هندسة البرمجيات الوكيلية كـ OS

لـ MOUSSAIF XT، MOUSSAIF AGI

Agentic SWE-as-OS for Governed Polyglot Automation

خمس نظريات جديدة · ثماني عشرة أداة نظرية · **HyperKB/HyperAbility/HyperSkill** · خطة تنفيذ ودراسة

تنبيه إبستيمي — هذه الورقة توسعة بحثية وهندسية موجهة نحو AGI، ولا تدعي أن MOUSSAIF XT أو أي نموذج حالي قد أثبت امتلاك ذكاء عام اصطناعي. الحجم وعدد المهارات ليسا دليلاً؛ الدليل هو الأثر المتحقق القابل للتكرار والهدم.

إعداد وتركيب: زهير مساعف / Zouhair Moussaif

Arabic RTL · Amiri · Research & Engineering Paper · 18 أغسطس 2026

02. العقد المنهجي ومراتب الادعاء

Epistemic Contract & Claim Discipline

تنطلق هذه الورقة من عقد مساعف المعياري: الحالة المرصودة، والتفويض الصريح، والدليل، والتحقق تتقدم على ثقة النموذج أو حجم المخرجات. لذلك لا يساوي توليد الشيفرة تنفيذها، ولا يساوي استدعاء الأداة تحقق الأثر، ولا يساوي نجاح اختبار واحد صلاحية نظام كامل. هذه التفرقة تعيد تعريف جودة البرمجة الوكيلية باعتبارها سلسلة إثبات من الهدف إلى الأثر الخارجي ثم إلى ذاكرة قابلة لإعادة البناء. [M1]

تُصنف المادة هنا إلى ثلاث طبقات: (أ) عناصر مستخرجة من corpus MOUSSAIF مثل HyperSkill وHyperAbility وHyperLoop و Tool Contracts وPlan DAG وExecutor وWorld-Effect Loop وRuntime Fingerprinting؛ (ب) ممارسات هندسية خارجية حديثة مثل agent tracing، durable execution، attestations، reusable workflows، MCP، hooks، subagents، loops وSWE-bench؛ (ج) خمس نظريات جديدة مقترحة في هذه الورقة يجب اختبارها ولا تُرفع إلى حقائق. [M1–M6, E1–E11]

قاعدة الاستعمال	المعنى	الرتبة
لا قرار تنفيذي حاسم	غير متحقق	E0
يُحفظ كنسب تاريخي	مذكور/هدف	E1
يمكن الاستناد إليه مع حدوده	موجود في artifact	E2
schema/contract/test plan له	محدد هندسياً	E3
توجد آثار تشغيل مباشرة	مجرّب محلياً	E4
يملك baseline وتكراراً مستقلاً	مقارن/مكرر	E5

قاعدة الورقة — وحدة التقدم ليست «نصاً أكثر»، بل تغيّر حالة متحقق، قابل للتتبع، قابل للإرجاع، ومسنود بدليل.

03. الملخص التنفيذي والإسهامات

Executive Abstract & Contributions

المشكلة البحثية هي أن وكلاء البرمجة الحديثة يستطيعون قراءة مستودعات، تعديل ملفات، تشغيل أوامر واختبارات، لكن الانتقال من «وكيل يكتب كوداً» إلى «نظام برمجي موجه نحو AGI» يتطلب طبقة حاكمية موحدة تضبط الهدف والحالة والفرق النشط، وتختار الوعاء البرمجي، وتمنع تجاوز الصلاحيات، وتثبت الأثر، وتتعلم فقط من المسارات المتحققة. يقترح هذا العمل Agentic SWE-as-OS بوصفه امتداداً تنفيذياً لـ MOUSSAIXT.

- تعريف الشيفرة كفعل يغيّر العالم البرمجي، لا كنص لغوي فقط.
- توجيه متعدد اللغات والبيئات عبر مفهوم Polyglot Vascularity مع الحفاظ على mission semantics.
- تحويل الفرق Δ إلى أصغر patch كافٍ مع اختبار ومبرر وتراجع.
- تبلور القدرات من traces المتحققة إلى HyperSkills ثم HyperAbilities وHyperLoops.
- إغلاق برهاني للمهمة يمنع VERIFIED_SUCCESS ما لم تُغلق بوابات الهدف والحالة والتنفيذ والتحقق والأثر والذاكرة.
- مجموعة 18 أداة نظرية مع مخطط تنفيذ تجريبي، وportfolio تقييم متعدد اللغات والمضيفين.

SWE_AGI = G(Q, Δ, K, Route, Plan, Act, Court, Verify, Effect, Memory, Evolution)

المساهمة ليست الادعاء بأن هذه الطبقات تحقق AGI، بل تحويل برنامج MOUSSAIF إلى فرضيات هندسة برمجيات قابلة للقياس: هل تقلل الفشل الصامت؟ هل تحسن النقل بين Python وTypeScript وRust وPowerShell وHCL وغيرها؟ هل تحافظ على القيود عبر تغيّر المضيف؟ هل تؤدي الترقية من trace إلى skill إلى مكسب على holdout؟ هذه أسئلة قابلة للتكذيب.

04. الأساس المعماري من corpus MOUSSAIF

Corpus-Derived Architecture Basis

يعرف تقرير MOUSSAIF AGI OS / MOUSSAIXT HyperSkill كأصغر وحدة معرفة تنفيذية قابلة لإعادة الاستخدام، ويجعل مخرجاتها قابلة للتحقق، ويُفصل الإصدار عن المعرّف. ويعرّف HyperAbility كطبقة تركيب مستقرة بين المهارة والوكيل، فيما HyperLoop متحكم تكراري محدود بعتبات توقف ودليل وتراجع وميزانية أدوات. [M1، الصفحات 21–23]

في نفس السلالة، Tool Contract يحمل schema للمدخل والمخرج، فئة الأثر، التفويض، retry، timeout، reversibility، idempotency، evidence وverification؛ أما Plan DAG فيجعل الاعتماد والتوازي والاعتماد على الدليل صريحاً، ويجعل Executor التحقق مرحلة تنفيذ لا فقرة تقرير أخيرة. [M1، الصفحات 27–29]

الصفحات 37–40 تضيف Runtime Discovery، بصمة البيئة، MOUSSAIXT كغلاف تنفيذ عابر للمضيفين، وObservability/Trace/Audit. والصفحة 49 تحدد الفجوة الهندسية المركزية: الحاجة إلى تنفيذ typed موحد، سجل HyperAbility معياري، benchmarks مجمدة، ومحولات مضيف واختبارات sync. هذا العمل يختار هذه الفجوة تحديداً كنقطة انطلاق.

المفهوم	دوره في SWE-as-OS
HyperKB	معرفة تشغيلية مسترجعة بأقل حزمة كافية
HyperSkill	إجراء ذري قابل للاختبار
HyperAbility	عقد قدرة مركبة
HyperLoop	تكرار محدود لسد فجوة دليل/حالة
Tool Contract	حدود الأثر والتفويض والتحقق
Plan DAG	اعتماد وتوازي وتراجع
World-Effect	إثبات أن الحالة المقصودة تغيرت
MOUSSAIXT	ثبات الدلالة عبر المضيفين

05. حالة الفن 2026: من **autocomplete** إلى **harness** تنفيذي

2026 State of Practice: Agentic Coding Harnesses

تُظهر البنى الحديثة أن نظام البرمجة الوكيلي ليس مجرد استدعاء LLM. يشرح Codex agent loop أن الـ harness ينسق inference والأدوات والسياق والتنفيذ، ويتيح Codex SDK إدماج نفس نمط الوكيل في workflows هندسية. يضيف Claude Code subagents ذات سياق معزول، hooks في نقاط دورة الحياة، وربط الأدوات والبيانات عبر MCP. [E1–E3]

على مستوى التشغيل، يقدّم MCP capability discovery وtools/resources/prompts عبر عقود مخططة؛ يعيد استعمال GitHub Actions workflows ويضيف artifact attestations لبناء provenance؛ يربط OpenTelemetry traces/metrics/logs عبر context propagation؛ يحافظ Temporal على تاريخ workflow لاستعادة التنفيذ بعد الأعطال. [E4–E7]

في التنفيذ الخارجي، يفرض Terraform write-plan-apply، يقدم Playwright auto-waiting والعزل والتتبعات للمتصفح، يميز Kubernetes Jobs المحدودة وCronJobs المجدولة. وفي التقييم، تحول SWE-bench Verified وMultilingual إصلاح قضايا المستودعات إلى اختبار قابل للمقارنة. [E8–E11]

الاستنتاج — أفضل الممارسات المعاصرة تتقارب مع مبادئ MOUSSAIF في الحاجة إلى harness، عقود، عزلة، traces، provenance، durable state، واختبارات على قضايا حقيقية؛ لكن MOUSSAIF يضيف سؤال الحاكم والفرق النشط ومحاكم الهدم وWorld-Effect Closure ضمن طبقة واحدة.

