المستوى. هذه أدوات قوية لكنها privileged؛ لذلك تُفعّل فقط في LinuxDiagnosticAbility مع authority وسياسة واضحة.

مستوى	أمثلة	الهدف
Classic	strace, perf, lsof	syscalls/perf/open files
eBPF tools	BCC	*biolatency, execsnoop, tcp
Programmable	bpftrace	custom probes
Kernel state	proc,/sys/	ground truth سريع

المعيار: لا نستخدم eBPF لمجرد التعقيد. MCD يرفع tracing depth عندما information gain المتوقع يبرر privilege/cost. نتائج tracing تصبح artifacts/aggregates لا dumps ضخمة داخل model context.

> مراجع: https://github.com/iovisor/bcc و https://github.com/bpftrace/bpftrace

العقد التنفيذي	نمط الفشل	اختبار ميداني
وضع Linux Deep Observability: eBPF كطبقة فتح تشخيصي خلف Adapter + ToolGateway + Authority مع Readback صريح.	تنفيذ مباشر غير محكوم، parsing هش، privilege زائد أو state drift.	Permission denial، timeout، induced drift، restart، post-action state verification.

Reproducible Linux Environments: Nix, containers and sandboxing

Reproducibility as capability

الهندسة الفائقة تحتاج القدرة على إعادة بناء environment. Nix يوفر package management reproducible على Unix؛ containers/Podman/Docker تعزل dependencies؛ OpenShell يضيف policy-enforced agent sandboxing مع filesystem/network/process/inference controls. النواة لا تختار واحداً دائماً؛ EnvironmentResolver يقرر حسب host والخطر.

أداة/طبقة	الدور في AL RAZZI
Nix	reproducible dev/tool environment
Docker/Podman	sandbox/runtime packaging
NVIDIA OpenShell	agent-first secure runtime policies
WSL	Linux compatibility on Windows
venv/uv	Python environment isolation

نستخدم upstream كطبقات optional لا كسيادة. OpenShell مثلاً مناسب لتشغيل Claude/OpenCode/Codex/Hermes داخل sandbox policies، بينما mission/evidence تبقى Moussaif-owned.

> مراجع: https://github.com/NixOS/nix و https://github.com/NVIDIA/OpenShell

العقد التنفيذي	نمط الفشل	اختبار ميداني
وضع Reproducible Linux Environments: Nix, containers and sandboxing خلف Adapter + ToolGateway + Authority مع Readback صريح.	تنفيذ مباشر غير محكوم، parsing هش، privilege زائد أو state drift.	restart، induced drift، timeout، Permission denial، post-action state verification.

Remote Linux/Fleet Engineering

SSH, bastions, inventories and bounded autonomy

يجب أن يفصل MOUSSAIFXT عن LocalHost. RemoteLinuxAdapter يكتشف available tools، host fingerprint، connection profile، وRemoteHost. لا تُنسخ authority scope إلى credentials، prompt؛ يمررها CredentialBroker إلى SSH/session layer. Fleet operations تحتاج inventory وconcurrency وbounds وblast-radius policy.

يوقف canary: inspect → dry-run → 1 host → verify → small batch → verify → expand. failure يستخدم rollback/repair ويطلق propagation. هذا نفس HarnessGenome canary principle مطبق على infrastructure.

$$\text{FleetRisk} \approx \text{ChangeRisk} \times \text{HostCount} \times \text{DependencyCentrality} \times \text{Irreversibility}$$

يمكن دمج Ansible أو native SSH scripts كـoptional providers إذا كانت موجودة، لكن يجب أن تُغلف داخل ToolContract بحيث يعرف النظام inventory، result parsing، check mode، idempotency، وevidence.

العقد التنفيذي	نمط الفشل	اختبار ميداني
وضع Remote Linux/Fleet Engineering خلف ToolGateway + Adapter + Authority مع Readback صريح.	تنفيذ مباشر غير محكوم، parsing هش، privilege زائد أو state drift.	Permission denial، timeout، induced drift، restart، post-action state verification.

