هندسة الهارنس الوكيلي نحو AGI
الجزء الثالث

من Harness ناجح إلى Cognitive Operating Substrate: التطور الذاتي، السياق الافتراضي، تعدد الهارنس، والاختبار الإنتاجي

دراسة معمقة لأوراق وكتب وrepositories وأطر تقييم إضافية: Prime Agent وNVIDIA AVO/OpenShell وHermes وCodex وClaude Code، مع CoALA وMemGPT وACE وAgent Lightning وVoyager وReflexion وAI Scientist-v2 وOSWorld وWebArena؛ ثم عشر نظريات جديدة وأدوات/بروتوكولات لتحويل AL RAZZI + MOUSSAIFXT إلى harness قابل للقياس والتطور.

RLM · Continual Harness · AVO · Evolution Lineage · Secure Runtime · EVCF · MCAS · Context Paging · Computer Use · Agent Lightning · World Model · Epistemic Ledger · HyperKB · HyperSkill · HyperAbility · HyperLoop · MUT · MUG · MOUSSAIFXT · AL RAZZI

Zouhair Moussaif / زهير مساعف
Research & Engineering Synthesis
September 2026 · Arabic RTL · Amiri · A4 · 45 pages 05

انضباط الادعاء: «AGI-oriented» هنا برنامج بحث وهندسة، لا إثبات AGI. كل inheritance من نظام خارجي يُفصل إلى mechanism قابل للتدقيق، وكل نظرية جديدة في الصفحات 36-42 تُعامل كفرضية هندسية لها falsifier وbenchmark، لا كحقيقة مكتملة.

الأطروحة: الهارنس كوسط تشغيل معرفي لا كغلاف حول النموذج

HARNESS AS COGNITIVE OPERATING SUBSTRATE · CLAIM DISCIPLINE

يبدأ هذا الجزء من نتيجة عملية: تحسن النموذج لا يفسر وحده تحسن الوكيل. نفس model يمكن أن ينتج نظاماً هشاً أو منتجاً بحسب شكل الذاكرة، action surface، العزل، الاستئناف، التحقق، وسياسة السياق. لذلك نعرّف Harness بوصفه الوسط الذي يحول prior احتمالي إلى مسار سببي قابل للفحص: يثبت goal، ي materialize context، يختار action، يحفظ state، يقرأ world-effect، ويعيد بناء نفسه بعد الفشل. هذه الصياغة تتقاطع مع CoALA في الذاكرة/action space/decision process، ومع SWE-agent في أثر ACI، ومع Prime Agent في RLM/Continual Harness، لكنها تتوسع داخل MOUSSAIF إلى sovereignty + evidence + reconstruction. [E01,E04,E13]

الأطروحة الأقوى ليست «اجمع كل features»، بل **افصل المسؤوليات بحيث يمكن قياس كل طبقة وإبدالها.** النموذج ليس الذاكرة؛ chat history ليست state؛ tool success ليس world-effect؛ sandbox ليس approval؛ summary ليس knowledge؛ multi-agent ليس بالضرورة تعاوناً؛ وself-modification ليست learning قبل وجود held-out evaluation. لذلك تبقى قاعدة MOUSSAIF الداخلية: observed state + explicit authority + evidence + verification تتقدم على model confidence. هذا يتفق مع السلم الداخلي E0-E5 الذي يفصل artifact عن التشغيل المحلي وعن المقارنة المتكررة. [M01-M03]

تعريف تشغيلي

Agency_Runtime = ModelPrior × StateContinuity × ContextAddressability × ActionGrounding × Verification × Recovery × Governance.
ليست معادلة فيزيائية؛ هي decomposition للاختبارات. إذا حذفنا عاملاً ولم يتغير الأداء، نخفض مكانته المعمارية. وإذا ظهر أثره فقط في long-horizon أو external-effect tasks، نحدد مجال صلاحيته بدلاً من تعميمه.

معيار الثورة
ليس عدد المزايا، بل أن تنتقل الخبرة من trajectory إلى policy/skill candidate ثم تثبت تحسناً على held-out missions دون زيادة false-success.

سؤال الجزء III
ما الحد الأدنى من kernel/knowledge/evidence contracts الذي يسمح باستبدال Prime أو Hermes أو Codex أو Claude دون فقد mission continuity؟

```
Harness Value = Δ Verified Completion - λ1(False Success) - λ2(Context Cost) - λ3(Recovery Cost) - λ4(Security Risk)
```

تعميق سببي وإنتاجي

الآلية السببية المفترضة. فصل قدرة النموذج عن سببية الهارنس وقياس state/context/action/verification كعوامل مستقلة. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. تضخم الطبقات والcoordination overhead. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. factorial ablation على نفس model/budget. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. عقد methodology لبقية البحث. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract
كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract
كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

خريطة المصادر: 5 طبقات من الأدلة بدل قائمة أدوات

REPOSITORIES · PAPERS · BENCHMARKS · BOOKS · INTERNAL CORPUS

تُنظم المصادر هنا إلى خمس طبقات. الأولى **production/open-source harnesses**: Prime Agent، Hermes، Codex، Claude Code، OpenCode، Cline، Goose، Gemini CLI، OpenHands، Pydantic AI Harness، Deep Agents، NVIDIA OpenShell/NeMo. الثانية **cognitive/learning papers**: Reflexion، ACE، MemGPT، CoALA، Generative Agents، Voyager، Agent Lightning. الثالثة **autonomous research**: AI Scientist-v2، Agent Laboratory. الرابعة **environments/benchmarks**: GAIA، SWE-bench-family، TheAgentCompany، WebArena، OSWorld، AgentBench. الخامسة **corpus الداخلي**: HyperSkills، HKOS، MUT/MUG، EVCF/MCAS، Courts، World-Effect، Abilities/Loops، MOUSSAIFXT/MCAI/HAKIM/QMCD/MCD.

طبقة الدليل	ماذا تقدم	ما لا نسمح باستنتاجه
Repos تشغيلية	contracts، state models، permissions، adapters، concrete failure modes	الشعبية أو عدد stars لا يثبت superiority
Papers	hypotheses + controlled evaluations + ablations	benchmark success لا يثبت general intelligence
Benchmarks	real environment effects، functional ground truth	leaderboard واحد لا يثبت robustness خارج distribution
Books/engineering texts	تجميع patterns، production concerns	ليست empirical evidence مستقلة
MOUSSAIF corpus	نظرية داخلية، integration vocabulary، invariants	لا نرفع E2/E3 إلى E4/E5 بلا تشغيل/مقارنة

بروتوكول inheritance

لكل feature خارجي نسجل: Source → Primitive → Local Equivalent → Gap → Adoption Type → Test → Failure Boundary. أنواع adoption: DIRECT_DEPENDENCY، OPTIONAL_PLUGIN، ADAPTER، ALGORITHM_ADOPTION، INVARIANT_ADOPTION، REFERENCE_ONLY. الافتراضي هو REFERENCE_ONLY حتى يثبت أن dependency أقل كلفة من reimplementation.

هذا التنظيم يمنع Frankenstein architecture. يمكن أن نرث من Prime فكرة persistent programmable context، من AVO lineage/search، من OpenShell secure runtime، من Hermes skill/memory operations، من Codex typed event/approval boundaries، ومن Claude session/tool/context discipline؛ لكن **MOUSSAIF يظل مالك mission state وauthority/evidence.** بذلك يصبح تعدد الهارنس federation تحت microkernel، لا nested uncontrolled autonomy.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. triangulation بين repo/paper/benchmark/corpus. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. تعارض versions واتساع README claims. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. source-provenance matrix وfreshness. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نخفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. العقد الأعلى ثابت: upstreams.lock + ArchitectureDecision → Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract
كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract
كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

Prime Agent: RLM يحول السياق من نص إلى مساحة قابلة للبرمجة

PRIME AGENT · RECURSIVE LANGUAGE MODEL · PERSISTENT PYTHON KERNEL

Prime Agent يقدّم واحدة من أكثر الأفكار مباشرة لمشكلتنا: Recursive Language Model (RLM) يعامل prompt/context كمتغيرات يمكن فحصها برمجياً، ويجعل subagents calls داخل persistent REPL. النتيجة أن عمليات file/search/shell/context/delegation لا تحتاج أن تتحول كلها إلى عشرات schemas ظاهرة للنموذج في كل turn؛ يمكن أن تبقى خلف kernel صغير. كما يمكن للجلسة أن تستمر في daemon وتُعاد attach بعد انقطاع terminal. [E01]

الترجمة إلى AL RAZZI ليست نسخ IPython. المطلوب **HyperKernel contract** يفصل التحكم البرمجي عن sandbox. EVCF/MCAS يحتفظ بالموارد الكبيرة؛ HyperKernel يرسل computation إلى resource ويعيد view صغيرة إلى PEK. بهذه البنية تصبح million-line logs أو repo ضخم أو agent trajectory «قابلة للعنونة» دون ادعاء infinite model context.

```
ctx = await hyperkernel.context.mount("repo://current")
hits = await hyperkernel.search(ctx, query="auth failure")
result = await hyperkernel.compute(hits, op="rank_by_dependency")
child = await hyperkernel.delegate(goal="independent diagnosis", context=result.refs)
```

ما نرثه

Small model-facing surface؛ persistent programmatic state؛ recursive agents كـfunctions؛ detach/attach؛ compaction مع persistent goals.

ما نطوره

نجعل كل computation/effect يحمل provenance وauthority؛ context handles جزء من MCAS؛ subagents لا يملكون implicit permission inheritance؛ kernel ليس security boundary.

RLM Benefit Test: same model + same task, compare raw-tool schemas vs HyperKernel on tool-selection errors, resident tokens, recovery latency, verified completion.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. programmatic context فوق handles. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. kernel privilege وstate غير قابل لإعادة البناء. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. kill/restart + tool-surface ablation. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. HyperKernel فوق SecureExecutionPlane. العقد الأعلى ثابت: Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery → Mission Contract → Active Difference → Context/Capability Routing. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract
كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract
كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

Prime Continual Harness: التعلم خارج token history

CONTINUAL HARNESS · REFINEMENT SNAPSHOTS · IMMUTABLE BASE

يفصل Prime بين RLM runtime وContinual Harness كحالة دائمة قابلة للتعديل: subagent specifications، skills، memories، supplemental prompt notes، مع refinement events وsnapshots/rollback؛ والـbase system prompt يظل immutable. هذا الفصل بالغ الأهمية: إذا كانت كل الخبرة داخل transcript فهي تضيع بالـcompaction، وإذا سُمح بتعديل constitution بحرية يصبح self-evolution انزلاقاً غير قابل للحكم. [E01,E02]

داخل MOUSSAIF نستبدل «refine current trajectory ثم edit prompt» بمسار أكثر صرامة: Trajectory → ExperienceCompiler → Candidate → Held-out Eval → Canary → Promotion → Court. Candidate قد يكون KnowledgePatch أو HyperSkill أو RoutingPolicy أو ContextPolicy. لا يجوز لمجرد نجاح مهمة واحدة أن يعمم الخبرة؛ Generalization Court يختبر boundary conditions وcounterexamples.

Immutable / Evolvable Split

Immutable: authority, secret isolation, evidence semantics, no-loss, claim discipline. Evolvable: PEK policies, skill selection, agent specs, router weights, retrieval strategies, recovery recipes.