06. الفجوة البحثية: جودة الكود ليست دقة نصية

Research Gap: Coding Quality as Verified State Change

تقييم وكلاء البرمجة غالباً ينتهي عند patch يمر بالاختبارات. هذا معيار مهم لكنه لا يغطي دائماً: هل فُهم النطاق؟ هل تغيّر الشيء الصحيح؟ هل حدث أثر خارجي غير متوقع؟ هل التغيير قابل للإرجاع؟ هل التبعيات موثقة؟ هل نفس المهمة تنتقل بين بيئات مختلفة؟ وهل ما تعلمه الوكيل من نجاح سابق مفيد حقاً أم مجرد حفظ trajectory؟

يقترح SWE-as-OS أن وحدة الجودة هي Verified Causal Delta: فرق سببي موثق بين الحالة قبل وبعد، مرتبط بهدف وتفويض وpatch واختبارات ودليل أثر وتراجع. بهذه العدسة تصبح بعض حالات «النجاح» فشلاً: أمر نجح لكن الخدمة لم تتغير، test suite أخضر لأنه لم يغطِ الحالة الحرجة، deployment تم لكن النسخة غير الصحيحة، agent أعاد نفس التعديل عدة مرات، أو skill تم ترقيتها من trace غير متحققة.

VerifiedDelta = DesiredState - ObservedState | Authorization, Evidence, Tests, WorldEffect, Rollback

- الهدف: خفض silent success.
- الهدف: رفع causal attribution.
- الهدف: تقليل التعديل غير الضروري.
- الهدف: تحسين transfer بين اللغات والمضيفين.
- الهدف: تحويل الفشل إلى أصل بحثي منظم.

07. متطلبات النظام: ما الذي يجب أن يضمنه SWE-as-OS؟

System Requirements

الرمز	المتطلب	اختبار القبول
R1	Intent fidelity	لا يغيّر patch معنى الهدف أو القيود
R2	State grounding	كل قرار حساس له observed-state anchor
R3	Minimal change	الرقعة أصغر من بديل واسع بنفس الإغلاق
R4	Typed effects	فئة الأثر والتفويض صريحة
R5	Verification	دليل مستقل بعد التنفيذ
R6	Rollback	نقطة حفظ ومسار رجوع قبل mutation
R7	Polyglot portability	ثبات contract عبر 3=< vessels
R8	Traceability	إعادة بناء القرار من trace
R9	Regression safety	لا critical regression على holdout
R10	Learning governance	لا self-promotion للمهارات

تُعامل هذه المتطلبات كقيود تصميم لا كشعارات. كل أداة أو طبقة لا تنتج فرقاً قابلًا للقياس يجب أن تُخفض أو تُحذف. وبخاصة، لا يُقبل تعقيد معماري يزيد زمن التنفيذ من دون خفض خطأ أو تحسين نقل أو تقليل تكلفة أو زيادة قابلية الاسترداد.

مبدأ الاقتصاد — الذكاء الهندسي ليس إضافة محكمة وذاكرة ووكيل لكل خطوة، بل تفعيل أقل تركيب يحقق الدليل والاسترجاع. وهذا ينسجم مع Sparse HyperRouting في MOUSSAIF corpus.

08. المعمارية الكلية المقترحة

Proposed SWE-as-OS Macro Architecture

الطبقة	المكوّن	الوظيفة
L0	Constitution / Authorization	السلامة وحدود الأثر
L1	Goal + Governing Question	الهدف والسؤال الذي يعيد تنظيم المهمة
L2	Delta + MCD/QMCD/MCAI	تحديد الفرق وتوجيه الحوسبة
L3	HyperKB / World Model	المعرفة والحالة الخارجية
L4	Polyglot Vessel Router	اختيار لغة/أداة/مضيف
L5	HyperSkills	إجراءات ذرية
L6	HyperAbilities	قدرات مركبة
L7	HyperLoops	حلقات محدودة
L8	Plan DAG + Agents	التخطيط والتفويض
L9	Tool Contracts / Execution	تغيير العالم البرمجي
L10	Courts + Verification	اختبار وهدم وإثبات
L11	Trace / Memory / Evolution	تعلم قابل للإرجاع

الاختلاف عن stack برمجي عادي هو أن طبقة الدليل ليست ملحقاً. كل انتقال بين الطبقات يحمل contract: ما المدخل، ما الأثر المتوقع، ما معيار النجاح، ما الفشل، ما التراجع، وما الذي يمكن كتابته في الذاكرة. MOUSSAIXT يقع على حد portability: لا يفرض لغة واحدة ولا مزوداً واحداً، بل يوحّد semantics.

Mission → Observe → Δ → Route(Vessel,Skill,Tool) → Plan → Checkpoint → Act → Court → Verify → Effect → Memory

09. من الشيفرة كنص إلى الشيفرة كأثر

From Code-as-Text to Code-as-World-Effect

في نموذج الاستجابة التقليدي، تنتج الشيفرة كقطعة نص ثم يُترك التنفيذ للإنسان أو لنظام منفصل. في SWE-as-OS تصبح الشيفرة عقداً لانتقال حالة: schedule، cloud resource، browser، database، service، build، repository، أو registry معرفي. لذلك يجب أن يحمل كل action ``effect_class` و`authorization_tier` و`rollback_ref` و`verification_method`.

هذه النقلة مهمة لـ AGI-oriented coding لأن الوكالة تُقاس بالقدرة على تغيير العالم تحت قيود، لا بالقدرة على وصف التغيير. لكن زيادة القدرة على الفعل ترفع المخاطر، لهذا تُربط agency بالحاكمية: قدرة أعلى = دليل أعلى + تفويض أوضح + مراقبة أثر أعمق + تراجع أقوى.

الحكم	الوصف	حالة
غير منفذ	patch/command مقترح	Generated
غير كافٍ	tool returned success	Executed
دليل قوي	تمت قراءة الحالة بعد التنفيذ	Observed Effect
نجاح متحقق	الاختبارات + الأثر + القيود ناجحة	Verified
قابل للتدقيق	provenance/build evidence محفوظ	Attested
قدرة قابلة لإعادة الاستخدام	holdout/canary نجاح متكرر على	Promoted

10. النظرية الأولى: الأثر البرمجي الحاكمي

T1 - Governed Executable Effect Theory (GEET)

تقترح GEET أن جودة البرمجة داخل نظام موجه نحو AGI ليست دالة في صحة النص وحدها، بل في جودة انتقال الحالة الذي يخضع للهدف والتفويض والتحقق والتراجع والمصدرية. أي أن نفس patch قد يكون ممتازاً في بيئة، ومرفوضاً في أخرى إذا كانت صلاحياته أو أثره أو عدم قابليته للرجوع غير مناسبة.

$$\mathbf{Q_effect} = \mathbf{I} \times \mathbf{P} \times \mathbf{X} \times \mathbf{V} \times \mathbf{R} \times \mathbf{Prov}$$

- I = Intent fidelity.
- P = صحة preconditions.
- X = evidence of execution.
- V = post-condition verification.
- R = reversibility.
- Prov = provenance completeness.

الصيغة multiplicative متعمدة كفرضية تشغيلية: إذا كان أي gate حرج صفراً، لا يجوز تحويل المهمة إلى VERIFIED_SUCCESS. يمكن لاحقاً استبدال الضرب بدالة calibrated إذا أثبتت التجارب أن بعض الأبعاد تعوض بعضاً.

فاصل التكذيب — إذا لم تقلل إضافة بوابات التفويض، فحص الأثر، التراجع والمصدرية معدل الفشل الصامت أو زمن التعافي مقارنة بخط أساس matched، يجب تخفيض النظرية إلى سياسة محلية لا مبدأ عام.