Polyglot SWE-as-OS: الهندسة البرمجية ليست مرتبطة بلغة

Language-independent software engineering substrate

مبدأ SWE-as-OS: نواة مشتركة تفهم security، Git، build/test contracts، dependencies، symbols، repo topology وdeployment؛ ثم توفر LanguageAdapters specifics. بذلك لا نحتاج prompt منفصل لكل لغة بل Profile، Debug، Package، Format، Lint، TypeCheck، Test، Build، Parse. capabilities.

```
LanguageAdapter {
 detect(), symbols(), dependencies(), build(), test(),
 typecheck(), lint(), format(), debug(), package(), docs()
}
```

Tree-sitter يعطي parsing incremental متعدد اللغات، وLSP يعطي symbols/references/diagnostics، وDAP يمكنه debugging عند توفره. هذه deterministic substrate تقلل اعتماد فهم الكود على model. الهدف «كل اللغات» يصبح معمارياً: أي لغة لها compiler/build/test يمكن إضافتها adapter دون تغيير Mission Core.

> مرجع Tree-sitter: https://github.com/tree-sitter/tree-sitter

العقد التنفيذي	نمط الفشل	اختبار SWE
اكتشاف toolchain الأصلي ثم تفعيل Polyglot SWE-as-OS: الهندسة البرمجية ليست مرتبطة بلغة كـLanguage/Engineering capability قابلة للقياس.	فرض stack خاطئ، تعديل unrelated state، أو اعتبار build وحده إغلاقاً.	Fresh-repo task + dirty-workspace no-loss + tests + diff integration/world-effect verification.

Code Intelligence Fabric: AST + LSP + Search + Git

Deterministic intelligence before LLM

ترتيب الاستكشاف المقترح: project metadata → symbol graph → exact references → lexical search → dependency graph → semantic retrieval → LLM. `ripgrep` للبحث النصي السريع، Tree-sitter لـAST، LSP لـreferences/types/diagnostics، Git للتاريخ والـdiff. EVCF يـmount النتائج كـobjects بدلاً من نسخ الملفات كلها.

السؤال	أداة قبل LLM
أين عُرّفت الدالة؟	LSP/Tree-sitter
أين تُستخدم؟	LSP references/rg
ماذا تغير؟	git diff/log/blame
ما الاختبار المتأثر؟	dependency/test graph
ما النص المتكرر؟	rg/structural search

هذه الطبقة هي أحد أهم مسارات Fattah: اختيار representation يجعل المشكلة أصغر. Agent يقرأ 200 سطر relevant بدلاً من 20k token عشوائي، فيزداد useful-context ratio.

العقد التنفيذي	نمط الفشل	اختبار SWE
اكتشاف toolchain الأصلي ثم تفعيل Code Intelligence Fabric: AST + LSP + Search + Git كـLanguage/Engineering capability قابلة للقياس.	فرض stack خاطئ، تعديل unrelated state، أو اعتبار build وحده إغلاقاً.	Fresh-repo task + dirty-workspace no-loss + tests + diff integration/world-effect verification.

Python HyperEngineering Plane

Production Python engineering

PythonAdapter يدمج `uv` لإدارة environments/dependencies، `Ruff` lint/format، `Pyright` type checking، `pytest` testing، coverage، packaging via pyproject، وAL RAZZI. optional profiling يكتشف الموجود ولا يفرض stack موحدة على repo قائم. الهدف هو تحويل الجودة إلى gates قابلة للقراءة الآلية.

مرحلة	أداة/مبدأ
Environment	uv/venv + lockfile
Style/Lint	Ruff
Types	Pyright/mypy حسب المشروع
Tests	pytest
Security	Semgrep/bandit-like adapters عند الحاجة
Profile	cProfile/py-spy/perf حسب البيئة

Ruff يدعم lint/format ومخرجات machine-readable؛ Pyright مصمم لقواعد كبيرة عالية الأداء. uv يعمل على Windows/Linux ويعطي environments/locks بسرعة. هذه الأدوات لا تصبح hard dependency لل core؛ PythonToolchainAdapter يديرها ويظهر health.

> مراجع: https://github.com/microsoft/pyright · https://github.com/astral-sh/ruff · https://github.com/astral-sh/uv

العقد التنفيذي	نمط الفشل	اختبار SWE
اكتشاف toolchain الأصلي ثم تفعيل Python HyperEngineering Plane كـLanguage/Engineering capability قابلة للقياس.	فرض stack خاطئ، تعديل unrelated state، أو اعتبار build وحده إغلاقاً.	diff + tests، dirty-workspace no-loss، Fresh-repo task + integration/world-effect verification.