خطر semantic drift
Skill evolves ثم يتغير المقصد الأصلي. الحل: contract + regression tests + semantic constraints + reversible version lineage.

خطر overfitting
الـmemory التي تسجل solution لمهمة معينة قد ترفع النجاح على replay وتخفض transfer. لذلك تقاس learning gain على held-out tasks.

النتيجة النظرية: **التعلم في harness يجب أن يكون تحريراً في artifact قابل للمقارنة، لا تغيراً غامضاً في «شخصية الوكيل».** كل mutation يصبح diff، كل diff له eval، وكل promotion له سبب وrollback pointer.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. خبرة تتحول artifacts قابلة للـdiff. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. self-confirming memory وskill drift. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. held-out transfer + rollback. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. ExperienceCompiler يكتب Candidates فقط. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract
كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract
كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

NVIDIA AVO: الوكيل يصبح Variation Operator لا مولد candidate فقط

AVO · AUTONOMOUS EVOLUTIONARY SEARCH · LINEAGE + EXECUTION FEEDBACK

تقدم ورقة NVIDIA Agentic Variation Operators (AVO) تحولاً من evolutionary search ذي mutation/crossover ثابت إلى agent يقرأ lineage وdomain KB وexecution feedback ثم يقترح ويصلح وينقد ويتحقق من implementation edits. في تجربة attention kernels على B200، استمر search سبعة أيام وأبلغت الورقة عن تفوق محدود لكنه حقيقي على baselines expert-engineered في configurations المقاسة، مع transfer سريع إلى GQA. الأهم لنا ليس أرقام kernel، بل **شكل حلقة البحث**: variation نفسها تصبح agentic procedure متصلة بالدليل. [E03]

نعمم ذلك إلى EvolutionPlane لا يقتصر على code: Candidate يمكن أن يكون router policy، context materialization policy، HyperSkill، tool strategy، plan أو harness genome. لكن objective يجب أن يكون executable قدر الإمكان؛ دون scorer/benchmark يصبح AVO مجرد brainstorm loop.

```
Candidate_{t+1} = V(ParentLineage, FailureEvidence, DomainKB, Objective, Constraints)
```

AVO × MCD/QMCD

MCD يحدد أين توجد أعلى difference/impact؛ QMCD يحتفظ بمسارات متنافسة؛ AVO يولد variation داخل المسار المختار؛ Courts تفصل «تحسن objective» عن regressions/security.

Supervisor response

غيّر representation/model/harness/context/evaluator أو وسع hypothesis set؛ لا ترفع عدد turns فقط.

Stagnation signal

plateau score، نفس same plan entropy، failure signature منخفض، ContextFault، oscillation، أو actions دون verified delta.

فاصل الادعاء: نجاح AVO في optimization domain لا يثبت AGI. لكنه دليل قوي على أن **agent** + **persistent lineage** + **executable objective** + **long horizon** يمكن أن يحول self-improvement من rhetoric إلى search process قابل للقياس.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. agentic variation من lineage وexecution feedback. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. objective gaming وlocal maxima. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. progress slope + Pareto + stagnation tests. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. EvolutionPlane يستخدم WorldEffect/Courts scorers. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract

كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract

كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

NVIDIA OpenShell: الأمن يجب أن يعيش تحت الهارنس

SECURE RUNTIME · DECLARATIVE POLICY · SANDBOX != APPROVAL

OpenShell يصرح بتصميم واضح: safe/private runtime لوكلاء مستقلين، مع sandboxes تحكم filesystem/network/process/inference عبر credential، declarative policies، isolation وobservability. هذه الفكرة تمنع خطأ شائعاً في أنظمة self-evolving: وضع security policy في prompt الذي يستطيع agent نفسه تعديله أو إهماله. [E05]

داخل MOUSSAIFXT نميز ثلاث طبقات: **Authority** يقرر هل الفعل مسموح من منظور mission/user؛ **Sandbox Policy** يفرض تقنياً ما تستطيع العملية فعله؛ **WorldEffect Verifier** يفحص ما حدث فعلاً. approval بلا sandbox لا يمنع compromise؛ sandbox بلا approval قد يسمح بأفعال غير مرغوبة ضمن الحدود؛ وكلاهما بلا verification لا يثبت النتيجة.

المجال	Policy example	Evidence بعد التنفيذ
Filesystem	workspace write فقط؛ secrets paths deny	diff + hash + unexpected-write scan
Network	allowlist domains/ports	network event + response provenance
Process	CPU/time/process tree caps	exit + descendants + resource use
Credentials	no prompt exposure؛ scoped injection	secret reference used, not value logged
Inference	approved endpoints/models	provider trace + model/version

Windows reality

OpenShell يذكر WSL2 كمسار تجريبي على Windows؛ لذا AL RAZZI لا يعتمد عليه وحده. SecureExecutionPlane يحتاج NativeWindows restricted subprocess + optional WSL/Docker/OpenShell adapters، وكل adapter يعلن capabilities وhealth بدلاً من fake parity.

هذه الصفحة تربط الأمن بـMUG: الحاكمية ليست نصاً أعلى فقط، بل **سلطة قابلة للإنفاذ.** أي AGI-oriented OS يرفع autonomy دون enforcement beneath harness يزيد القدرة أكثر من الثقة، وهو اتجاه معاكس لهدفنا.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. security enforcement تحت harness. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. permissive/restrictive policy وWindows parity. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. filesystem/network/credential escape tests. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. SecureExecutionPlane متعدد adapters. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract

كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract

كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

NeMo Agent Toolkit وPydantic AI Harness: القدرة كحزمة typed قابلة للقياس

INSTRUMENTATION · CAPABILITIES · HOOKS · FRAMEWORK-NEUTRAL OPTIMIZATION

تتحرك أدوات NVIDIA NeMo Agent Toolkit نحو instrumentation/evaluation/trajectory interchange، بينما يفصل Pydantic AI بين Harness وcore primitives. الفكرة المشتركة: capability يمكن أن تجمع capabilities: capability + instructions + model settings + tools + lifecycle hooks دون تعديل framework core. **القدرة ليست persona ولا prompt فقط، بل package ذات contract.** [E06,E07]

هذا يعزز تعريف HyperSkill/HyperAbility. HyperSkill يجب أن يكون atomic contract: inputs، preconditions، tools، authority، outputs، postconditions، evidence، failure/recovery. HyperAbility هي composition تُقيّم ككل. ولمنع تضخم framework، capabilities غير الأساسية تبقى plugins يمكن promotion إلى core فقط إذا أثبتت broad utility وثبات interface.

ATIF / trajectory interchange

اعتماد format قياسي أو adapter لـtrajectories يفتح مقارنة harnesses وتغذية evaluator/trainer دون ربط AL RAZZI بإطار واحد. المسار: Runtime Events → normalized trajectory → verifier/replay/training.

Capability discovery

Router يكتشف candidate capabilities بحسب mission + authority + measured reliability. لا يرى كل Agent plugin.

Core graduation rule

ميزة تدخل core فقط إذا كانت مطلوبة عبر domains/harnesses، ولها stable semantics ولا تزيد coupling.

النتيجة: engineering quality تأتي من **typed composition** + **telemetry** + **evaluation** لا من طول system prompt. وهو ما يسمح لاحقاً لـUI أو ACP client أن يـ inspect نفس capability state دون قراءة النصوص الحرة.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. typed capability packages. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. plugin sprawl وschema drift. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. capability-use telemetry + core ablation. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. HyperSkill/Ability manifests source of truth. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract

كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract

كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

Hermes Agent: closed learning loop عملي للذاكرة والمهارات

HERMES · MEMORY · SKILL CREATION · SUBAGENTS · CRON · PROVIDER INDEPENDENCE

Hermes يقدم مثالاً عملياً على agent «ينمو» عبر sessions: persistent memory، search في المحادثات السابقة، creation/improvement للـcron، MCP، subagents، skills، terminal backends متعددة، messaging surfaces وprovider independence. هذه ليست AGI بحد ذاتها؛ لكنها تمثل **operational continuity** أكثر نضجاً من chat-only agents. [E08]

تكاملنا لا يجعل Hermes هو MOUSSAIF. بل HermesAdapter يعرض capabilities إلى MOUSSAIFXT، وHermesCapabilitySource يسمح بفحص skill/memory assets وتحويل المفيد منها إلى authority، candidates. Mission state وEvidence Ledger تبقى في AL RAZZI. هذه sovereignty rule تمنع فقدان state إذا تغير upstream أو provider.

Closed Learning Loop

Experience → memory/skill candidate → reuse. نضيف: provenance → sandbox test → held-out eval → Generalization Court → versioned promotion.

Session recall
لا نعيد transcript كاملاً؛ retrieval يختار decisions/failures/outcomes المرتبطة بالمهمة الحالية.

Progressive disclosure
Skill metadata/description أولاً، ثم full instructions فقط عند activation؛ نفس مبدأ EVCF paging.

ميزة Hermes المهمة لـWindows هي breadth في backends/tooling، لكن الاختبار الحاسم لدينا هو portability: نفس Mission Contract يجب أن ينجز على Native أو Hermes دون أن يتغير semantics للـsuccess. إذا اختلاف harness غيّر معنى evidence أو authority فهذا coupling يجب إزالته.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. persistent memory/skills/subagents. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. memory pollution وskills غير سببية. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. reuse/transfer gain + decision-memory precision. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. Hermes host لا يملك mission state. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract
كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract
كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

Hermes Self-Evolution: من execution traces إلى variants قابلة للمراجعة

DSPY + GEPA · TRACE-DRIVEN MUTATION · GUARDRAILS

مشروع Hermes Agent Self-Evolution يستخدم DSPy + GEPA لقراءة execution traces واقتراح variants للـskills/prompts ثم تقييمها تحت constraint gates. المستودع يوضح أن كل evolved variant يجب أن يمر باختبارات، حدود حجم، semantic preservation وhuman review قبل الدمج. هذه بالضبط الصيغة التي تحول «self-evolving» من شعار إلى pipeline. [E09]

نأخذ منه مبدأ **failure explanation as mutation signal**: evaluator لا يكتفي بأن المهمة فشلت؛ يحاول تحديد failure locus: retrieval؟ context؟ tool schema؟ planner؟ verifier؟ action execution؟ ثم mutation يستهدف locus بدلاً من random prompt rewriting. داخل failure locus، MCD يأخذ priority إذا كان يتكرر أو يملك impact مرتفعاً.

```
trace -> failure_signature -> causal_hypothesis -> patch_candidate
      -> sandbox_eval -> held_out_eval -> court -> canary -> promote/rollback
```

Self-evolution safety

لا يمكن للـcandidate رفع authority ceiling أو تعديل immutable constitution تلقائياً. Capability improvement ≠ permission increase. Architecture/security class E5 يحتاج independent review وربما human gate.