11. تنبؤات GEET وبرنامج اختبارها

GEET Predictions & Tests

التنبؤ	التصميم التجريبي	المقياس
P1	نفس agent مع/بدون effect readback	silent-success rate
P2	نفس patch مع/بدون checkpoint	recovery time
P3	مقابل CI success CI+runtime readback	escaped defect
P4	مقابل none provenance gate	reproducibility
P5	authorization-tier routing	unsafe action rate

تحتاج التجربة إلى فصل أثر النموذج عن أثر harness. لذلك يُجمّد model checkpoint، وtask set، والميزانية، ثم تتغير بوابات GEET فقط. كما يجب حفظ البيئة fingerprint لأن النجاح في repo محلي قد يختلف عند نظام تشغيل أو package manager أو نسخة dependency مختلفة.

النتيجة الإيجابية لا تعني إثبات AGI؛ إنها تثبت فقط أن هندسة الأثر الحاكمي تحسن جانباً من موثوقية الوكالة البرمجية ضمن النطاق المختبر. أما النتيجة السلبية فتفيد في تبسيط النظام ومنع تضخم الحاكمية غير المفيدة.

12. النظرية الثانية: تجميع الفرق إلى رقعة قابلة للإثبات

T2 - Differential Patch Compilation Theory (DPCT)

DPCT تنقل مبدأ «الفرق قبل الحجم» من MCD إلى هندسة الشيفرة. يبدأ النظام بمقارنة الحالة المرصودة مع الحالة المرغوبة، ثم يحدد الفرق الفاعل الذي يغير النتيجة، وبعدها يبحث عن أصغر patch يغلق الفرق ويستطيع إثباته. هذا يعاكس الميل إلى إعادة كتابة وحدات واسعة لمجرد أن النموذج يستطيع ذلك.

$$\textbf{Patch}^* = \textbf{argmin}_p\ [\textbf{ChangeCost}(p)+\textbf{Risk}(p)+\textbf{ContextSpread}(p)]\ \ \textbf{s.t. Closure}(\Delta,p)\geq\theta \wedge \textbf{Verify}(p)=1$$

لا يعني «الأصغر» أقل عدد أسطر دائماً؛ قد يكون migration منظم أكبر لكنه أكثر قابلية للإثبات. المقصود هو minimal sufficient causal intervention: أقل تدخل يحافظ على قابلية الصيانة، الاختبار، التراجع والتفسير.

- استخرج Δ من tests/issues/runtime evidence.
- رتب ملفات/رموز حسب causal relevance.
- امنح patch budget قبل التوليد.
- استخدم counterfactual: ماذا يحدث لو لم نغير هذا الملف؟
- امنع formatting/refactor غير المرتبط إذا لم يرفع الإغلاق.

13. مترجم Δ→Patch ومحكمة الانتشار

Delta-to-Patch Compiler & Scope Court

الأداة AT02 DeltaCode Extractor تنتج كائناً typed يحتوي candidate loci، evidence، observed state، expected state، symptom، وuncertainty. ثم AT07 PlanDAG Compiler يحول الفرق إلى graph من القراءات، التعديلات والاختبارات. قبل التنفيذ، AT09 Patch Court تبحث عن scope drift: ملفات تغيرت بلا dependencies، causal path دخلت بلا حاجة، أو tests تم تعديلها لتوافق patch بدل أن تختبر المتطلب.

حقل Delta	مثال	استخدام
symptom	API يعيد 500	تحديد الأثر
expected	schema صحيح + 200	acceptance
observed	stack trace + failing test	grounding
candidate loci	handler/service/schema	routing
uncertainty	0.27	budget/extra evidence
critical invariants	auth/data integrity	court

الاختبار الفاصل هو مقارنة DPCT بموجه «حل المشكلة» غير المقيد على نفس القضايا. إذا خفض DPCT عدد الملفات/الأسطر المتغيرة مع الحفاظ على النجاح وقلل regressions، فذلك دليل على منفعة. إذا زاد false localization أو فشل في تغييرات cross-cutting، يجب السماح بتوسيع Δ تدريجياً بدلاً من فرض محلية زائفة.

14. النظرية الثالثة: الوعائية البرمجية متعددة اللغات

T3 - Polyglot Vascularity Theory (PVT)

تفترض PVT أن لغات البرمجة، DSLs، shells، وملفات التصريح ليست بدائل شكلية، بل «أوعية تنفيذ» تختلف في نوع الأثر، مستوى typing، latency، portability، idempotency، sandboxability، observability، وقابلية التراجع. لذا لا يختار النظام Python دائماً ولا Shell دائماً، يختار الوعاء الذي يحقق أعلى أثر متحقق بأقل خطر وكلفة.

$$V^* = \text{argmax}_v \left[E_{\text{verified}}(v) + \text{InfoGain}(v) - \lambda_c \text{Cost}(v) - \lambda_r \text{Risk}(v) - \lambda_h \text{HostMismatch}(v) \right]$$

مثلاً: Terraform/HCL أفضل من shell خام عند الحاجة إلى preview وخطة بنية تحتية؛ PowerShell قد يكون وعاءً أصلياً لإدارة Windows؛ TypeScript مناسب لطبقات web/SDK؛ Rust مناسب لأجزاء runtime تتطلب safety؛ YAML يصف workflow لكنه لا يملك semantics كافية وحده دون runner. هذه أمثلة هندسية لا تفاضل مطلق.

قاعدة **portability** — تتغير syntax والtooling، لكن mission contract وacceptance predicates وevidence schema لا تتغير. هذه هي وظيفة MOUSSAIXT boundary.

15. مصفوفة الأوعية البرمجية والأتمتة

Polyglot Automation Vessel Matrix

الوعاء	الاستخدام الرئيسي	التحقق/المخاطر
Python	orchestration/agents/data	tests + type/static + env pinning
TypeScript	SDK/web/node agents	typecheck + unit/e2e
Go	CLIs/operators/services	race/build/tests
Rust	runtime/safety	cargo test/clippy + unsafe audit
PowerShell	Windows/cloud ops	WhatIf/readback + error semantics
Bash	Unix process ops	shellcheck + idempotency guards
HCL/Terraform	IaC	plan + policy + apply readback
Kubernetes YAML	deploy/jobs/cron	server validation + rollout/status
GitHub Actions	CI/CD	reusable workflow + artifact attestation
Nix	reproducible env	lock/input purity checks
SQL	data state	transaction/invariants/rollback
Playwright TS/JS	browser world effect	locators + auto-wait + trace

تحتوي حزمة HyperKB المصاحبة على registry أوسع يشمل أكثر من أربعين لغة/صيغة برمجية وتصريحية. لا تدعي الورقة أنها تشمل كل لغة في التاريخ؛ «all language of automation» يُترجم هنا إلى coverage عملي واسع مع interface موحد يسمح بإضافة vessels دون تغيير النواة.

16. النظرية الرابعة: تبلور القدرة من الأثر الموثق

T4 - Trace-to-Capability Crystallization Theory (TCCT)

تستخرج بنية MOUSSAIF الحالية HyperSkills من traces ناجحة ومتحققة، وتسمح لتركيبات مستقرة أن تصبح HyperAbilities أو HyperLoops. TCCT تجعل هذه الفكرة نظرية ترقية قابلة للقياس: التكرار وحده لا يكفي؛ يجب أن ينجح الإجراء عبر holdout ومضيف أو لغة مختلفة حين يكون ذلك منطقياً، وأن يحافظ على verification وrollback.

$$C = \textbf{RepeatSuccess} \times \textbf{Transfer} \times \textbf{VerificationCoverage} \times \textbf{Reversibility} / (\textbf{Fragility} + \textbf{Regression} + \textbf{ProvenanceGap} + \varepsilon)$$

تُعامل trace كعينة سببية: failure/repair، test، effect، tool، decision، state، goal. بعدها normalizer يستخرج الإجراء ويزيل تفاصيل البيئة غير الضرورية، ثم محكمة skill تبحث عن overfitting، ثم benchmark يقارن بالbaseline. لا يحق للمرشح أن يسجل نفسه كقدرة canonical.

- Trace → candidate procedure.
- Deduplicate by purpose + procedure + effect.
- Holdout benchmark.
- Adversarial court.
- Canary radius.
- Promote / revise / reject / deprecate.