TypeScript/JavaScript Engineering Plane

JS/TS repositories and browser-native systems

TS/JS Adapter يكتشف npm/pnpm/yarn/bun، tsconfig، ESLint/formatter، test runner (Vitest/Jest/Playwright)، build system (Vite/Next/Turbo وغيرها) من المشروع. لا يفترض framework. LSP/tsserver يوفر symbols/types؛ Playwright يصبح verifier ممتازاً للـUI.

HyperAbility للـfrontend يجب أن تغلق ما وراء browser console، network errors، visual smoke، accessibility، DOM state. العالم النهائي هو browser/app لا TypeScript compiler فقط. لذلك WorldEffect لواجهة المستخدم قد يجمع build + E2E + screenshot/DOM assertions.

> **قاعدة:** إذا كان failure في browser، لا تعتبر passing unit tests إغلاقاً كاملاً. verifier يطابق level الفعل.

العقد التنفيذي	نمط الفشل	اختبار SWE
اكتشاف toolchain الأصلي ثم تفعيل TypeScript/JavaScript Engineering Plane كـLanguage/Engineering capability قابلة للقياس.	فرض stack خاطئ، تعديل unrelated state، أو اعتبار build وحده إغلاقاً.	diff + tests، dirty-workspace no-loss، Fresh-repo task + integration/world-effect verification.

Rust / C / C++ Engineering Plane

Systems-level software engineering

للRust نستخدم cargo metadata/check/test/clippy/fmt، ولـC/C++ compilers/build systems مثل clang/gcc/cmake/ninja مع static analyzers حسب المشروع. Kernel/SRE tasks تحتاج valgrind-like tooling، sanitizers، أو perf، وLanguageAdapter يقرأ symbol/debug info. عندما تتوفر compilation database (`compile_commands.json`) لتحسين references.

عائلة	التحقق الأدنى
Rust	cargo check + test + clippy
C/C++	build + unit tests + compiler warnings
Systems perf	benchmark + perf/trace
Memory safety	sanitizers/valgrind equivalents

الفكرة ليست أن AL RAZZI يصبح compiler؛ بل أن planning يرتبط بأدلة toolchain native. عند bug منخفض المستوى، Fattah layer قد تفتح طريقاً عبر sanitizer أو trace بدلاً من speculation.

العقد التنفيذي	نمط الفشل	اختبار SWE
اكتشاف toolchain الأصلي ثم تفعيل Rust / C / C++ Engineering Plane كـLanguage/Engineering capability قابلة للقياس.	فرض stack خاطئ، تعديل unrelated state، أو اعتبار build وحده إغلاقاً.	tests + diff + Fresh-repo task، dirty-workspace no-loss، integration/world-effect verification.

Go / JVM / .NET Engineering Plane

Enterprise polyglot adapters

GoAdapter يستخدم go list/test/vet/format/profile؛ JVM Adapter يكتشف static analysis، JUnit، Maven/Gradle، والـJDK؛ DotNetAdapter يستخدم `dotnet` Roslyn/LSP، restore/build/test/format. الهدف unify semantics: build/test/type/lint/package، مع respect لـconventions الحالية.

Stack	Discovery	Verification
Go	go.mod/workspace	go test/vet
Java/Kotlin	pom/gradle	test + compile + static analysis
#NET/C.	sln/csproj	dotnet test/build
Mixed monorepo	workspace graph	per-package + integration gates

Mission DAG يمكن تشغيل nodes لكل package بالتوازي إذا dependency graph يسمح. نتائج toolchains تدخل EvidenceGraph بصورة معيارية بحيث Court لا يحتاج معرفة كل format الخام.

العقد التنفيذي	نمط الفشل	اختبار SWE
اكتشاف toolchain الأصلي ثم تفعيل Go / JVM / .NET Engineering Plane كـLanguage/Engineering capability قابلة للقياس.	فرض stack خاطئ، تعديل unrelated state، أو اعتبار build وحده إغلاقاً.	Fresh-repo task + diff، dirty-workspace no-loss، tests + integration/world-effect verification.