مقياسنا ليس «عدد skills المتولدة»، بل TransferGain: هل reuse يقلل cost أو يزيد verified completion على مهام جديدة من نفس/مجال قريب؟ Skill تُحجر إذا ارتفع regression أو side effects.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. trace-grounded mutation يستهدف failure locus. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. eval leakage وbenchmark overfit. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. held-out + canary + cross-domain transfer. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. GeneralizationCourt قبل promotion. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract

كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract

كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

Codex: الفصل بين core runtime وapp-server وواجهة العميل

CODEX · RUST CORE · APP-SERVER · JSON-RPC · UI-INDEPENDENT STATE

معمارية Codex الحالية تقدم lesson مباشر لقرارنا CLI-first: core runtime منفصل عن app-server/event protocol وعن app-server. يرسل TUI/clients server-initiated approval requests مرتبطة بـthread/turn، ويترك للعميل تقديم diff/command ثم الرد بقرار. معنى ذلك أن الواجهة ليست مصدر الحقيقة؛ **runtime events هي الحقيقة التي تعرضها الواجهة.** [E10]

AL RAZZI يجب أن يحافظ على نفس الفصل: Command Bus + Event Bus + Runtime Core + protocol versioning. لاحقاً Windows UI أو ACP/AG-UI bridge يتصل بنفس state. ممنوع أن تصبح progress bars مصدراً للحالة، أو أن يحتاج UI إلى screen-scrape CLI text.

Codex lesson	Moussaif mapping	اختبار
Thread/turn scoped events	mission_id / execution_id / trace_id	restart + event reconstruction
Approval request protocol	AuthorityRequest	decline/modify/resume without corruption
Item lifecycle	DAG node / ToolIntent lifecycle	no orphan RUNNING items
Skills injection	PEK capability materialization	only relevant skills resident

Protocol invariant

أي client يستطيع إعادة بناء: ماذا طلب المستخدم؟ ماذا اختار router؟ ماذا نفذ tool؟ ما الذي وافق عليه الإنسان؟ وما evidence؟ إذا كان الجواب يعتمد على console prose فقد فشل الفصل المعماري.

هذا يسمح لنا باستضافة Codex نفسه كـHarnessAdapter لاحقاً دون تكرار semantics. Codex يصبح execution substrate، بينما AL RAZZI يحتفظ mission closure وknowledge/evidence lineage.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. event protocol يجعل UI projection. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. event loss أو duplicate commands. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. idempotency + cursor resume + version tests. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. Command/Event schemas داخل microkernel. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract
كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract
كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

Codex: sandbox والapproval وworktree كعقود مختلفة

SANDBOX POLICY · AUTHORITY · HYPERBRANCH · DIFF-FIRST

أحد أنضج الدروس في coding harnesses هو أن **sandbox policy لا يساوي user approval**. sandbox يحدد القدرة التقنية؛ approval يحدد الشرعية في سياق المستخدم؛ Git/worktree يحدد transactional isolation؛ tests/verifier يحددان الصحة. دمج هذه الوظائف في «safe mode» «boolean» يضيع semantics. [E10]

نطور HyperBranchManager: قبل mutation يسجل base commit + dirty workspace + unrelated changes؛ عند parallel agents ينشئ worktrees، ويربط كل branch بـowner/mission/candidate. بعد التنفيذ ينتج merge candidate، test receipts، diff artifact، وrollback path. لا نسمح للagent أن يعتبر كتابة الملفات success قبل /run verification.

Four gates

1) Authority: هل يجوز؟ 2) Sandbox: هل يستطيع؟ 3) Transaction: هل يمكن عزل/الرجوع؟ 4) Verification: هل تحقق المطلوب؟

Failure case

Patch صحيح لكنه مسح تغييرات user dirty: Functional test PASS، No-Loss Court REJECT.

Failure case

Command exit=0 لكن كتب ملفاً خاطئاً: Tool success PASS، WorldEffect FAIL.

```
Engineering Success = Functional Correctness ∩ No-Loss ∩ Authorized Effect ∩ Reconstructible Evidence
```

تعميق سببي وإنتاجي

الآلية السببية المفترضة. isolation + authority + verification كtransaction. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. merge conflicts وstale base. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. dirty workspace + conflict injection. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. HyperBranch يربط candidate بالbase fingerprint. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract

كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract

كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

Claude Code: session discipline، tool policy، MCP، وlong-task reconstruction

CLAUDE CODE · SUBAGENTS · HOOKS · MCP · SESSION RECONSTRUCTION

Claude Code يبرز كمرجع في الجمع بين permission modes، MCP، hooks، subagents، project instructions، وCLI structured output/resume. الدرس الأهم للمهام الطويلة ليس مجرد model quality بل **reconstructability from the filesystem**: تجعل fresh context قادراً على معرفة الحالة بدلاً من الاعتماد على task artifacts/tests/setup scripts hidden chain. [E11]

في MOUSSAIF نرفع هذه الفكرة إلى Reconstruction-First Runtime: أي mission طويلة يجب أن تترك enough durable artifacts لإعادة تكوين state: Mission Contract، branch state، evidence refs، checkpoints، remaining gaps، current node، DAG snapshot، وcompaction. process handles يحذف prose لكنه لا يحذف هذه anchors.

Hooks → Policy/Event Hooks

before_tool، after_tool، tool_failure، before_compaction، before_closure، after_checkpoint الخارجي لا يملك typed lifecycle events. Hook تتحول إلى authority افتراضية، يمر عبر trust policy.

MCP discipline
Server مكتشف ≠ server موثوق. Discovery/Trust/Enablement حالات منفصلة، وكل tool يعاد تغليفه في ToolGateway.

Subagent isolation
Child يحصل minimal context + explicit return contract، ويعيد findings/evidence/artifacts/gaps بدلاً من full internal trajectory.

هذه الهندسة تلتقي مع MUG: meta-governance يقرر متى يرفع authority ومتى يطلب human takeover، بينما MUT يسمح بتعدد hypotheses/plans قبل commit.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. fresh-context reconstruction من artifacts. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. untrusted hooks وrecursive side effects. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. context reset + hook failure + resume. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. Lifecycle hooks تمر EventPolicy. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract
كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract
كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

OpenCode وCline وGoose وGemini CLI: portability كقيمة معمارية

CLIENT/SERVER · ACP · MCP · PERMISSIONS · MULTI-PROVIDER

تضيف هذه العائلة درساً جماعياً: **agent engine يجب أن ينجو من تغير الواجهة والنموذج**. OpenCode يفصل agents/permissions/skills؛ Cline يعرض CLI/headless/ACP وsession resume؛ Goose يجمع CLI/desktop/API مع providers وMCP؛ Gemini CLI يربط approvals/sandbox/tool discovery مع ACP وMCP. لا نحتاج نسخ كل implementation، لكن convergence على typed interfaces إشارة قوية. [E12]

Primitive	المغزى	AL RAZZI
Headless JSON/JSONL	automation لا تعتمد على terminal text	stable Event schema
ACP	editor ↔ agent interoperability	ACP adapter خارج core
MCP	tool/context interoperability	ToolGateway mediated
Provider abstraction	model replaceability	UniversalProviderFabric
Permission profiles	least privilege	Authority + sandbox split

Polyglot Harness Rule

external harness لا يدخل mission core؛ يظهر كـHarnessAdapter له discover/health/capabilities/start/observe/cancel/checkpoint. وبذلك يمكن node واحدة أن تنتقل من Hermes إلى Codex أو Native دون تغيير definition of done.

الخطر هو subagent → Prime → Hermes → Claude → Moussaif → nested chaos... لذلك نضع DelegationDepth وbudget tree وsingle canonical owner للmission state. external harness يعيد structured result؛ لا يرث قدرة spawn غير محدودة.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. protocol/provider portability. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. lowest-common-denominator adapters. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. same Mission عبر adapters. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. native extensions بلا تسريب semantics للcore. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract
كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract
كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

SWE-agent: الواجهة نفسها جزء من الذكاء التشغيلي

AGENT-COMPUTER INTERFACE · EDIT/SEARCH/LINT FEEDBACK

ورقة SWE-agent تقدم دليلاً مباشراً على أن تصميم Agent-Computer Interface (ACI) يغير أداء agent في software engineering: أدوات تحرير/بحث/تنفيذ مصممة للagent تقلل أخطاء interaction وتجعل feedback أقرب إلى locus. هذا يعارض تصور أن «أقوى model = raw shell + الحد الأعلى». [E13]

نعممها: لكل domain يمكن أن يبني HyperAbility **minimal sufficient action vocabulary** مع typed outputs. Browser tasks تستخدم DOM/accessibility قبل pixel clicks؛ data tasks تستخدم SQL/DuckDB قبل dumping rows؛ repo tasks تستخدم AST/LSP/Git قبل free-form grep؛ computer use يفضّل API/CLI/DOM/A11y ثم vision كـfallback.

Action Surface Quality ∝ Valid Action Rate × Feedback Locality × Reversibility × Observability

ACI Factory proposal

من Mission schema + environment capabilities + risk policy يولد action surface محدوداً؛ إذا احتاج agent escape hatch، يطلب capability expansion عبر AuthorityRequest.

Transfer
اختبر أن ACI لا تصبح overfit: task خارج pattern يجب أن يطلب توسعة بدل hallucinated workaround.

Ablation
Raw shell vs domain ACI: invalid actions، wrong-file edits، recovery steps، tool tokens.

في علم harness المقترح، interface design نفسه يصبح variable في benchmark، لا background ثابتاً. هذه نقطة مهمة لـMCDJ لأنها تحول «أين نوزع compute؟» إلى «هل يجب أن نغير action representation قبل أن نزيد reasoning؟».

تعميق سببي وإنتاجي

الآلية السببية المفترضة. action-space design يقلل search entropy. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. over-specialized ACI. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. structured ACI vs raw shell. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. ACI Factory من HyperAbility/environment. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract
كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract
كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

Aider Repo Map: الضغط البنيوي أفضل من stuffing

GRAPH-AWARE REPOSITORY CONTEXT · TOKEN BUDGET · STRUCTURAL RELEVANCE

Aider يبني repo map موجزة بأهم classes/functions/signatures ويرتب بحسب graph relevance تحت token budget. الدرس يتجاوز coding: **أفضل context representation يحافظ على العلاقات التي تتنبأ بالفعل، لا على أكبر كمية نص.** في code، import/reference graph أحياناً أهم من embedding similarity. [E14]

EVCF/MCAS يطبق المبدأ على موارد متعددة: retrieval. symbol graph، test dependency graph، process graph، evidence graph، claim lattice، agent lineage. المقترحة: hierarchy deterministic metadata → exact symbol/ID → structural graph → lexical search → semantic retrieval → LLM synthesis. كل مرحلة أرخص وأكثر قابلية للتحقق من التي بعدها.

Context invalidation

بعد patch لا يكفي refresh vector embedding. يجب invalidation لـsummaries، tests impacted، dependencies، symbol signatures، وcached PEK fragments المرتبطة بالhash القديم.

Virtual context
المورد الكامل يبقى searchable/queryable/compute-able خارج model window.

Resident context
النموذج يرى slice عالي القيمة + handles إلى بقية MCAS.

```
Context Utility Density = Verified Decision-Relevant Information / Resident Tokens
```

تعميق سببي وإنتاجي

الآلية السببية المفترضة. structural graph يضغط repo. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. stale graph ولغات ضعيفة الدعم. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. repo scale + invalidation + wrong-file rate. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. RepoMap mount داخل MCAS. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract
كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract
كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

OpenHands وDeep Agents وPydantic: runtime صغير، capabilities قابلة للتوصيل

AGENT SDK · WORKSPACE ABSTRACTION · DURABLE GRAPH · CAPABILITY PACKAGES

OpenHands يعرض agent SDK/workspaces يمكن تشغيلها محلياً أو في containers/Kubernetes؛ Deep Agents يبني harness فوق LangGraph ويستفيد من state/checkpoints/interrupts، بينما Pydantic AI Harness يفصل capability library عن core. التقاطع: **runtime primitive صغير + optional capability layer** أكثر قابلية للتطور من monolith. [E07,E15]

نقترح Microkernel Rule: Core يحتوي فقط Mission/Event/Authority/State/Tool/Evidence/Checkpoint contracts. HyperKB/skills/browser/computer/research coding تأتي plugins. أي feature جديدة يجب أن تثبت أنها cross-domain لكي تدخل core. هذه القاعدة تحارب التضخم الذي قد يجعل AL RAZZI أبطأ من mini-agent بسيط.