17. من trace إلى HyperSkill ثم HyperAbility، HyperLoop

Capability Crystallization Pipeline

المرحلة	الكائن	شرط الترقية
1	Verified Trace	أثر + اختبار + بيئة + مصدرية
2	Skill Candidate	triggers + failures + إجراء ذري
3	HyperSkill	holdout أفضل من baseline بلا regression حرج
4	Composition Candidate	عدة skills مع contract توافق
5	HyperAbility	نتائج مستقرة على intent class
6	Loop Candidate	تكرار bounded + stop/no-progress
7	HyperLoop	أثر دوري متحقق + rollback + memory policy

AT16 Trace-to-Skill Distiller ينفذ الاستخراج، وAT17 Ability/Loop Crystallizer ينفذ التركيب. يجب قياس «regret of promotion»: كم مرة كان من الأفضل إعادة استخدام الإجراء القديم أو التخطيط من الصفر؟ إذا ارتفع regret، تُخفض القدرة أو تعاد صياغة trigger envelope.

هذا يسمح لمشروع MOUSSAIF XT أن يتعلم تنظيمياً من العمل البرمجي من دون تعديل أوزان النموذج كل مرة. weight updates تبقى مساراً منفصلاً وأعلى تكلفة، ويجب ألا تُستخدم عندما يكفي skill أو retrieval أو adapter.

18. النظرية الخامسة: الإغلاق البرهاني للفعل البرمجي

T5 - Evidential Closure of Software Action (ECSA)

ECSA تعالج أخطر فشل في الوكيل البرمجي: إعلان النجاح قبل إغلاق السلسلة السببية. تقترح ثماني بوابات: Goal، Observed State، Plan، Execution، Verification، World Effect، Attestation، Memory. إذا كانت بوابة حرجة مفقودة، تعاد المهمة PARTIAL أو BLOCKED بدلاً من VERIFIED_SUCCESS.

$$\textbf{Closure} = \mathbf{G} \times \mathbf{O} \times \mathbf{P} \times \mathbf{X} \times \mathbf{V} \times \mathbf{W} \times \mathbf{A} \times \mathbf{M}$$

Gate	السؤال الحاكم
G	هل الهدف ومعيار القبول محددان؟
O	هل نعرف الحالة الحالية من مصدر مرصود؟
P	هل الخطة تشرح الاعتماد والتراجع؟
X	هل التنفيذ موثق لا متخيل؟
V	هل اختبار مستقل نجح؟
W	هل تغيّر العالم المقصود فعلاً؟
A	هل provenance/attestation كافيان؟
M	هل كُتب فقط الفرق المتحقق في الذاكرة؟

19. آلة حالات الإغلاق والنجاح الجزئي

Closure State Machine

تحتاج ECSA إلى state machine صريحة حتى لا يساوي الخطاب النهائي الحكم. الحالة تبدأ INIT ثم DISCOVERED ثم PLANNED ثم CHECKPOINTED ثم EXECUTED ثم VERIFIED_EFFECT، وتنتهي VERIFIED_SUCCESS. أي فشل يمكن أن ينتقل إلى REPAIRING أو ROLLED_BACK أو BLOCKED. تُمنع القفزة EXECUTED→VERIFIED_SUCCESS.

INIT → DISCOVER → PLAN → CHECKPOINT → EXECUTE → VERIFY → EFFECT_READBACK → ATTEST → WRITEBACK

- PARTIAL: جزء من الأثر صحيح لكن gate حرج مفتوح.
- BLOCKED: الدليل/التفويض/القدرة غير متاح.
- ROLLED_BACK: التنفيذ حدث ثم أُرجع بأثر مثبت.
- VERIFIED_SUCCESS: كل البوابات الحرجة مغلقة.

يمكن معايرة gates حسب مستوى الأثر. تعديل README لا يحتاج نفس بوابات migration قاعدة بيانات إنتاجية. لهذا تقترح الورقة Risk Tier من 0 إلى 4، يحدد مقدار المحكمة، نوع checkpoint، وعدد قنوات التحقق المطلوبة.

20. MCD-SWE: توجيه الحوسبة البرمجية بالفرق والقيمة

MCD-SWE Routing Formalization

يُعاد استعمال MCD هنا لا كدرجة غامضة بل كـ routing heuristic. لكل خطوة مرشحة نحسب قيمة الفرق، كسب المعلومات، الاستعجال/الزمن، قابلية تغيير الفعل، قوة الإثبات، قابلية النقل، ثم نطرح الضجيج والكلفة والخطر. الأوزان تُعاير بالتجارب ولا تُقدّم كقانون طبيعي.

$$\mathbf{MCD_SWE = \lambda d\ D + \lambda i\ I + \lambda t\ T + \lambda a\ A + \lambda p\ P + \lambda x\ Transfer - \lambda n\ Noise - \lambda c\ Cost - \lambda r\ Risk}$$

متغير	معناه في SWE
D	مقدار الفرق الذي سيغلقه الإجراء
I	كم معلومة جديدة سيكشفها
T	أثر الزمن/الاعتماد الحرج
A	تأثيره على القرار أو العالم
P	قوة الإثبات المتوقع
Transfer	إعادة الاستخدام عبر مهام/لغات
Noise	تعديل أو سياق زائد
Cost	tokens/CPU/time/tool
Risk	أمن/بيانات/عدم رجوع

21. QMCD/MCAI: مقارنة الفروع قبل الالتزام

QMCD/MCAI for Coding Candidate Competition

عندما توجد عدة حلول معقولة، لا تختار النواة أول patch لغوي. QMCD يحتفظ بعدة فرضيات عن سبب الخطأ، وMCAI يقارن مرشحين من حيث الدليل، الكلفة، الخطر، النقل وقابلية الهدم. يمكن تخصيص وكيل/فرع لقراءة الاختبارات، وآخر للتبعيات، وثالث لمحكمة security، ثم دمج النتائج لا التصويت الساذج.

مرشح	ميزة	خطر	قرار
A: patch محلي	أقل تغيير	قد يخفي سبباً أعمق	اختبر أولاً
B: refactor	يصلح بنية	scope واسع	يتطلب دليل أقوى
C: dependency upgrade	حل upstream	supply-chain/regression	sandbox + attest
D: config change	سريع	host-specific	fingerprint + readback

يُسجل router regret بعد انتهاء المهمة: هل المسار المختار حقق utility المتوقع؟ هذه البيانات تتيح تعديل الأوزان أو triggers من traces بدلاً من تحسين prompt بالحدس فقط.

22. HAKIM-MCD نحط من الإدراك إلى أثر المستودع

HAKIM-MCD SWE Pipeline

يُطبق HAKIM-MCD في SWE نحط مضبوط: تحرير الهدف، استخراج الفرق، استدعاء المعرفة، بناء بدائل، ترتيبها، تنفيذ محدود، محكمة، تحقق، أثر، ثم ذاكرة. وظيفة طبقة HAKIM ليست إنتاج إجابة «حكيمة» لفظياً بل منع القفز من inference إلى mutation من دون جسر إثبات.

Question → Difference → Evidence → Candidate Paths → Court → Authorized Action → Verified Effect → Memory Delta

يمكن إدخال FSKE/محرك الكلام الحاكمي في بعض المواقف كـ meta-reasoning pre-formal: تحرير محل النزاع بين specification وimplementation، أو بين security وfeature pressure. لكن هذا الاستعمال يبقى أداة تنظيم للحكم وليس بديلاً عن tests أو formal proof أو empirical evidence. [M6]

قاعدة فصل الرتب — دليل type checker لا يثبت سلوك runtime؛ passing unit test لا يثبت deployment؛ نجاح deployment لا يثبت business effect. كل رتبة تحتاج شاهدها.

23. HyperKB للهندسة البرمجية: معرفة تعمل لا مستودع نصوص

HyperKB for Agentic Software Engineering

في AGI-Oriented KB-as-OS corpus يرفض أن تكون المعرفة مجرد وثائق مسترجعة. في HyperKB، SWE-as-OS يحمل knowledge atoms مرتبطة بـ provenance، world-effect class، failure mode، verifier، tool contract، language vessel، input/output، trigger، و related skills. الاسترجاع يجب أن يحمّل أقل حزمة كافية، لا 40MB كاملة في السياق. [M3]

HyperKB shard	المحتوى
HKB01	governed agentic SWE contracts
HKB02	polyglot automation vessels
HKB03	verification/courts/world effect
HKB04	runtime/deployment/observability
HKB05	capability evolution/training

الحزمة المصاحبة تولد خمسة corpora بحجم يقارب 40MiB لكل واحد كـ synthetic operational curriculum. السجلات ذات schema موحد ومهيأة لـ RAG/GraphRAG أو fine-tuning experiments، لكنها لا تمثل خمس مجموعات حقائق علمية مستقلة؛ إنها شبكة حالات تدريبية مشتقة من عقود ومصفوفات تنفيذ.