Shell Engineering: Bash + PowerShell كعقد لا كسلاسل نصية

Shell reliability and portability

Shell هو universal escape hatch لكنه أخطر من APIs structured. لذلك ShellToolContract يفرض timeout، environment allowlist، working directory، shell dialect، effect class، output cap، encoding. Bash scripts تمر `shellcheck` وtests مثل bats عند جدوى؛ PowerShell تمر Pester. generated command عالي الخطورة يُعرض dry-run أو يحتاج approval.

Cross-platform planning يجب أن يعرف semantics: quoting/path/permissions تختلف. لا نكتب Bash داخل Windows PowerShell أو نفترض `/tmp`. HostAdapter يقدم shell profile. هذا يمنع فئة كبيرة من failures في agentic coding.

> فتح هندسي مهم: كثير من الأخطاء ليست reasoning errors بل shell/environment mismatch؛ Context Genome يجب أن يثبت host facts قبل code generation.

العقد التنفيذي	نمط الفشل	اختبار SWE
اكتشاف toolchain الأصلي ثم تفعيل Shell Engineering: Bash + PowerShell كعقد لا كسلاسل نصية كـLanguage/Engineering capability قابلة للقياس.	فرض stack خاطئ، تعديل unrelated state، أو اعتبار build وحده إغلاقاً.	diff + tests، dirty-workspace no-loss، Fresh-repo task + integration/world-effect verification.

Repository Intelligence: من شجرة ملفات إلى نموذج سببي للكود

Code world model

CodeWorldModel: modules، symbols، imports، APIs، tests، build graph، owners، recent changes، runtime services التقليدي يُغنى إلى RepoMap وGraph edges. مثل deployment artifacts. calls/depends_on/tests/verifies/deployed_as. كل edit يولد WorldModelDelta بعد verification.

يمكن استخدام Tree-sitter/LSP/Git لإنشاء graph deterministic، ثم model يستنتج higher-level architecture. الفرق بين «read repository» و«learn repository» هو الاحتفاظ بعلاقات موثقة مع freshness. عند commit جديد، invalidate affected nodes فقط.

Context value ≈ Relevant graph neighborhood / tokens materialized

العقد التنفيذي	نمط الفشل	اختبار SWE
اكتشاف toolchain الأصلي ثم تفعيل Repository Intelligence: من شجرة ملفات إلى نموذج سببي للكود كـLanguage/Engineering capability قابلة للقياس.	فرض stack خاطئ، تعديل unrelated state، أو اعتبار build وحده إغلاقاً.	dirty-workspace no-loss، tests + diff + Fresh-repo task integration/world-effect verification.

Testing & Verification Fabric: test ليس مرحلة أخيرة

Multi-layer verification as runtime

VerificationAdapter يطبع test results من pytest/Pester/JUnit/Cargo/Go/dotnet/Playwright وغيرها إلى schema واحد: duration، failed، passed، suite، artifacts، flaky suspicion. Courts تستخدمها مع static analysis وWorldEffect readback.

طبقة	أمثلة
Unit	function/module
Integration	service/db/tool boundary
E2E	browser/API/system
Property/Fuzz	state machines/parsers
Regression	known failures
Security	static/dynamic policy
Performance	benchmark/latency/resource

AL RAZZI يجب أن يرفض الخلط بين «test command ran» و«test passed». كذلك flaky pass لا يساوي reliability؛ نستخدم repeated runs أو pass^k عند الحاجة.

العقد التنفيذي	نمط الفشل	اختبار SWE
اكتشاف toolchain الأصلي ثم تفعيل Testing & Verification Fabric: test ليس مرحلة أخيرة كـLanguage/Engineering capability قابلة للقياس.	فرض stack خاطئ، تعديل unrelated state، أو اعتبار build وحده إغلاقا.	+ Fresh-repo task، dirty-workspace no-loss، tests + diff integration/world-effect verification.

Security & Static Analysis as Courts

Engineering assurance without blocking all progress

SecurityCourt يمكن أن يجمع Semgrep أو secret scanning، dependency audit، language-native analyzers، وpolicy checks حسب المشروع. الهدف ليس تشغيل كل scanners دائماً؛ MCD/risk يحدد العمق. التغييرات التي تمس auth/crypto/network/permissions ترفع depth تلقائياً.

Semgrep مثال لأداة open-source pattern/static analysis يمكن تغليفها كـSecurityToolProvider. النتائج تدخل Evidence Ledger مع severity/path/rule وليس screenshot. fix suggestions تبقى candidates حتى tests/architecture review.