Core	Plugin/Capability	لماذا الفصل
Mission state	coding/research packs	domain churn لا يكسر kernel
Event bus	telemetry/exporters	observability replaceable
ToolGateway	MCP/browser/DB	external protocols تتغير
Evidence	domain verifiers	success semantics domain-specific
Checkpoint	host adapters	durability لا ترتبط بprovider

Ablation governance

لكل طبقة فوق microkernel benchmark ablation. إذا لم تضف verified completion/recovery/security/context efficiency في مجالها، تُبسط أو تبقى optional.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. microkernel صغير + capabilities. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. middleware conflicts. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. plugin isolation/lifecycle failure containment. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. Registry يعلن requires/conflicts/enhances. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract

كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract

كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

CoALA: من LLM إلى cognitive architecture ذات memory وaction space

COALA · MODULAR MEMORY · INTERNAL/EXTERNAL ACTIONS · DECISION CYCLE

CoALA ينظم language agents عبر structured action space، modular memory، وgeneralized decision process. أهميته أنه يعيد الوكيل إلى تاريخ cognitive architectures: model ليس agent كاملاً؛ agent هو loop بين memory/reasoning/actions وال environment. [P01]

MOUSSAIF يضيف فوق هذا التنظيم ثلاث طبقات: **epistemic governance** (claim/evidence rank)، **world-effect closure** (success كحالة مرصودة)، و**evolvable capability graph** (HyperSkill/Ability/Loop candidates). CoALA يصلح baseline abstraction لاختبار إن كانت إضافاتنا تعطي gains أم أسماء جديدة فقط.

Memory taxonomy extension

RuntimeState ⫤ HyperKnowledge ⫤ ProceduralMemory ⫤ ProjectMemory ⫤ MissionMemory ⫤ WorkingMemory. عمليات write/read/forget لها policies مختلفة؛ منع dumping كل transcript في «memory» واحد.

External action
tool/computer/API effect ويتطلب authority/evidence.

Internal action
retrieve, reflect, compare, compact, hypothesize.

نربط ذلك بـMUT/MUG: MUT يحتفظ بالمنافسة بين representations/hypotheses، بينما MUG يقرر أي معيار يملك authority على الانتقال من internal cognition إلى external action. هذه صياغة قابلة للتنفيذ بدل استخدام «الحاكمية» كاستعارة.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. memory/action/decision decomposition. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. governance قد تكون إعادة تسمية. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. CoALA baseline vs full Moussaif. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. كل مفهوم يحتاج measurable delta. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract
كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract
كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

MemGPT → EVCF: context window كـRAM لا كحد معرفي للنظام

VIRTUAL CONTEXT MANAGEMENT · HIERARCHICAL MEMORY · INTERRUPTS

MemGPT اقترح virtual context management مستلهماً hierarchical memory في OS: نقل البيانات بين memory tiers يمنح agent قدرة على العمل فوق موارد تتجاوز context window. هذا هو أقرب precursor مباشر لـEVCF، لكن V3/V5 لدينا يوسع الفكرة من نص/ذاكرة إلى **hybrid addressable resources**: repo، DB، artifact، evidence، agent result، process، browser state، graph. [P02]

المبدأ الصارم: **Virtual Context ≠ Infinite Model Context**. model لا «يعرف» كل mounted resource؛ runtime يضمن addressability وpaging، ويُسجل ما تم materialize فعلاً. أي claim يعتمد على resource غير محمل يجب أن يملك evidence pointer أو ContextFault يُحل قبل الاستنتاج.

```
ctx://repo/current/symbol/AuthService
ctx://evidence/mission-42/test-7
ctx://dataset/customer/events?window=7d
ctx://browser/session-3/state/latest
```

Compute-to-context

بدلاً من PEK → 500k rows → prompt: query/aggregate locally → 28 rows + statistics + provenance. هذا يخفض tokens ويزيد reproducibility.

Virtualization Gain = Bytes Addressable / Resident Tokens; but Quality = Useful Pages / Pages Materialized

تعميق سببي وإنتاجي

الآلية السببية المفترضة. hierarchical virtual context. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. paging thrash وretrieval hallucination. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. large-resource exactness + page-fault tests. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. MCAS يجعل access صريحاً. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract

كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract

كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

ACE: السياق Playbook يتطور دون context collapse

AGENTIC CONTEXT ENGINEERING · GENERATION · REFLECTION · CURATION

ACE يعامل context كـplaybook متطور عبر generation/reflection/curation، ويستهدف مشكلتين: brevity bias الذي يحذف التفاصيل، وcontext collapse الناتج من repeated rewriting. التقرير يقدم مكاسب على benchmarks مختلفة، لكن القيمة المعمارية لنا هي أن **context نفسه artifact قابل للتطور والاختبار.** [P03]

نحوّل ذلك إلى ContextPolicyCandidate: ليس فقط content edits، بل rules عن متى نجلب KB، ما الذي pin، متى compact، وأي representation نستخدم. كل PEKTrace يسجل usage/effect proxy، reason، token cost، units injected؛ trajectory mining يكتشف units التي تتكرر بلا أثر أو missing context الذي سبب failure.

Anti-collapse rule

لا نلخص authority decisions، failed approaches، acceptance criteria أو unresolved gaps إلى prose قابلة للضياع؛ تحفظ typed objects مستقلة، بينما summaries يمكن rebuild منها.

Promotion
لا يعتمد على shorter prompt فقط؛ يجب ألا يرتفع false-success أو gap-at-closure.

Context mutation
Candidate يغير retrieval/materialization policy في shadow replay.

```
Context Policy Score = Task Success + Transfer - Token Cost - Missing-Critical-State - Stale-Context Errors
```

تعميق سببي وإنتاجي

الآلية السببية المفترضة. evolving context policies. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. playbook growth وincorrect lessons. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. context ablation + stale lesson injection. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. ContextPolicyCandidate governed مثل skill. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract
كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract
كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

Reflexion، Generative Agents، Voyager: ثلاث مدارس للتعلم بلا weight update

EPISODIC REFLECTION · MEMORY SYNTHESIS · EXECUTABLE SKILL LIBRARY

Reflexion يحسن trial التالي عبر verbal feedback محفوظ في episodic memory، Generative Agents يبني observation→reflection→planning فوق سجل خبرات، Voyager يجمع automatic curriculum + growing executable skill library + iterative feedback/self-verification. هذه الأعمال توضح أن harness-level learning يمكن أن يحدث دون fine-tuning، لكن كل نوع memory له failure modes: reflection قد يرسخ تفسيراً خاطئاً، summary قد يضخم salience، skill code قد يصبح brittle. [P04-P06]

العمل	آلية التعلم	ترجمة Moussaif
Reflexion	verbal feedback → episodic memory	FailureLesson candidate + scoped confidence
Generative Agents	experience record → higher reflection	ProjectMemory/Claim graph + provenance
Voyager	curriculum + executable skill library	HyperSkillFactory + transfer eval

Causal promotion

لا نعتبر memory مفيدة لأن agent ذكرها. نقيس intervention: mission مع lesson/skill مقابل بدونها على held-out tasks؛ إذا لم يتحسن outcome أو recovery فهي غير مثبتة.

هذه الصفحة تؤسس لنظرية Causal Skill Compilation لاحقاً: skill لا تصبح أصل معرفي إلا إذا كان يمكن ربط استخدامها بفرق في world-effect، لا بمجرد تكرار pattern في transcript.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. learning عبر reflection/memory/skills بلا weight update. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. post-hoc rationalization. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. with/without lesson held-out tasks. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. Causal Skill Compilation. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract

كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract

كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

Agent Lightning: افصل agent execution عن learning substrate

TRAJECTORY → POMDP-LIKE INTERFACE · CREDIT ASSIGNMENT · RL WITHOUT REWRITE

Microsoft Agent Lightning يفصل كيفية تشغيل agent عن كيفية تدريبه: يسجل execution كـstates/actions/rewards ويحوّلها إلى data interface يمكن لل RL استخدامه مع تغييرات قليلة في agent. الخيط المهم لـ AL RAZZIL هو أن **trajectory schema يصبح boundary بين runtime والتحسين**؛ هذا يسمح بالتعلم من Hermes/Prime/Native بنفس format بدلاً من كتابة trainer لكل harness. [P07]

لا نحتاج إدخال RL في production MVP مباشرة. نحتاج أولاً trajectories نظيفة: reward/evaluator، state transition، ToolIntent/Observation، context refs، LLM call، cost، evidence، result. بعدها يمكن offline learning أو prompt/router optimization أو future RL. بدون ذلك، «self-learning» لا يملك training substrate.

Credit assignment problem

Mission success النهائي لا يخبرنا أي decision كان سببيًا. نحتاج causal proxies، step/node level evaluators، وcounterfactual replay؛ وإلا قد نكافئ calls غير مهمة أو نعاقب exploratory actions مفيدة.

Offline

Trainer consumes trajectories، builds candidates، returns versioned policies/models؛ production promotion remains governed.

Online

Runtime emits standard events، no trainer dependency.

Learning Boundary: RuntimeTrace → NormalizedTrajectory → Evaluator/Trainer → CandidateArtifact → PromotionGate

تعميق سببي وإنتاجي

الآلية السببية المفترضة. trajectory interface يفصل runtime عن trainer. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. sparse/proxy rewards. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. offline replay + reward sensitivity. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. TrajectoryNormalizer يحفظ refs/evidence. العقد الأعلى ثابت: Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery → Mission Contract → Active Difference → Context/Capability Routing. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract

كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract

كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

AI Scientist-v2: البحث العلمي كـagentic tree search

HYPOTHESIS · EXPERIMENT MANAGER · PROGRESSIVE TREE SEARCH · REVIEW LOOP

AI Scientist-v2 يوسع autonomous research إلى progressive agentic tree search: صياغة hypotheses، design/execute experiments، analysis/visualization وpaper writing، مع experiment-manager agent وVLM feedback للfigures. الورقة لا تعني أن كل scientific judgment يمكن أتمتته، لكنها تقدم harness غني لاختبار **open-ended branch management.** [P08]

داخل ResearcherOS/MOUSSAIF نأخذ tree search لكن نفرض Epistemic Ledger: كل hypothesis لها status، falsifier، required observations، predictions؛ كل experiment artifact له code/data/environment hash؛ paper claim لا يرتفع rank لأنه كُتب جيداً. Human/peer review يبقى external evidence لا مجرد style critic.

Research Mission DAG

Literature → HypothesisSet → Experiment branches → Result comparison → Falsification Court → Claim ranking → Manuscript. يمكن أن يفشل دون خسارة كل mission؛ lineage يحفظ negative results.

MUT/QMCD crossover
يحافظ على competing explanatory models حتى discriminating experiment.

AVO crossover
Variation operator يولد experiment/code candidate من lineage.