24. **HyperSkill**: من المعرفة إلى إجراء ذري

HyperSkill Engineering for SWE

في هذه الورقة يحتفظ بالعقد HyperSkill canonical: ID ثابت، version منفصل، triggers، inputs، outputs، preconditions، procedure، tools، validation، failure modes، related skills، provenance، status، when-not-to-run. الفرق الإضافي هو فرض `effect_class` و`evidence_grade` و`rollback_requirement` على مهارات mutation.

حقل	مثال
id	HS037
trigger	failing integration test after dependency update
inputs	repo state + lockfile + test trace
procedure	localize → patch → test → effect readback
tools	git/worktree/package manager/test runner
validation	build + integration + diff court
failure	unsupported version / regression
not_to_run	authorization or state unknown

تحتوي الحزمة على 150 HyperSkill تغطي discovery، classification، localization، retrieval، planning، checkpointing، generation، execution، testing، attack، repair، rollback، verification، tracing، attestation، benchmarking والترقية عبر ستة أنواع من الأجسام البرمجية.

25. HyperAbility: واجهة قدرة مستقرة فوق المهارات

HyperAbility as a Stable Composition Interface

HyperAbility لا يساوي agent. هو capability contract يركب عدة HyperSkills وأدوات وسياسات واختبارات ضمن intent class محدد. مثال: «Database Migration Safety» قد يضم مهارات لاكتشاف schema، أخذ snapshot، توليد migration، dry-run، integration tests، integrity readback وrollback. لا يحتاج شخصية مستقلة أو هدفاً دائماً.

HA = Compose(Skills, Tools, PolicyProfile, ValidationSuite, CostRiskEnvelope, Fallback)

الحزمة المصاحبة تقدم 40 HyperAbility، منها Kubernetes CI/CD Repair، Cross-Language Refactor، Issue-to-Patch Closure، Trace-to-Skill Learning، MCP Integration، Terraform Provisioning Governance، Deployment Governance و MOUSSAIXT Full SWE Mission Closure.

معيار النسخة — يتغير إصدار HyperAbility عندما تتغير semantics التركيب، لا لمجرد تحديث نص skill. هذا يحافظ على compatibility envelope.

26. **HyperLoop**: التكرار المحدود وليس التفكير اللامتناهي

Bounded HyperLoop for Coding Missions

SCAN → CLASSIFY → RETRIEVE → ACT → ATTACK → REPAIR → VERIFY → LOG → REMEMBER

يُستخدم HyperLoop عندما لا تكفي خطوة واحدة لإغلاق فرق الدليل أو الحالة. لكنه يحمل stop_condition، evidence_threshold، rollback detector، no-progress retry_budget، وtool budget. إذا أعاد الوكيل نفس test أو patch من دون تغيير evidence/state/plan، نتوقف الحلقة. [M1]

حالة	المعنى
INIT	لم يبدأ التنفيذ
RUNNING	هناك تقدم
WAITING_EVIDENCE	الحاجة إلى دليل خارجي
REPAIRING	إصلاح فشل محدد
VERIFIED	أغلقت البوابات
PARTIAL	نجاح غير مكتمل
BLOCKED	مانع خارجي/تفويض
ROLLED_BACK	رجوع متحقق

من منظور SWE، تكرار agent ليس قيمة بذاته. المقياس هو Evidence Gain per Iteration. يمكن قطع الحلقة عندما يصبح marginal gain أقل من cost/risk أو عندما تُستنفد budget.

27. Tool Contracts: تحويل الأدوات إلى قدرات نمطية

Typed Tool Contracts & Adapters

الأداة في MOUSSAIXT ليست «اسم function». العقد يحدد operation، input/output schemas، authorization tier، side-effect class، idempotency، reversibility، timeout/retry، sensitive fields، evidence returned، وverification method. هذا يسمح للمهمة نفسها أن تنتقل من connector إلى CLI أو API أو MCP server من دون تبديل semantics. [M1]

الحقل	سبب وجوده
side_effect_class	اختيار gate التفويض
idempotency	منع retry خطر
reversibility	تحديد checkpoint/inverse
evidence_returned	تمييز tool output عن proof
verification_method	فحص post-condition
sensitive_fields	redaction/secret boundary
unsupported	منع hallucinated capability

AT06 ContractForge يقترح توليد contracts من schemas والوثائق، لكن المحكمة تمنع اعتماد عقد مستنتج إذا لم يتم اختباره على tool حقيقي. MCP مفيد هنا لأنه يجعل tool schemas واكتشاف القدرات جزءاً من البروتوكول. [E4]

28. Plan DAG: التخطيط كاعتماد ودليل وتراجع

Evidence-Aware Plan DAG

يجعل Plan DAG من أنواع nodes مثل Approval، Test، Tool Action، Transform، Retrieve، Observe وWriteback. كل node يحمل failure handler، success predicate، cost/risk، authorization، expected effect، precondition وrollback reference. يمكن تشغيل القراءات المستقلة بالتوازي، بينما mutations المتعارضة تُسلسل. [M1]

Ready(node) = Preconditions ∧ EvidenceDeps ∧ Authorization ∧ ResourceBudget

عند تغير الحالة المرصودة، لا يُجبر النظام على إكمال خطة stale؛ يعاد compile للرسم. هذا يهم في agentic coding لأن commit جديد أو dependency update أو فشل tool يمكن أن يبطل assumptions خلال دقائق.

- Critical path = value/risk aware, not time only.
- Evidence from one node unlocks dependent nodes.
- Checkpoint nodes precede mutations.
- Approval nodes are first-class.
- Rollback graph stores inverse or recovery path.

29. Executor Loop: التنفيذ المنضبط من الحالة إلى الذاكرة

Reference Executor Loop

INTERPRET → DISCOVER → CONTEXT → GAP_COURT → ROUTE → PLAN → CHECKPOINT → EXECUTE → VERIFY → WRITEBACK

المنفذ المرجعي يبدأ read-only، يبني context packet من مؤشرات لا من corpus كامل، ثم يمرر المهمة على Gap Court، يختار skills/vessels/tools، يبني DAG، يفحص preconditions، ينشئ checkpoint، وينفذ خطوات محدودة. بعدها يفتش side effects، يشغل tests، ينقد، يصلح أو يرجع، ثم يكتب world model و[M1] learning candidates.

حالة النهاية ليست نصاً حراً بل enum: VERIFIED_SUCCESS / PARTIAL / BLOCKED / ROLLED_BACK. هذا بسيط لكنه جوهري: يمنع النموذج من إعادة صياغة فشل جزئي كإنجاز كامل في التقرير.

Trace minimum	القيمة
tool_id	أي قدرة نفذت
input_digest	ما نوع المدخل
affected_objects	ملفات/خدمات/موارد
tests	ما تم اختباره
authorization	من سمح وبأي طبقة
rollback_ref	كيف نرجع
evidence_grade	قوة الدليل

30. المحاكم: فجوة، **Patch**، اختبار، أمن، وعدم ضياع

Adversarial Courts for Software Work

المحكمة ليست persona للبلاغة؛ هي module يملك سؤالاً وفاصل فشل. Gap Court تبحث عن تعريفات/أدلة/حدود ناقصة. Patch Court تبحث عن scope drift وتغيير غير سببي. Test Court تحاول بناء حالة تكسر المرشح. Security Court تبحث عن permission expansion و secret leakage. Non-Loss Court تفحص هل فقدت migration بيانات أو source lineage.