> مبدأ MUG: capability improvement لا يمنح authority أعلى. Skill يستطيع اكتشاف vulnerability لكنه لا يملك حق exploitation أو production mutation تلقائياً.

العقد التنفيذي	نمط الفشل	اختبار SWE
اكتشاف toolchain الأصلي ثم تفعيل Security & Static Analysis as Courts كـLanguage/Engineering capability قابلة للقياس.	فرض stack خاطئ، تعديل unrelated state، أو اعتبار build وحده إغلاقاً.	tests + diff، dirty-workspace no-loss، + Fresh-repo task integration/world-effect verification.

Git, CI/CD & HyperDeployment

Transactional software evolution

يقرأ acceptance. CI/CD adapter فقط بعد merge ؛edit: diff/tests/courts بعد ؛edit: status/base/checkpoint قبل .software effects للـ transaction log يصبح Git pipelines بدل أن يخترع deploy command. HyperDeployment يمثل staging/canary → integration → local tests → promotion lifecycle: candidate production حسب authority.

Candidate → Local Verify → CI → Staging → Canary → Production

«Rollback remains first-class: code via commit/revert، deployment via previous artifact، config via snapshot. «deploy succeeded service health يحتاج + metrics/readback. هذه هي ترجمة WorldEffect إلى DevOps.

العقد التنفيذي	نمط الفشل	اختبار SWE
اكتشاف toolchain الأصلي ثم تفعيل Git, CI/CD & HyperDeployment لـLanguage/ Engineering capability قابلة للقياس.	فرض stack خاطئ، تعديل unrelated state، أو اعتبار build وحده إغلاقاً.	+ Fresh-repo task، dirty-workspace no-loss، tests + diff integration/world-effect verification.

Open-Source Harness Federation: ما نأخذه وما لا نأخذه

Prime, Hermes, OpenCode, OpenHands, mini-SWE

المشروع	قوته	دورنا المقترح
Prime Agent	persistent programmable kernel, recursive subagents, continual harness	RLM/continual patterns + optional adapter
Hermes Agent	memory, skills, many tools, Windows/Linux, MCP, cron	first-class HostAdapter + capability source
OpenCode	clean coding agent, plan/build, Windows desktop	coding harness adapter / backend patterns
OpenHands	SDK/tools/workspace/server separation, composability	sandbox/SDK architecture reference
mini-SWE-agent	minimal high-performing baseline	ablation baseline / simple executor

لا نبني Frankenstein. لكل upstream adoption mode: REFERENCE_ONLY أو PATTERN أو ADAPTER أو OPTIONAL_PLUGIN أو DEPENDENCY.
default هو REFERENCE_ONLY حتى يثبت benchmark أن التكامل يستحق lock-in. تُحفظ commit SHA/license في upstreams.lock.json.

المراجع: https://github.com/anomalyco/opencode · https://github.com/NousResearch/hermes-agent · https://github.com/PrimeIntellect-ai/prime-agent · https://github.com/SWE-agent/mini-swe-agent · https://github.com/OpenHands/OpenHands

سياسة الدمج	الخطر	بوابة الاعتماد
ربط Open-Source Harness Federation: ما نأخذه وما لا نأخذه عبر PATTERN/ADAPTER/PLUGIN لا عبر نسخ framework كاملة.	version/license drift، lock-in، تضارب ownership أو ازدواج state.	Pin SHA/license، capability probe، benchmark، unplug test قبل promotion.

Open-Source Computer & Browser Toolchain

Browser/computer control stack

الأداة	قوتها	تكامل Moussaif
Playwright	deterministic cross-browser automation/testing	BrowserDeterministicAdapter
Browser Use	agentic browser control, CLI/MCP, dynamic tasks	BrowserAgentAdapter
pywinauto	Windows GUI automation via native controls	WindowsUIAdapter
Hermes computer-use-linux	AT-SPI/Wayland/X11 desktop control	LinuxComputerAdapter candidate
MCP ecosystem	tool/data interoperability	MCPAdapter behind ToolGateway

الترتيب يبقى state-first. لا نستخدم computer-use vision عندما Playwright/Accessibility/API يعطي target أدق. وفي المواقع/التطبيقات opaque ننتقل إلى visual action مع readback. كل adapter ينشر capabilities بدل أن يغير core.