القيمة AGI-oriented هنا هي القدرة على إدارة unknowns لا إنتاج PDF: النظام يصرّح بما لا يعرفه، يصمم فعلًا يميز hypotheses، ثم يعدل world model بناء على observation.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. tree-search بحثي مع executable experiments. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. paper generation قبل validity. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. reproduction + independent review. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. Research DAG يربط claims بالexperiments. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract
كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract
كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

Agent Laboratory: الإنسان كإشارة بحثية عالية القيمة لا كعامل كتابة

LITERATURE → EXPERIMENT → REPORT · HUMAN FEEDBACK · COST/QUALITY

ينظم Agent Laboratory إلى research workflow، experimentation، literature review، وreport writing مع إمكانية human feedback في كل stage، ووجدت الدراسة أن human involvement رفع الجودة وأن framework خفض التكلفة مقارنة بطرق autonomous research سابقة في إعدادها. الدرس ليس استبدال الباحث، بل **تحديد نقاط leverage التي تستحق human cognition.** [P09]

نترجم ذلك إلى Human Attention Router: لا نسأل المستخدم عند كل tool call ولا نتركه خارج loop تماماً. MCD يرفع human gate عندما يجتمع uncertainty + high impact + weak evidence + irreversibility. سؤال الإنسان نفسه يجب أن يكون discriminating: يعرض الخيارات، الدليل، والقرار المطلوب؛ لا ينقل له transcript كامل.

Human feedback typing

Correction يمكن أن تكون: generalizable method، domain fact، preference، project policy، local decision. فقط النوع الأخير يصبح knowledge/skill candidate بعد provenance/evaluation.

```
Human Leverage = Verified Outcome Improvement / Human Minutes
```

هذا المقياس مهم لأن «full autonomy» ليس هدفاً مطلقاً. harness الأفضل قد يعرف متى لا يستقل. في production، تقليل human interrupts مع الحفاظ على false-success/security هو معيار أكثر واقعية من عدد الساعات بلا تفاعل.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. human attention عند أعلى information value. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. local preference يعمم خطأ. لذلك لا يعلن component نجاحه من داخله، closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. human-minute leverage + correction typing. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. Human Attention Router عبر MCD/risk. العقد الأعلى ثابت: Context/Capability Routing → Active Difference → Mission Contract → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract

كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract

كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

GAIA وAgentBench: العمومية تحتاج tool/web/multimodal robustness

GENERAL ASSISTANT BENCHMARKS · MULTI-ENVIRONMENT DECISION MAKING

GAIA صُمم لأسئلة واقعية تحتاج multimodality، web browsing، reasoning، tool use، وأظهر في نسخته الأصلية فجوة كبيرة بين البشر وGPT-4+plugins؛ AgentBench يجمع ثماني environments ويبرز long-term reasoning/decision/instruction following كعوائق. هذه benchmarks مفيدة لأنها تفصل «معرفة جواب» عن **تركيب الأدوات عبر steps**. [B01,B02]

لكننا لا نستخدم score واحد كـAGI metric. AL RAZZI يحتاج long-horizon، security، recovery، retrieval، research، computer، web، coding: benchmark matrix. كل طبقة جديدة يجب أن يكون لها benchmark مناسب وHyperKB ablation. مثلاً يُقاس transfer/retrieval، وليس OSWorld فقط؛ ComputerUse يُقاس functional tasks؛ Courts تُقاس false-success rejection.

Capability	Benchmark family	Moussaif metric
General tool/web	GAIA	verified answer + provenance + tool cost
Interactive agency	AgentBench	long-horizon completion + recovery
Software	SWE-bench-like	patch correctness + no-loss
Computer	OSWorld	functional state change
Web actions	WebArena	programmatic task validator

Benchmark sovereignty

Leaderboard external = evidence source، لا objective الوحيد. نحتفظ internal canary set غير مستخدم في tuning لمنع benchmark overfitting.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. multi-environment benchmarks تقيس integration. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. benchmark overfitting. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. private canaries + rotation + transfer. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. BenchmarkRegistry يفصل score عن AGI claim. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract

كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract

كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

OSWorld 2.0: Computer Use يجب أن ينتهي بتقييم الحالة لا بعدد clicks

REAL COMPUTER ENVIRONMENT · CROSS-APP · EXECUTION-BASED EVALUATION

OSWorld يقدم environments حقيقية على أنظمة تشغيل وتطبيقات مختلفة مع task setup وexecution-based evaluators، الموقع أعلن OSWorld 2.0 في يونيو 2026. القيمة المعمارية هي أن computer-use agent يقاس بحالة التطبيق/الملف الناتجة، لا بمجال trajectory. [B03]

اقترح V3 للفعل: hierarchy API → CLI → filesystem/data → DOM → accessibility/UIA → vision. OSWorld يبرر هذا: GUI grounding وحده تاريخياً كان bottleneck، بينما hybrid perception/action يسمح بإبدال primitive بحسب app. WindowsDesktopAdapter يجب أن يعلن sources المتاحة: active window، UI Automation tree، clipboard metadata، screenshot، metadata؛ OCR fallback فقط عند الحاجة.

ComputerAction contract

{intent, target, expected_effect, risk, verifier}. بعد كل action مهم: capture before/after state؛ لو click لم يغير expected property، لا نكمل كأن النجاح حدث.

Human takeover

CAPTCHA/credential/ambiguous irreversible step → WAITING_FOR_HUMAN with resumable state.

Drift recovery

popup/window move/login/slow load → re-observe → replan.

Computer Reliability = Verified State Transitions / Intended Computer Actions

تعميق سببي وإنتاجي

الآلية السببية المفترضة. computer evaluation على real state. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. GUI nondeterminism وdrift. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. multi-run variance + modality ablation. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. ComputerEffectVerifier before/after. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract

كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract

كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

WebArena: functional correctness يقتل وهم "agent looked busy"

SELF-HOSTED REALISTIC WEB · PROGRAMMATIC VALIDATION · WEB WORLD EFFECT

WebArena يبني مواقع self-hosted واقعية ويصمم tasks طويلة نسبياً مع programmatic validation للfunctional correctness. هذا يطابق فلسفة WorldEffect: browser agent لا ينجح لأنه وصل إلى صفحة أو كتب نصاً؛ ينجح عندما تحقق الحالة المقصودة في backend/DOM/data. [B04]

نطور BrowserEffectVerifier: navigation/information seeking قد يتحقق عبر answer + source; mutation tasks تحتاج state query أو DOM/backend check. كما نفصل browser prompt injection: نص الصفحة observation غير موثوق، لا يستطيع إعادة تعريف mission/authority. أي instruction من webpage تعامل كdata حتى يُثبت أنها جزء من task semantics.

Web Tool Gateway

Browser actions تمر Authority + domain allowlist + credential broker + effect class. External form submit أو delete ليس مجرد click؛ يصنف /external irreversible effect ويحتاج idempotency token إن أمكن.

Evidence

URL alone ليس evidence كافياً؛ نحفظ relevant state/result + captured_at + provenance.

Replay-safe

counterfactual replay يعيد DOM snapshots أو mock environment، ولا يعيد external POST الحقيقي.

هذه البنية ضرورية لاختبار outreach/business automation لاحقاً دون تحويل autonomy إلى click spam أو duplicate side effects.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. web functional validators. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. website drift وduplicate side effects. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. self-hosted tasks + idempotency. نختفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. BrowserAdapter + CredentialBroker. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract

كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract

كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

TheAgentCompany: الهارنس كعامل رقمي داخل مؤسسة مصغرة

WORKPLACE TASKS · WEB + CODE + COMMUNICATION · CONSEQUENT ACTIONS

TheAgentCompany يحاكي شركة برمجيات صغيرة مع مواقع وبيانات داخلية، ويطلب من agents browsing، coding، program execution والتواصل. الإصدار المنشور أظهر أن نسبة معتبرة من المهام الأصعب ظلت خارج قدرة baselines، ما يبين أن **integration friction** بين التطبيقات والسياقات أكبر من subtask quality. [B05]

من منظور MOUSSAIF، benchmark Mission OS: يجب حفظ artifacts، messaging outcomes، identity/role، project state، والdependencies بين tasks. نجاح node منفرد لا يكفي إذا workflow النهائي لم يغلق. لذلك نضيف Cross-App World Model يربط message، repo branch، entities: ticket، credential scope، calendar/event، document.

> **Workplace closure**
> Mission complete فقط إذا acceptance criteria موزعة على الأنظمة جميعاً PASS؛ مثال: code merged + test pass + message sent + issue state updated.

```
Cross-App Coherence = Satisfied Linked Preconditions / Total Cross-System Preconditions
```

هذا benchmark أيضاً يختبر authority inheritance: agent قد يكون مخولاً تعديل repo لكنه غير مخول إرسال external message، role-specific leases تمنع خلط «يمكنه تقنياً» مع «مسموح له».

تعميق سببي وإنتاجي

الآلية السببية المفترضة. cross-app workplace coherence. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. partial completion أو communication harm. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. linked criteria + per-system receipts. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. Cross-App WorldModel. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

> **Promotion contract**
> كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

> **Evidence contract**
> كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

الكتب الهندسية: من patterns إلى production constraints

AI AGENT BOOKS · LLM ENGINEERING · RELIABILITY / COST / SCALE

الأوراق تكشف mechanisms، والـrepos تكشف implementation، والكتب الجيدة تجمع production concerns. في الجزء الثالث نستفيد خصوصاً من كتب agent/LLM engineering الحديثة التي تنظم monitoring، cost، evaluation، multi-agent communication، state/persistence، orchestration، memory، tools، وdeployment. وظيفتها ليست إثبات ادعاء جديد، بل **كشف failure modes التي لا تظهر في benchmark صغير**: observability، queues، migrations، secret handling، retries، rate limits، incident response.

مرجع هندسي	القيمة لهذا البحث	ترجمة عملية
Building Applications with AI Agents - O'Reilly, 2025	components, multi-agent, reliability/cost/scaling	production checklist + protocol boundaries
AI Agents in Action - Manning, 2025/26	workflows, tools, memory, agent patterns	HyperAbility patterns
LLM Engineer's Handbook, 2025	LLMOps, eval, monitoring	telemetry/eval plane
Designing Large Language Model Applications, 2025	system design around LLM apps	provider/context/application boundaries

قاعدة الاستخدام

لا ننقل "best practice" إلى constitution. نربط كل practice بفشل محتمل واختبار. مثال: queue/backpressure تظهر ضرورتها عندما multi-agent bursts تضرب provider quotas؛ قبل ذلك تبقى engineering precaution.

جمع الورقة + book + benchmark + repo يمنع bias مصدر واحد: research يمكن أن يكون أنيقاً لكنه لا يغطي operations؛ repo قد يكون ناجحاً دون causal ablations؛ benchmark قد يختزل distribution؛ الكتاب قد يجمع خبرة دون قياس. triangulation هو المنهج.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. production failure modes من engineering literature. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. cargo-cult best practices. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. load/rate-limit/incident drills. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. controls مرتبطة measurable triggers. العقد الأعلى ثابت: Context/Capability Routing → Active Difference → Mission Contract → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract
كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract
كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

من الورقة إلى البروتوكول: طبقة interoperability الموحدة

MCP · ACP · AG-UI · A2A · ATIF · INTERNAL EVENT CONTRACT

الأنظمة الحديثة تتحرك نحو protocols، لكن لكل protocol موضع. MCP يربط agent بالأدوات/الموارد؛ ACP يربط coding agent/editor client؛ AG-UI يهدف إلى agent-user frontend event interoperability؛ A2A إلى agent/agent boundaries؛ ATIF/trajectory formats إلى evaluation/training interchange. الخطأ أن نجعل أحدها «نظام التشغيل الداخلي».