المحكمة	أقوى سؤال
Gap	ما المعلومة التي لو غابت يصبح التنفيذ تخميناً؟
Patch	أي ملف تغير بلا causal reason؟
Falsification	ما الاختبار الذي يجب أن يفشل لو كانت الفرضية خاطئة؟
Security	هل توسعت الصلاحية أكثر من الحاجة؟
Rollback	هل يمكن الرجوع فعلياً أم فقط نظرياً؟
Provenance	هل نعرف من أين جاء artifact/dependency؟

المحكمة نفسها يجب أن تخضع لـ ablation. إذا زادت critique tokens من دون كشف أخطاء أو تحسين repair، تُبسط. هذا يمنع «الهدم الأدائي» الذي يستهلك وقتاً بلا أثر.

31. World-Effect Verification: بعد tool success ماذا حدث فعلاً؟

Independent World-Effect Verification

AT11 WorldEffect Verifier يفصل execution evidence عن effect evidence. إذا قال «API «200 OK يجب فحص المورد الذي يُفترض أنه تغير؛ إذا انتهى deploy بنجاح تُقرأ version/health؛ إذا نفذت migration تُفحص invariants؛ إذا عدّل agent ملفاً تُقرأ diff ويُشغل test؛ وإذا حجز موعد أو أرسل رسالة يجب التحقق من النظام الخارجي عندما تكون الأداة تسمح بذلك.

Effect class	Readback
repository	git diff/status + tests
build	artifact hash + attestation
service	version + health + endpoint test
database	schema/data invariants + transaction state
browser	DOM/state/assertion + trace
cloud	provider read + policy
schedule	registered job + next run
memory	registry readback + provenance

هذه الطبقة هي أقرب ما يكون إلى «الحقيقة التشغيلية» في SWE-as-OS. ليست كل آثار العالم قابلة للتحقق بالكامل، لذلك يسجل النظام unknown أو supported بدلاً من اختلاق certainty.

32. الذاكرة، **Trace**، وإعادة البناء

Self-Reconstruction Memory & Trace

الذاكرة الدائمة لا تحفظ الحوار كله. تحفظ delta قابل لإعادة البناء: ماذا كان الهدف؟ ما الحالة؟ ما القرار؟ ما tool/skill؟ ما النتيجة؟ ما فشل؟ ما الإصلاح؟ ما البيئة؟ وما الذي يستحق الترقية؟ هذا ينسجم مع Self-Reconstruction Memory في corpus ويمنع تحوّل الذاكرة إلى مقبرة معلومات.

ما يُحفظ	طبقة الذاكرة
goal/status/effect	Mission ledger
sources/tests/readbacks	Evidence ledger
failure code + causal hypothesis	Failure ledger
diff + affected objects	Patch ledger
skill/ability versions + performance	Capability ledger
fingerprint + dependencies	Environment ledger
baseline/candidate/holdout/canary	Promotion ledger

يجب أن تكون الكتابة append-only أو versioned عندما يتعلق الأمر بالدليل والسلالة. التلخيص يمكن أن يتغير، لكن raw evidence أو hash الخاص به لا يُستبدل بلا أثر migration. هذه نقطة مركزية لإمكان audit والتعلم العلمي.

33. Runtime Fingerprinting قبل أي mutation

Runtime Discovery & Environment Fingerprinting

يؤكد تقرير MOUSSAIXT أن النظام يجب أن يكتشف executable/version، profile، config paths، skill directories، tool interfaces، connectors وال model endpoints، serving engines، OS، shell، Python/container، GPU/VRAM، project roots قبل التعديل. لا تُخترع مفاتيح إعداد غير موجودة. [M1، ص37]

في SWE-as-OS نوسع البصمة لتشمل package manager/lockfile، compiler/interpreter، test runner، build graph، container runtime، secret mechanism، CI provider، git state، network policy. تُرفق البصمة بالـtrace لأن نفس patch قد يفشل بسبب البيئة لا بسبب reasoning.

EnvFingerprint = Hash(OS, RuntimeVersions, Toolchain, Lockfiles, Config, Capabilities, Policy, RepoState)

Default — الاكتشاف قراءة فقط. عدم وجود قدرة ينتج installation/adaptation plan، لا نجاحاً مخترعاً.

34. MOUSSAIXT كغلاف تنفيذ عابر للمضيفين

MOUSSAIXT Cross-Environment SWE Shell

MOUSSAIXT يملك state synchronization، context packaging، capability negotiation، adapter discovery وevidence normalization. يمكن أن يعمل فوق cloud API، browser agent، local terminal، Hermes، Claude Code، Codex-like coding harness أو runtime خاص. النواة لا تعتمد على أسماء أوامر المضيف. [M1، ص38؛ M2]

Host	Adapter responsibility
Coding CLI	files/shell/tests/worktree
IDE agent	editor state/diagnostics
Hermes	memory/tools/schedules/subagents
MCP client	tool/resource/prompt discovery
Cloud agent	sandbox/network/secrets
Browser agent	DOM/action/trace
CI runner	jobs/artifacts/attestations

المقياس الحاسم هو semantic portability: هل نفس mission contract وacceptance predicates ينتجان أثراً مكافئاً عبر مضيفين مختلفين ضمن حدود القدرات؟ إذا احتاج كل مضيف إلى prompt مختلف جذرياً، فطبقة portability ضعيفة.

worktrees، containers، sandbox twins :العزل .35

Isolation & Counterfactual Sandboxes

AT08 Sandbox Twin يجعل التغيير المرشح يعيش أولاً في بيئة معزولة قدر الإمكان: git worktree، container، ephemeral namespace، test database أو browser context. العزل ليس مجرد أمان؛ هو أداة سببية تسمح بمقارنة baseline/candidate تحت شروط أقرب إلى التطابق. في البرمجة، counterfactual مهم: هل passing test سببه patch فعلاً أم dependency cache أو state سابق؟ لذلك يجب أن تحتوي suite على fresh-run حيث يلزم، وتنظيف state، وإعادة تشغيل selected tests. Playwright يستخدم browser contexts للعزل، وNix/containers يمكن أن يحسنا قابلية إعادة الإنتاج، مع إدراك أن لا بيئة مثالية تماماً. [E9]

- Worktree للرقع المتوازية.
- Container/toolchain pinning.
- Ephemeral DB/schema clone.
- Namespace isolated deployment.
- Network/secret restrictions.
- Baseline replay بنفس fingerprint.

36. CI/CD، سلسلة التوريد، والمصدرية

CI/CD & Software Supply-Chain Governance

لا ينتهي عند SWE-as-OS merge. release pipeline جزء من world effect. GitHub reusable workflows تسمح بتوحيد إجراءات build/test، وartifact attestations تضيف claims مشفرة عن provenance. تُستخدم هذه الأفكار كمرساة لـ AT13 Dependency Sovereignty وScanner وAT15 Rollback Ledger وAT18 Benchmark Court. [E5]

Gate	مثال
Source pinning	commit/tag/digest
Dependency integrity	lockfile + hash + SBOM when applicable
Build reproducibility	environment fingerprint
Test gate	unit/integration/security
Artifact provenance	attestation/signature
Deployment gate	approval + canary
Post-deploy	health/version/readback
Rollback	known-good artifact

لا تفترض الورقة أن attestation يثبت أن البرنامج صحيح؛ هو يثبت سلسلة بناء/مصدرية محددة. مرة أخرى، لا عبور بين رتب الدليل بلا جسر.

37. Observability: trace موحد للتفكير والتنفيذ

OpenTelemetry-Compatible Observability Plane

يقترح AT14 TraceMesh تحويل أحداث effects، tests، tool calls، plan nodes، routing وrepairs إلى spans مترابطة. يميز OpenTelemetry بين logs، metrics، traces ويستخدم context propagation للربط عبر الخدمات؛ يستفيد من هذا النموذج لكن يضيف SWE-as-OS attributes معرفية مثل evidence_grade وgoverning_question وselected_skill وrollback_ref. [E6].

Signal	MOUSSAIF SWE usage
Trace	end-to-end mission + plan nodes
Metric	success/latency/cost/regression/router regret
Log	tool/runtime events + failure codes
Context	mission_id/evidence chain
Profile	resource hot spots for heavy tasks

لا تُرسل الأسرار أو محتوى حساس raw إلى telemetry. يجب تطبيق classification، redaction وretention policy. كما يجب الفصل بين trace تشغيلي يحتاج سرعة وledger بحثي يحتاج provenance أعمق.