مراجع: https://github.com/microsoft/playwright · https://github.com/browser-use/browser-use · https://github.com/pywinauto/pywinauto

سياسة الدمج	الخطر	بوابة الاعتماد
ربط Open-Source Computer & Browser Toolchain عبر PATTERN/ADAPTER/PLUGIN لا عبر نسخ framework كاملاً.	version/license drift، lock-in، تضارب ownership أو ازدواج state.	Pin SHA/license، capability probe، benchmark، unplug fallback test قبل promotion.

الدمج النهائي: Fattah HyperEngineering Kernel واختبار الإنتاج

Final synthesis, deployment gates and research closure

يتلخص الجزء كله في نواة واحدة: **Fattah HyperEngineering Kernel** لا تعني ذكاءً سحرياً، تعني أن system يمتلك state أوسع من model، يكتشف الفرق والسؤال الحاكم، يختار representation وأداة/لغة/host بأعلى leverage، ينفذ داخل حدود موثقة، يقرأ العالم بعد التنفيذ، ثم يحول التجربة إلى candidate معرفة/قدرة قابلة للاختبار. هذه هي نقطة اتحاد النظريات التطبيقية مع software/computer/Linux engineering.

```
GOAL
 -> Q / Active Difference
 -> MCD/QMCD/Fattah Opening Search
 -> EVCF/MCAS + PEK
 -> HyperAbility/Skills/Loop
 -> Mission DAG
 -> MOUSSAIFXT Router
 -> Language/Browser/Computer/Linux Tooling
 -> Secure Effect
 -> Readback
 -> Evidence/Courts
 -> WorldEffect Closure
 -> ExperienceCompiler
 -> Shadow Evolution Candidate
```

بوابات Production Candidate

Gate	الشرط
P1	كل prior golden tests تمر
P2	real repo feature with verified tests
P3	browser task + Windows/Linux bounded control
P4	restart/resume without state loss
P5	provider/host failover without false success
P6	EVCF handles context > model window
P7	security/authority bypass tests fail closed
P8	self-evolution candidate improves held-out task before promotion

برنامج القياس النهائي

نقيس Verified Completion، False Success، Actions-per-Verified-Delta، Context Efficiency، Tool Reliability، Recovery Rate، Cross-Language Transfer، Linux/Windows Control Success، Browser State Closure، Skill Transfer Gain وOpening Gain. إذا لم تظهر طبقة ما قيمة مقارنة بالbaseline، تُبسط أو تُحذف. بهذا يتحول «الإغلاق النظري» إلى **إغلاق هندسي قابل للنشر والهدم.**

> **الخلاصة:** الهدف ليس بناء أكثر agent ازدحاماً، بل نظاماً يعرف متى يفكر، متى يبحث، متى يبرمج، متى يتحكم في النظام، متى يستدعي إنساناً، ومتى يرفض إعلان النجاح. حين تصبح هذه السلسلة قابلة للقياس والتعميم، نكون قد اقتربنا من معنى عملي لذكاء وكيل فتاح - لا عبر ادعاء AGI، بل عبر هندسة متراكمة قابلة للإثبات.

مراجع تشغيلية مختارة

مصادر المشروع: MOUSSAIFXT Deep Engineering Architecture / MOUSSAIF AGI OS، MOUSSAIF XT Agentic SWE-as-OS، Conscious AGI Master Reference؛ أوراق AL RAZZI Paper I-IV. مراجع Browser، Playwright، mini-SWE-agent، OpenHands، OpenCode، Hermes Agent، Prime Agent، OSS: NVIDIA OpenShell، Tree-sitter، PowerShell، Ruff، Pyright، Nix، BCC/bpftrace Use. تُثبت الإصدارات الفعلية في upstreams.lock.json عند الدمج، وتُراجع الرخص قبل نسخ أي source code.

سياسة الدمج	الخطر	بوابة الاعتماد
ربط الدمج النهائي: Fattah HyperEngineering Kernel واختبار الإنتاج عبر PATTERN/ADAPTER/PLUGIN لا عبر نسخ framework كاملاً.	lock-in، version/license drift، تضارب ownership أو ازدواج state.	Pin SHA/license، capability probe، benchmark، unplug/promotion قبل fallback test.