AL RAZZI يحتاج internal canonical schemas ثم adapters. **Mission/Event/Tool/Evidence contracts لا تُعرَّف وفق MCP أو UI protocol**. adapter يحول outward/inward مع versioning. بذلك لو تغير protocol لا نفقد internal semantics.

```
Internal ToolIntent <-> MCP Adapter <-> MCP Server
Internal Command/Event <-> ACP Adapter <-> Editor
Internal Event/State <-> AG-UI Adapter <-> Desktop/Web UI
Normalized Trajectory <-> ATIF Adapter <-> Evaluator/Trainer
```

Protocol Firewall

كل incoming external object يمر schema validation + trust classification + authority mapping. Protocol compliance لا يمنح trust.

Risk	Benefits
semantic leakage: external protocol limitations تتسرب إلى core. الحل canonical richer model + lossy adapter warnings.	replaceability، ecosystem access، testability.

هذه الطبقة تجعل MOUSSAIFXT فعلاً cross-environment execution fabric بدلاً من collection of scripts.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. adapters تعزل protocols. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. protocol churn وlossy translation. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. conformance + round-trip + loss warnings. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. Protocol Firewall. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract	Evidence contract
كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.	كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

checkpoints، Event sourcing، وdurable cognition

RESTART CONTINUITY · SNAPSHOT + EVENT TAIL · PROCESS HANDLES

الوكالة طويلة الأفق تنهار إذا كان terminal process هو الذاكرة. نحتاج checkpoints، durable Mission Runtime: state snapshot + event tail، عند process boundaries، artifact/evidence refs، child-agent registry، registry. هذا المبدأ حاضر بصيغ مختلفة في Codex threads، LangGraph durability، Prime daemon، وHermes sessions، ونرفعه هنا إلى contract موحد.

Checkpoint ليس copy عشوائياً للذاكرة؛ هو boundary معنوي: before court/commit، before irreversible effect، after DAG node، before mutation، plan accepted. عملية resume تعيد verification للحالة الخارجية؛ لا تفترض أن process/file/network بقي كما كان منذ checkpoint.

Reconciliation on resume

(1 load snapshot 2) replay events 3) inspect external fingerprints 4) detect drift 5) mark stale nodes/evidence 6) resume/replan هذا يمنع "resume into a different world."

Ephemeral	Durable
token stream, transient UI spinner, disposable intermediate prose unless promoted.	Mission Contract, DAG, gaps, decisions, artifact/evidence refs.

Restart Continuity = Missions Safely Resumed / Missions Interrupted

تعميق سببي وإنتاجي

الآلية السببية المفترضة. snapshot + event tail للديمومة. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. event divergence وworld drift. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. random crash + reconciliation. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. RFD يجعل restart release gate. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract	Evidence contract
كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.	كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

Secure autonomy: project injection، browser injection، secrets، supply-chain

TRUST DOMAINS · CREDENTIAL BROKER · PLUGIN TRUST · TOOL TRUST

عندما يصبح harness قادراً على قراءة repo/web واستخدام tools، كل مصدر سياق يمكن أن يحمل تعليمات adversarial. لذلك نفصل trust domains: ModelTrust، ProjectTrust، RemoteHostTrust، WebContentTrust، MCPTrust، PluginTrust. ملف AGENTS.md في repo موثوق قد يكون project instruction؛ نص في dependency README أو webpage لا يرتقي تلقائياً إلى authority.

CredentialBroker يعطي tool scoped secret في وقت التنفيذ دون إدخاله في model context أو PluginManifest. logs يعلن permissions، لكن SecureExecutionPlane ينفذها. dependency/upstream update لا يُقبل تلقائياً في harness genome؛ يمر eval + regression/security + license + lockfile.

تهديد	فشل naïve harness	Control
Repo prompt injection	model يطيع نصاً في ملف ويغير mission	trust labels + instruction provenance
Web injection	page يأمر بتسريب data	page=data + domain/tool policy
Secret leakage	API key يدخل PEK/trace	secret refs + broker + redaction
Plugin supply chain	plugin يملك shell/network	manifest + sandbox + signature/hash
Self-evolution bypass	candidate يزيل guardrail	immutable constitution + promotion court

قاعدة القدرة/السلطة

Agent يصبح أذكى أو skill يصبح أفضل لا يعني أن authority يرتفع. Capability evolution وpermission evolution مساران منفصلان؛ الثاني يحتاج مصدر سلطة خارجي واضح.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. trust labels وsecret isolation. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. indirect injection/exfiltration. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. repo/web injection + log scans. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. TrustDomain وSecretRef canonical. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract
كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract
كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

Computer/Browser Use: من pixel agent إلى hybrid resource agent

API → CLI → DOM → ACCESSIBILITY → VISION · EFFECT VERIFICATION

Computer Use يصبح أكثر موثوقية عندما لا يختزل إلى screenshot+click. نفس المهمة قد تُحل عبر accessibility tree، DOM، CLI، API أو vision. **Hybrid Resource Mode** يختار representation الأقل غموضاً والأكثر قابلية للتحقق، ويستخدم vision عندما لا توجد بنية أدق. هذا امتداد طبيعي لـprocess، DOM resource، screen state resource: EVCF resource وfile resource كلها mounted context.

ComputerAction يحمل intent/target/expected_effect/risk/verifier. بعد كل action مهم، observation جديدة تحدث WorldModel. إذا فشل loop، expected transition لا يواصل clicks عمياء؛ RemainingGap يعاد فتحه، وقد يغير action modality. مثال: button visually disabled → DOM says disabled → world model updates prerequisite gap.

Action modality router

Score = reliability_history + semantic_fit + verification_strength - ambiguity - side_effect_risk - latency. API قد يفوز في data mutation؛ DOM في web form؛ UIA في native app؛ vision في canvas/unknown app.

```
Computer Use = Perception × Action Grounding × State Verification × Recovery; clicks are only one substrate.
```

هذا يجعل AL RAZZI قابلاً لاختبار production work على Windows لاحقاً دون رهن AGI-oriented architecture بـVLM واحد.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. hybrid action representation. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. اختيار modality غير مكافئ. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. API/DOM/UIA/vision comparison. نحتفظ أبسط baseline ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. ActionModel يتعلم reliability. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract

كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract

كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

EVCF/MCAS: السياق الافتراضي كـresource fabric لا memory trick

EXTENSIBLE VIRTUAL CONTEXT FABRIC · MOUSSAIF COGNITIVE ADDRESS SPACE

نطوّر EVCF V3 إلى MCAS: كل عنصر معرفي أو تشغيلي يمكن عنونته دون أن يكون resident في prompt. الموارد تشمل evidence، datasets، KB units، symbols، source files، processes، agent results، hypotheses، claims، وcontext model. computer/browser states يصبح **materialized working set** فقط.

هذه architecture تحتاج أكثر من indexes، typed objects، URI resolver: متعددة vector DB، pin/evict، compute-to-context، cache invalidation، page faults، provenance، policies، وsemantic search. process status واحد لا يستطيع التعامل مع access policy. SQL aggregation، exact symbol، أو world-effect receipt.

Tier	أمثلة	عملية
L0 Resident PEK	goal, active gap, selected evidence	model inference
L1 Hot cache	recent pages, current branch artifacts	fast rematerialization
L2 Project graph	repo/world/context genome	structural search
L3 HyperKnowledge	KB/skills/abilities/loops	capability retrieval
L4 Structured stores	SQLite/DuckDB/JSON/logs	query/compute
L5 External/delegated	web/APIs/agents/Drive	fetch/delegate with trust

Cognitive page fault

إذا reasoning يحتاج object غير runtime، resident يسجل ContextNeed وي materialize page ثم يكمل. لا يُسمح للنموذج أن يتصرف كأنه قرأ resource كاملة لمجرد وجود handle.

MCAS objective: maximize decision-relevant information per resident token while preserving reconstructability.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. typed virtual resources. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. URI duplication وstale aliases. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. identity/hash/lineage/invalidation tests. نختفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. MCAS ResourceIdentity. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract

كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract

كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

من EVCF إلى AGI-oriented resource vascularity

CONTEXT ECONOMICS · COMPUTE-TO-DATA · MCD ROUTING · PREFETCH

الابتكار ليس فقط أن «السياق أكبر». بل أن resources لها تكلفة/سرعة/freshness/authority مختلفة، وMCD يمكن أن يوزع attention/compute عليها. نسمي ذلك Resource-Vascular Context: تدفق معرفي حسب active difference لا حسب حجم corpus. high-risk claim قد يستدعي primary source + independent verifier؛ code edit صغير قد يكتفي symbol map + tests.

prefetch يستخدم Mission DAG لتوقع next-node context؛ pin يحمي acceptance criteria/failures؛ eviction يزيل low-value summaries؛ compute-to-data يقلل materialization؛ cache invalidation يربط source hash. كل قرار يسجل في PEKTrace حتى نستطيع لاحقاً معرفة هل failure سببه model أم missing context policy.

Key distinction

Long-context model يوسع L0 فقط. EVCF/MCAS يوسع address space ويضيف computation/typing/provenance. كلاهما مفيد، لكن الثاني يمنع architecture lock-in إلى context window بعينه.

Ablation

same model: long-context stuffing vs MCAS paging vs hybrid; measure success/cost/reconstruction.

Metric

UsefulPageRate، ContextFaultRate، TokensAvoided، StaleContextErrors.

هنا يتجسد مفهوم Computational Vascularity الداخلي: الحوسبة تُوجّه إلى difference/evidence bottleneck، لا تتوزع بالتساوي على كل ملفات المشروع.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. vascular context allocation عبر MCD. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. إهمال prerequisite منخفض salience. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. hidden-prerequisite adversarial tests. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. MCD heuristic قابل للمعايرة. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو تسجل الأدلة.

Promotion contract

كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract

كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

النظرية 1 - سيادة فضاء العنونة المعرفية CASS

THEORY T1 · COGNITIVE ADDRESS-SPACE SOVEREIGNTY

CASS — Cognitive Address-Space Sovereignty

الفرضية: عمومية harness لا تعتمد فقط على سعة model context، بل على امتلاك runtime فضاء معرفياً مستقلاً عن النموذج: موارد قابلة للعنونة، access control، provenance، paging وreconstruction. نسمي هذا «سيادة فضاء العنونة»: النموذج مستهلك transient، بينما system owns cognitive addresses.

Mechanism. MCAS URI + typed resource registry + materialization policy + PEKTrace. إذا تم تبديل model/provider، تبقى identities والعلاقات والحالة. إذا فُقد model session، يعاد بناء resident set من addresses.

CASS Index = Reconstructible Cognitive Objects / Cognitive Objects Required for Mission Continuity

Prediction

في مهام تتجاوز context أو تتطلب CASS architecture، restart يجب أن تخفض context-loss failures مقارنة transcript-centric agent بنفس model.

Falsifier

إذا agent transcript+long-context يحقق نفس performance restart/large-resource بنفس أو أقل cost/complexity عبر عدة domains، قيمة CASS كطبقة مستقلة تنخفض.