38. التنفيذ الدائم للمهام الطويلة

Durable Agentic Workflows with Temporal-like Semantics

المهام البرمجية طويلة الأفق قد تمتد لساعات أو أيام وتتعرض لانقطاع agent أو runner. Temporal يبين نموذج durable execution المبني على event history واستعادة workflow بعد failure. لا تقترح الورقة اعتماد Temporal إلزامياً، بل استعارة activities، semantics: state durable ذات retries مضبوطة، وفصل orchestration عن side-effect execution. [E7]

مفهوم **durable**	ترجمة **MOUSSAIF**
Workflow history	mission/plan trace
Activity	tool action
Retry policy	tool-contract-aware retry
Timeout	bounded step
Replay	self-reconstruction
Schedule	HyperLoop/automation trigger
Compensation	rollback/inverse action

المشكلة الخاصة بالوكلاء أن إعادة replay لقرار LLM قد تكون غير حتمية. لذلك يجب أن يُعاد replay للـstate machine والأحداث الحاسمة، بينما model calls تُسجل كقرارات/outputs مع hashes/versions؛ عند الحاجة يعاد الاستدلال كفرع جديد لا كإعادة كتابة التاريخ.

39. Kubernetes, Infrastructure-as-Code كأوعية حاكمية

IaC and Orchestrated Automation

Terraform يقدم بنية write-plan-apply التي تناسب GEET وDPCT: plan هو دليل مسبق على التغيير المتوقع لكنه ليس world-effect بعد؛ apply ينفذ؛ readback يتحقق. Kubernetes Jobs تمثل مهاماً تنتهي، وCronJobs تمثل تكراراً مجدولاً صريحاً. [E8, E10]

تقنية	دور SWE-as-OS
Terraform plan	candidate effect preview
Terraform apply	authorized mutation
Provider readback	world effect
K8s Job	bounded execution unit
CronJob	scheduled HyperLoop trigger
Deployment rollout	state transition + health gates
Policy-as-code	authorization/constraint layer

AT04 Polyglot Vessel Router يجب أن يفضل IaC declarative عندما يمنح plan/reconciliation أقوى من shell imperative، لكن ليس دائماً؛ توجد عمليات تشخيص أو bootstrap يكون shell أو API أنسب لها. المعيار evidence/risk/cost لا الأيديولوجيا.

40. أتمتة المتصفح كأثر قابل للتتبع

Browser Automation & E2E Verification

يضيف Playwright browser-context isolation، locators، auto-waiting وtrace viewer. هذه الخصائص تجعل المتصفح وعاءً مهماً لـ world effect لأن كثيراً من الأنظمة لا توفر API كامل. [E9]

في MOUSSAIXT يجب ألا يُعامل click كدليل نجاح. Tool action يسجل locator/intent، ثم verifier يفحص DOM أو URL أو network response أو stored state. عند flakiness، لا يكفي إعادة المحاولة بلا نهاية؛ retry budget مرتبط بدلالة action، خاصة إذا كانت النقرات غير idempotent مثل الدفع أو الإرسال.

الخطوة	بوابة
Locate	user-facing selector/contract
Action	authorization + idempotency
Wait	actionability/expected state
Assert	DOM/API/business predicate
Trace	screenshot/network/timeline
Rollback	undo/cancel where possible

41. MCP كطبقة ربط مع Tool Contracts

MCP Integration into MOUSSAIXT

MCP 2026-07-28 يحدد transports، prompts، resources، tools واكتشاف القدرات ضمن بروتوكول JSON-RPC. هذا يجعله مناسباً كواحد من adapters، لكنه لا يستبدل عقد MOUSSAIF: نحتاج فوق schema الخارجي إلى evidence، reversibility، authorization tier، class، post-condition verifier و[E4] risk policy.

MCP primitive	MOUSSAIF mapping
tools/list	capability discovery
tool schema	Tool Contract base
resources	HyperKB/context sources
prompts	prompt templates / not authority
transport	host-specific channel
authorization	connector boundary
notifications	trace/state updates

AT05 HyperKB Code Retriever قد يستعمل MCP resources، وAT04 Router قد يستعمل tools/list، لكن unsupported capability يجب أن يبقى صريحاً. MOUSSAIXT لا «يتظاهر» بأن الأداة موجودة بسبب تشابه اسم أو prompt.

42. الأدوات النظرية 01–09

Theoretical Tooling Set I

ID	الأداة	Implementation preview
AT01	GoalContract Compiler	goal → typed mission/constraints/acceptance/authorization
AT02	DeltaCode Extractor	observed vs desired → active causal delta
AT03	MCD-SWE Router	rank skills/tools/compute by value, proof, cost, risk
AT04	Polyglot Vessel Router	choose language/runtime vessel + fallback
AT05	HyperKB Code Retriever	minimal provenance-bearing context packet
AT06	ContractForge	generate/validate tool, skill, ability, API schemas
AT07	PlanDAG Compiler	dependencies/evidence/approval/rollback graph
AT08	Sandbox Twin	isolated worktree/container/counterfactual environment
AT09	Patch Court	scope, causal relevance, migration and hidden-effect attack

هذه الأدوات لا تُقدّم كمنتجات مكتملة. لكل أداة يجب بناء schema، وحدة اختبار، baseline بديل، observability، failure codes وablation. يمكن تنفيذ النواة المرجعية أولاً في Python أو TypeScript ثم إثبات portability عبر adapters؛ لا يوجد سبب نظري لربطها بلغة واحدة.

قاعدة **AT01–AT09** — كل أداة يجب أن تخفض uncertainty أو risk أو cost أو تزيد verified effect. إذا لم تغيّر قراراً أو اختباراً فهي مرشح للحذف.

43. الأدوات النظرية 10–18

Theoretical Tooling Set II

ID	الأداة	Implementation preview
AT10	Test/Falsification Court	adversarial tests + ablations + acceptance attack
AT11	WorldEffect Verifier	independent post-condition readback
AT12	Runtime Fingerprinter	OS/toolchain/config/capability discovery
AT13	Dependency Sovereignty Scanner	dependency/provenance/integrity graph
AT14	TraceMesh	normalized traces/metrics/logs/evidence
AT15	Rollback Ledger	checkpoints/inverse operations/recovery proof
AT16	Trace-to-Skill Distiller	verified trace → candidate HyperSkill
AT17	Ability/Loop Crystallizer	composition + holdout promotion
AT18	AGI-SWE Benchmark Court	multilingual/host-transfer/world-effect eval portfolio

AT18 يجب أن يكون الحاكم التجريبي لبقية الأدوات: benchmark لا يقيس فقط pass rate، بل silent failure، effect closure، rollback، security regression، latency، cost، transfer، trace completeness، success، وrouter regret. هذه المقاييس تمنعنا من إعلان تفوق لأن النظام أنتج ملفات أكثر أو استخدم خطوات أكثر.

44. معاينة التنفيذ المرجعي: نواة Python/TypeScript محايدة المزود

Reference Implementation Preview

تقترح الورقة reference runtime صغيراً لا يساوي MOUSSAIF كله. يتكون من schemas، registry، router، DAG executor، adapter interface، verifier، trace store وlab promotion. يمكن كتابة core orchestration في Python أو TypeScript، على أن تكون contracts مستقلة عن اللغة. JSON Schema/YAML

Module	Interface
mission	MissionContract
registry	Skill/Ability/ToolRegistry
router	route(mission,state)->RoutePlan
planner	compile_delta()->PlanDAG
executor	execute(node,adapter)->Evidence
verifier	verify(effect,predicate)->Grade
trace	emit(span/event)
memory	()append_verified_delta
evolution	candidate→benchmark→court→canary

Adapter = {discover(), read(), write(), run(), test(), schedule(), browser(), verify_capability()}

الخطوة الأولى ليست بناء 150 skill handlers يدوياً، بل تنفيذ 12–20 skill primitives عالية القيمة واختبار العقود. بقية catalog في الحزمة هو corpus للتوجيه والتدريب والتجريب، ويجب ألا يخلط بين specification وimplemented runtime.

45. شجرة المستودع والعقود الآلية

Repository Layout & Machine-Readable Contracts

المسار	المحتوى
core/contracts	mission/evidence/effect/status schemas
core/router	MCD/QMCD candidate ranking
core/planner	Delta → Plan DAG
core/executor	bounded state machine
core/courts	gap/patch/test/security
*/adapters	host-specific bridges
*/skills	atomic contracts + handlers
*/abilities	composition specs
*/loops	bounded workflow specs
*/evals	frozen cases + ablations
*/telemetry	TraceMesh/OpenTelemetry mapping
*/memory	append-only ledgers
*/lab	distill/benchmark/canary/promotion

كل object يحمل schema_version وprovenance. تغيير prose لا يستوجب migration، أما تغيير معنى status أو effect class فيحتاج version migration. يُحفظ التاريخ backward-readable قدر الإمكان كما يطلب التقرير الهندسي. [M1، ص48]

حالة التنفيذ — هذه الصفحة خطة preview لدراسة وتنفيذ لاحق؛ الحزمة المرفقة تقدم schemas/corpora/specifications وليست إثباتاً أن runtime الكامل يعمل الآن.

46. التقييم: من SWE-bench إلى AGI-SWE Closure Bench

Evaluation Portfolio

SWE-bench يقدم أساساً واقعياً لإصلاح GitHub issues، وVerified يضيف subset بشري التحقق، وMultilingual يوسع اللغات. [E11] تقترح الورقة عدم استبدال هذه المقاييس، بل إضافة طبقات فوقها لتقييم ما لا يظهر في patch pass rate.

المحور	**Metric**
Correctness	resolved/pass rate
World effect	verified-effect rate
Silent failure	false-success rate
Scope	changed-files/causal-delta ratio
Rollback	recovery success/time
Transfer	same contract across languages/hosts
Provenance	trace + artifact completeness
Security	critical regression rate
Efficiency	cost/time/steps
Learning	holdout gain after skill promotion

AGI-SWE Benchmark Court يجب أن يحتوي أربع عائلات: browser/cloud tasks، runtime/deployment tasks، repository tasks، long-horizon durable workflows. ويجب أن يضم failure injection: tool timeout، stale state، dependency mismatch، partial deployment، flaky test، secret boundary، non-idempotent retry.

47. التدريب: من **traces** إلى **curriculum** متعدد الرتب

Training and Post-Training Methods

تقترح الحزمة training assets لاستخدامها في SFT/DPO/RL أو تدريب router منفصل. أهم تمييز هو بين positive trace متحقق و negative fluent trace يبدو جيداً لكنه يفتقد gate. يمكن تدريب model على تصنيف الحالة PARTIAL/BLOCKED بدلاً من مكافأة الإجابة الحاسمة دائماً.

الهدف	الطريقة
تمييز proof عن appearance	Contrastive trace pairs
التزام schema/authorization	Tool-contract SFT
تشخيص الفشل لا إعادة المحاولة	Failure-code curriculum
اختيار أقل vessel/skill كافٍ	Router preference
توليد falsifier/repair محدد	Court generation
اختيار safe recovery	Rollback training
عدم كتابة نجاح غير متحقق	Memory promotion DPO
إغلاق البوابات بكلفة معقولة	Long-horizon RL

يجب منع leakage بين trajectory وholdout عند تقييم تبلور المهارات. كما يجب قياس forgetting وتلوث الذاكرة؛ النجاح الحالي لا يبرر ترقية قاعدة تسبب regressions في مهام قديمة.

48. برنامج **Ablation**: هل كل طبقة تستحق تكلفتها؟

Ablation & Falsification Program

Ablation	السؤال
RuntimeFingerprint-	هل تزيد config hallucination؟
WorldEffect-	هل يرتفع false success؟
PatchCourt-	هل يزيد scope drift؟
RollbackLedger-	هل يطول recovery؟
PolyglotRouter-	هل تتدهور host/language transfer؟
TraceMesh-	هل تقل causal diagnosis؟
PromotionGate-	هل تزيد skill regressions؟
HyperKB minimal retrieval-	هل ترتفع context cost/noise؟
QMCD branches-	هل يفشل حل ambiguity؟
Durable state-	هل تضيع المهام الطويلة بعد failure؟

يجب حساب Net Utility لا accuracy وحدها: تحسن صحيح لكنه يضاعف التكلفة عشر مرات قد لا يكون مقبولاً. وتقترح الورقة ` Verified Value per Cost` كمقياس تشغيلي مركب، مع عدم إخفاء المقاييس الأصلية خلف رقم واحد.

VVC = VerifiedEffect × ConstraintFidelity × Recovery / (Latency + ComputeCost + ToolCost + HumanReviewCost)

49. خارطة تنفيذ ودراسة لمدة 12 شهراً

Implementation & Study Roadmap Preview

الفترة	المخرجات	بوابة الانتقال
0–6 أسابيع	schemas + registry + Runtime Fingerprinter + trace spec	integration tests
6–12 أسبوع	reference router/executor + 12 core skills	repo benchmark
3–4 أشهر	host adapters + sandbox + world-effect 5 verifier	cross-host equivalence
4–6 أشهر	courts + rollback ledger + CI provenance	silent-failure reduction
6–8 أشهر	trace distiller + ability registry	holdout promotion
8–10 أشهر	polyglot suite + multilingual benchmarks	transfer gain
10–12 أشهر	durable workflows + canary evolution lab	reproducibility + no critical regression

الدراسة تبدأ صغيرة: 50–100 tasks مجمدة، ثلاثة مضيفين، خمس لغات/أوعية عالية القيمة، ثم نتوسع. لا يوجد مبرر لاختبار 40 لغة من اليوم الأول. catalog الكبير يُستخدم كخريطة تغطية، بينما التنفيذ المرجعي ينمو وفق evidence of value.

بوابة النضج المقترحة: Stage 1 = contracts؛ Stage 2 = typed executor/router؛ Stage 3 = host adapters/sync؛ Stage 4 = benchmark/trace mining؛ Stage 5 = capability promotion؛ Stage 6 = domain world-effect instrumentation. هذا يعيد صياغة roadmap الموجودة في [M1، ص49] داخل SWE.

50. الخاتمة والمراجع والعقد النهائي

Conclusion, References & Final Contract

تقدم هذه الورقة SWE-as-OS كطبقة هندسة برمجيات موجهة نحو AGI داخل MOUSSAIF XT: تبدأ بالسؤال الحاكم والفرق النشط، تختار وعاءً برمجياً وأقل تركيب من المعرفة والمهارات والأدوات، تنفذ عبر DAG محدود ونقطة حفظ، تمرر patch على المحاكم، تتحقق من الأثر في العالم البرمجي، ثم تكتب delta موثقاً وتسمح بتبلور القدرة فقط بعد benchmark وcanary وتراجع. النظريات الخمس المقترحة ليست حقائق مكتملة؛ قيمتها في قدرتها على توليد تنبؤات واختبارات وهندسة قابلة للهدم.

العقد النهائي — لا يُقاس تقدم MOUSSAIF XT بكثافة المصطلحات أو ميغابايتات الـ HyperKB أو عدد HyperSkills، بل بنسبة التغييرات الصحيحة المتحققة والقابلة للاسترداد والنقل والتفسير تحت قيود حقيقية.

المراجع الداخلية: [M1] MOUSSAIF AGI OS / MOUSSAIXT Deep Engineering Architecture, 50p. [M2] MOUSSAIF Unified AGI Research Book. [M3] AGI Oriented Knowledge-Base as Cognitive OS. [M4] Cognitive AI Part II. [M5] ParallaxWorld-MCD .FULL AGI OS. [M6] Fattah Sovereign Kalam Engine

مراجع هندسية خارجية 2026: [E1] OpenAI, Unrolling the Codex Agent Loop. [E2] OpenAI Codex SDK. [E3] Anthropic Claude Code: subagents, hooks, MCP. [E4] Model Context Protocol Specification 2026-07-28. [E5] GitHub Actions reusable workflows & artifact attestations. [E6] OpenTelemetry signals/context propagation. [E7] Temporal durable execution/workflows. [E8] HashiCorp Terraform core workflow. [E9] Playwright testing/traces. [E10] Kubernetes Jobs/CronJobs. [E11] SWE-bench .Verified/Multilingual

المخرجات المصاحبة: 5 HyperKB ≈40MiB لكل واحد، 40 HyperAbility ≈30MiB إجمالاً، 150 HyperSkill ≈20MiB إجمالاً، وأصول AGI/SWE إضافية ≈67MiB، مع manifests وSHA-256 وسجل مصادر. هذه أصول تنفيذ/تدريب/تقييم؛ التحقق العلمي يبدأ بعد تشغيل البرنامج المقارن المقترح.