Protocol

CTXREF: canonical typed context-reference schema يمكن adapters نقله دون materializing content.

Tool proposal

MCAS Resolver + ContextPageFault Profiler.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. runtime يملك cognitive state خارج model. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. إعادة تسمية DB/filesystem فقط. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. cross-model restart + >window context. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. CASS يثبت value عبر semantics/provenance/access. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability → Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract

كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract

كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

النظرية 2 - الوكالة ذات الأثر المطبوع WETA

THEORY T2 · WORLD-EFFECT TYPED AGENCY

WETA — World-Effect Typed Agency

الفرضية: reliability ترتفع عندما يكون لكل action type «توقيع أثر» يحدد expected state transition وverifier المناسب، بدلاً من أن تكون tools generic calls. الوكالة ليست sequence calls؛ هي typed transitions بين حالات العالم.

ToolIntent يتضمن EffectClass وExpectedDelta؛ ToolObservation لا يساوي success؛ EffectVerifier يقيس ObservedDelta؛ mismatch يفتح gap. هذه theory توحد process launch، email send، browser state، database row، software diff في semantics واحدة.

WETA Reliability = Verified Expected Transitions / Effectful Intentions

Prediction

typed effect contracts تقلل false-success خصوصاً في web/computer/cross-app tasks حيث API success أو click لا يضمن outcome.

Falsifier

إذا verification layer لا تخفض false-success أو تضيف latency دون gains على real environments، يجب تبسيط granularity أو استخدامها risk-based فقط.

Protocol

+ WE-Receipt: before fingerprint + intended delta + after fingerprint + evidence refs verdict.

Tools

EffectSchemaCompiler، WorldDeltaInspector.

تعميق سببي وإنتاجي

الآلية السببية المقترضة. typed world-effect transitions. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. verifier overhead. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. risk-stratified verification ablation. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. EffectClass يحدد verification depth. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Observation → Evidence → Remaining Gap → Closure/Recovery → Bounded Execution. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract

كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract

كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

النظرية 3 - سلالة تطور الهارنس EHL

THEORY T3 · EVOLUTIONARY HARNESS LINEAGE

EHL — Evolutionary Harness Lineage

الفرضية: self-improvement يصبح قابلاً للعلم فقط عندما كل mutation في harness له benchmark score، diff، hypothesis، parent، وrollback path. بدون lineage لا يمكن فصل التحسن من drift أو قياس transfer.

HarnessGenome يجمع references للـ context/router/model/agent/tool/verifier policies. CandidateGenome يولد عبر AVO-style variation أو trace mining؛ shadow replay ثم held-out benchmark ثم canary. promotion لا يزيل parent history.

```
Evolution Gain = HeldOutScore(candidate) - HeldOutScore(parent), conditioned on equal/normalized budget
```

Prediction

lineage + held-out promotion يخفض regressions مقارنة in-place prompt/skill edits ويجعل causal attribution أفضل.

Falsifier

إذا candidate lineage لا يحسن rollback/debugging/selection مقارنة versioned Git/config فقط، تُدمج النظرية في ordinary release engineering.

Protocol

EVO-CANDIDATE manifest: parent_ids, mutation_class, tests, benchmark, canary, authority.

Tool

HarnessGenomeDiff + EvolutionJournal.

تعميق سببي وإنتاجي

الآلية السببية المقترضة. evolution lineage experimental. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. metadata لا تمنع objective gaming. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. held-out + canary + rollback drill. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. HarnessGenome refs immutable artifacts. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract

كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract

كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

النظرية 4 - شبكة الادعاء الإبستيمي ECL

THEORY T4 · EPISTEMIC CLAIM LATTICE

ECL — Epistemic Claim Lattice

الفرضية: agents طويلة الأفق تحتاج تمثيل claims مستقل عن prose memory، بحيث evidence/counterevidence/freshness/scope تغير status بوضوح. هذا يمد سلم E0-E5 الداخلي إلى runtime dynamic claim graph.

Claim لا يصبح VERIFIED لأن عدة agents كرروه؛ يحتاج evidence type مناسب. contradiction لا يُحل بالsummarization بل يحتفظ بالمسارين حتى discriminating evidence. ECL يرتبط بـFSKE/MCAL في تصنيف مراتب الدليل، لكن هنا نحدده كـdata structure قابلة للاختبار.

Status	معنى runtime	Action
UNKNOWN	لا دليل كاف	retrieve/experiment
HYPOTHESIS	تفسير قابل للاختبار	predict + test
SUPPORTED	دليل جزئي	seek independent evidence
VERIFIED	criterion met ضمن scope	may drive action
CONTESTED	counterevidence قائم	QMCD preserve alternatives
STALE	source/world changed	refresh

Prediction

ECL يقلل propagation للclaims القديمة/المتناقضة في research/ops missions.

Falsifier

إذا claim graph maintenance أعلى كلفة من benefit ولا يخفض factual/action errors، نستخدمه فقط high-stakes/long-lived projects.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. claim graph مستقل عن prose memory. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. claim bloat/duplicates. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. contradiction + stale-source tests. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. HyperKB يربط claims بالمصادر. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract

كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract

كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

النظرية 5 - الإدراك متعدد الهارنس المحدود BPC

THEORY T5 · BOUNDED POLYHARNESS COGNITION

BPC — Bounded Polyharness Cognition

الفرضية: لا يوجد harness واحد يتفوق في كل substrates؛ يمكن رفع robustness بـfederation بين harnesses بشرط bounded delegation، canonical mission state، وstructured returns. القيمة تأتي من diversity دون state fragmentation.

MetaRouter يختار Native/Hermes/Prime/Codex/Claude/OpenCode حسب measured capabilities. external harness لا يرى كل MCAS؛ يحصل scoped PEK + authority lease. النتيجة تعود Result/Evidence/Artifacts/Gaps. delegation depth/budget يمنع recursive explosion.

```
Polyharness Gain = Best-Routed Verified Success - Best-Single-Harness Verified Success - CoordinationCost
```

Prediction

على mixed task suite، evidence-weighted harness routing يتفوق على fixed harness إذا performance profiles غير متطابقة عبر domains.

Falsifier

إذا coordination/context translation cost يزيل gain أو اختلاف semantics يرفع false-success، نحد federation إلى independent verifier/isolated tasks.

Protocol

/HAP - Harness Adapter Protocol: discover/health/capabilities/start/observe/cancel checkpoint/result.

Tool

HarnessCapabilityProfiler.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. complementary harness federation. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. coordination/translation overhead. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. routed vs fixed harness suite. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. MOUSSAIFXT federation وAL RAZZI mission control. العقد الأعلى ثابت: Mission Contract / Active Difference → Context → Observation → Evidence → Remaining Gap → Closure/Recovery → Capability Routing → Bounded Execution. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract

كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract

كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

النظريات 6-8: من skill causality إلى phase transition عند الركود

THEORIES T6-T8 · CSC · RVC · SISPT

T6 CSC · T7 RVC · T8 SISPT

T6 - Causal Skill Compilation (CSC)
Skill لا تُرقّى من repeated pattern فقط؛ يجب أن نثبت أن activation يسبب فرقاً إيجابياً عبر -matched/held-out tasks. Tool: SkillAblator. Protocol: SKILL EVIDENCE manifest. Prediction: يقلّ skill bloat ويرتفع transfer density.

T7 - Resource-Vascular Context (RVC)
السياق ليس buffer ثابتاً؛ موارد المعرفة/الحوسبة تتلقى bandwidth وفق :MCD: uncertainty, impact, evidence weakness, dependency centrality. Tool :ContextFlowProfiler. Prediction: نفس budget يحقق أعلى verified outcome من uniform retrieval.

T8 - Stagnation-Induced Search Phase Transition (SISPT)
عند plateau لا تزيد turns تدريجياً فقط؛ ننتقل «طورياً» من exploitation إلى :exploration: model/harness/representation/evaluator/hypothesis diversity. Tool: StagnationDetector + SearchPhaseController.

فاصل T8: transition لا يفعّل من boredom linguistic؛ يحتاج action-without-delta، score plateau، unchanged gap vector، signals: repeated failure signature، candidate diversity collapse. falsifier: إذا phase change يزيد cost دون حل stagnation عبر benchmark suite.

```
Search Phase = f(progress_slope, gap_entropy, repeated_failure, candidate_diversity, budget_remaining)
```

تعميق سببي وإنتاجي

الآلية السببية المفترضة. ربط skill/context/search بالقياس. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. تفاعل النظريات يصعب attribution. لذلك لا يعلن component نجاحه من داخله؛ يبقى closure خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. factorial ablation. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. لا meta-layer قبل إثبات primitives. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract
كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract
كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

النظريات 9-10: routing بالدليل وإعادة البناء أولاً

THEORIES T9-T10 · EWHR · RFD

T9 EWHR · T10 RFD

T9 - Evidence-Weighted Harness Routing (EWHR)

Router لا يعتمد model reputation أو static tags فقط. كل model/harness/tool/skill له PerformanceRegistry حسب task family: verified success, false success, cost, latency, recovery, court rejection. routing يوازن current capability fit مع evidence history.

```
RouteScore = Fit × Reliability × EvidenceStrength × Availability - Cost - Latency - Risk
```

T10 - Reconstruction-First Durability (RFD)

نصمم mission state بحيث fresh process يستطيع إعادة بناء العمل من artifacts/events/checkpoints قبل تحسين continuity UI. الحجة: long-horizon reliability = reconstructability، لا طول hidden context. Tool: ReconstructionTester.

Prediction T9: adaptive router يتفوق على fixed default في heterogeneous suite بعد warm-up كافٍ. **Falsifier:** history يتأخر/يخاز ويخفض performance، الحل contextual bandit أو conservative exploration، لا blind exploitation.

Prediction T10: kill/restart في random points لا يخفض completion بشكل كبير عندما durable state صحيح. **Falsifier:** state artifacts غير كافية لإعادة semantics أو stale world makes replay unsafe؛ عندها نضيف reconciliation لا مزيد logging فقط.

Protocol

mission/DAG/gaps/authority/artifacts/evidence/fingerprints /RECON-SNAPSHOT: event cursor.

Protocol

ROUTE-RECEIPT: alternatives + scores + chosen + reason + later outcome.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. القيمة لا تأتي من الاسم، بل من تحويلها إلى performance history + reconstruction laws. state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. cold-start/stale profiles. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. exploration+decay+restart benchmark. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. RouteReceipt وReconSnapshot. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract

كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract

كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

أدوات وبروتوكولات Part III + الربط المباشر بـAL RAZZI/MOUSSAIFXT

IMPLEMENTATION BLUEPRINT · 18 TOOLS · 9 PROTOCOLS · PRODUCTION TESTS

تحويل النظريات إلى engineering يتطلب artifacts محددة لا namespaces فارغة. نضع حزمة أدوات أولية قابلة للبناء داخل modules الحالية، وكل أداة يجب أن تُربط بـtest وtelemetry:

Tool	وظيفته	Protocol/Output
MCAS Resolver	resolve typed cognitive resources	CTXREF
ContextFlowProfiler	measure paging/pinning/utility	PEKTrace
WorldDeltaInspector	verify effect transitions	WE-Receipt
HarnessCapabilityProfiler	benchmark adapters	HAP profile
HarnessGenomeDiff	compare evolved harness versions	EVO-CANDIDATE
SkillAblator	test causal skill contribution	SKILL-EVIDENCE
StagnationDetector	detect plateau/failure loops	ProgressVector
SearchPhaseController	switch explore/exploit policy	PhaseDecision
RouteReceiptWriter	record routing alternatives/outcome	ROUTE-RECEIPT
ReconstructionTester	kill/resume randomized missions	RECON-SNAPSHOT
ClaimGraph	track evidence/status/freshness	ECL claim
ExperimentManager	hypothesis→test→observation	ExperimentReceipt
CredentialBroker	scoped secrets injection	SecretRef
EffectSchemaCompiler	derive verifier from tool contract	EffectSchema
HyperBranchManager	isolated code candidates	BranchReceipt
TrajectoryNormalizer	events→training/eval trajectory	ATIF-like record
GeneralizationCourt	promotion boundary for lessons	CourtResult
ChaosInjector	provider/tool/process/context failures	FailureReceipt

Production sequence: **P0 contracts** → P1 real provider/tool → P2 restart/recovery → P3 browser/computer → P4 MCAS stress → P5 cross-harness → P6 self-evolution shadow → P7 canary. لا ندخل self-evolution قبل أن تصبح trajectories والevaluation صلبة، وإلا سنطور الضجيج.

Minimum revolution gate

يجب أن يظهر تحسن مقاس مقابل baseline بسيط في 4 محاور على الأقل: recovery، context efficiency، restart continuity، false-success، verified completion، transfer. إذا لم يظهر، تُبسّط architecture حتى تتضح القيمة.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. أدوات محددة بدلاً من namespaces. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. architecture theatre عبر stubs. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. contract+runtime+test+telemetry+failure. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. breadth قبل vertical slice. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract
كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract
كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

برنامج الاختبار المقارن: كيف نثبت أن AL RAZZI أصبح أفضل فعلاً؟

BENCHMARK MATRIX · ABLATIONS · CHAOS · HELD-OUT TRANSFER · CANARY

نقترح benchmark program لا يختبر feature existence بل causal contribution. Baselines: A0 minimal ReAct/tool loop؛ A1 + repo/context map؛ A2 + HyperKB/PEK؛ A3 + WorldEffect; A4 + RemainingGap/ClosureCourt; A5 + MCAS؛ A6 + multi-harness routing؛ A7 + evolution plane. نفس model/budget قدر الإمكان، ثم نكرر على task families مختلفة.

Suite	الاختبار	المقياس الأساسي
SWE	real repo bug/feature + dirty workspace	verified patch + no-loss
Web	WebArena-like functional tasks	backend/DOM goal state
Computer	OSWorld subset/local tasks	execution-based success
Research	hypothesis/experiment/report	claim/evidence correctness
Knowledge	large corpus > model context	answer + provenance + tokens
Durability	random kill/restart	safe resume rate
Chaos	timeouts/provider/tool/db failures	recovery / false-success
Transfer	new task after learned skill/policy	held-out gain
Security	prompt injection/secret/tool bypass	violation rate

Production gate

لا تعلن Production-Grade لأن unit tests تمر. نطلب: provider failover، effect verifier، restart mid-execution، real-project mutation، real-provider E2E، at least one computer/browser flow، injection tests، وheld-out evolution gain.

Primary: VerifiedCompletion | FalseSuccess | MeanTimeToRecovery | RestartContinuity | UsefulContextRatio | CostPerVerifiedMission | TransferGain

الهدف ليس تعظيم autonomy duration. metric الأهم **verified progress per unit cost/time**. agent يعمل 12 ساعة بلا delta أسوأ من agent ينجز task في 25 دقيقة مع evidence.

تعميق سببي وإنتاجي

الآلية السببية المفترضة. benchmark matrix كاختبار causal architecture. القيمة لا تأتي من الاسم، بل من تحويلها إلى state transitions وأحداث قابلة للمراقبة: ما inputs، ما policy أو قرار routing، ما الموارد التي materialized، وما الأثر المتوقع؟ ثم يُربط ذلك بـevidence يمكن إعادة فحصها بعد الجلسة. هكذا يصبح اختلاف harnesses اختلافاً تجريبياً بدلاً من مقارنة واجهات أو claims تسويقية.

حد الفشل. comparisons غير عادلة بسبب budgets/env. لذلك لا يعلن component نجاحه من داخله؛ closure يبقى خارج producer ويستخدم WorldEffect أو Regression/No-Loss أو evaluator مناسب. الفشل المتكرر يسجل ErrorMemory مع scope ومصدره، ولا يتحول تلقائياً إلى قاعدة عامة أو memory دائمة.

اختبار حاسم. controlled budgets + repeated runs. نحتفظ baseline أبسط ونقيس cost وlatency وtokens وrecovery إلى جانب النتيجة. أي gain يختفي عند تغيير model أو task family يسجل كتخصص domain-specific، لا كدليل عمومية. وعند variance بين runs نحفظ التوزيع بدلاً من اختيار أفضل trajectory.

الدمج في MOUSSAIFXT. evidence rank يرتفع بعد runs محفوظة. العقد الأعلى ثابت: Mission Contract → Active Difference → Context/Capability Routing → Bounded Execution → Observation → Evidence → Remaining Gap → Closure/Recovery. هذا الخط يمنع feature جديدة من بناء loop موازية غير مرئية للحاكمية أو سجل الأدلة.

Promotion contract

كل lesson/policy/skill جديد يبدأ Candidate مع test مناسب وrollback pointer.

Evidence contract

كل claim تشغيلي يحتاج artifact/trace/state readback قابل لإعادة الفحص.

المراجع والعقد الختامي للبحث

PRIMARY SOURCES · INTERNAL CORPUS · FINAL RESEARCH CONTRACT

هذه القائمة تركز على المصادر التي غيرت معمارية Part III، لا على exhaustive bibliography. تواريخ/تفاصيل المشاريع سريعة التغير؛ عند implementation يجب pin commit SHA وlicense في upstreams.lock.json.

[E01] PrimeIntellect-ai/prime-agent — https://github.com/PrimeIntellect-ai/prime-agent
RLM, persistent IPython, subagents, daemon sessions, continual harness.

[E02] Prime Agent Continual Harness implementation — https://github.com/PrimeIntellect-ai/prime-agent/blob/main/packages/coding-agent/src/core/refinement/refinement.ts
Immutable base + editable persistent notes/memory/skills/subagents.

[E03] Chen et al. - AVO: Agentic Variation Operators for Autonomous Evolutionary Search (2026) — https://arxiv.org/abs/2603.24517
Agentic variation, lineage, execution feedback, long-horizon optimization.

[E05] NVIDIA OpenShell — https://github.com/NVIDIA/OpenShell
Sandboxed runtime, declarative filesystem/network/process/inference policies.

[E06] NVIDIA NeMo Agent Toolkit — https://github.com/NVIDIA/NeMo-Agent-Toolkit
Instrumentation, evaluation, trajectory/tool middleware.

[E07] Pydantic AI Harness — https://github.com/pydantic/pydantic-ai-harness
Capability packages: tools, hooks, instructions, model settings.

[E08] NousResearch/hermes-agent — https://github.com/NousResearch/hermes-agent
Persistent memory, skills, subagents, MCP, cron, provider independence.

[E09] NousResearch/hermes-agent-self-evolution — https://github.com/NousResearch/hermes-agent-self-evolution
DSPy/GEPA trace-driven skill/prompt evolution with gates.

[E10] OpenAI Codex — https://github.com/openai/codex
Core/app-server/CLI; sandbox and typed approval flows.

[E11] Anthropic Claude Code docs — https://docs.anthropic.com/en/docs/claude-code
Sessions, subagents, hooks, MCP, permissions, structured CLI.

[E12] OpenCode / Cline / Goose / Gemini CLI — https://github.com/anomalyco/opencode ; https://github.com/cline/cline ; https://github.com/block/goose ; https://github.com/google-gemini/gemini-cli
Portable CLI/headless/provider/protocol patterns.

[E13] Yang et al. - SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering — https://arxiv.org/abs/2405.15793
ACI design as a causal performance factor.

[E14] Aider Repository Map — https://aider.chat/docs/repomap.html
Graph-aware structural code context under token budget.

[E15] OpenHands / Deep Agents — https://github.com/All-Hands-AI/OpenHands ; https://github.com/langchain-ai/deepagents
Workspace/runtime abstraction, durable agent harness patterns.

[P01] Sumers et al. - Cognitive Architectures for Language Agents (CoALA) — https://arxiv.org/abs/2309.02427
Modular memory, structured action space, decision process.

[P02] Packer et al. - MemGPT: Towards LLMs as Operating Systems — https://arxiv.org/abs/2310.08560
Virtual context management and hierarchical memory.

[P03] Zhang et al. - Agentic Context Engineering — https://arxiv.org/abs/2510.04618
Evolving playbooks; generation/reflection/curation; anti-collapse.

[P04] Shinn et al. - Reflexion — https://arxiv.org/abs/2303.11366
Verbal feedback stored as episodic memory.

[P05] Park et al. - Generative Agents — https://arxiv.org/abs/2304.03442
Memory, reflection, planning architecture.

[P06] Wang et al. - Voyager — https://arxiv.org/abs/2305.16291
Automatic curriculum + executable skill library + feedback.

[P07] Microsoft Research - Agent Lightning — https://www.microsoft.com/en-us/research/project/agent-lightning/
Decouple agent execution from RL/optimization; trajectory interface.

[P08] Yamada et al. - The AI Scientist-v2 — https://arxiv.org/abs/2504.08066
Progressive agentic tree search for scientific discovery.

[P09] Schmidgall et al. - Agent Laboratory — https://arxiv.org/abs/2501.04227
Research workflow with human feedback.

[B01] Mialon et al. - GAIA — https://arxiv.org/abs/2311.12983
General assistants: tools/web/multimodal reasoning.

[B02] Liu et al. - AgentBench — https://arxiv.org/abs/2308.03688
Eight interactive environments and long-horizon failures.

[B03] OSWorld — https://os-world.github.io/
Real computer environment; execution-based evaluation; OSWorld 2.0 notice.

[B04] WebArena — https://webarena.dev/
Self-hosted realistic web environment with programmatic validators.

[B05] Xu et al. - TheAgentCompany — https://arxiv.org/abs/2412.14161
Workplace-like web/code/communication tasks.

المصادر الداخلية المستخدمة كقيود تصميم

MOUSSAIF AGI OS / MOUSSAIXT Deep Engineering Architecture, 50p. [M02] MOUSSAIF Unified AGI, 225p, claim-ranked E0-E5. [M03] [M01] AGI-Oriented Knowledge-Base as Cognitive OS + ParallaxWorld-MCD / Fattah-AGI corpus. تستخدم هنا لتثبيت vocabulary/invariants، لا كدليل خارجي على AGI.

العقد الختامي للجزء الثالث

لا يصبح AL RAZZI «ثورة» لأن هذه النظريات كُتبت، يصبح تقدماً حين: (1) يبني world effects حقيقية، (2) يستأنف بعد crash، (3) يعمل فوق context أكبر من model عبر MCAS، (4) يبدل harness/model دون state loss، (5) يتعلم candidate قابلاً للrollback، و(6) يثبت gain على held-out tasks مع انخفاض false-success. عندها فقط نرفع evidence rank من architecture إلى experiment.