الغلاف

neoAGI-v3

Simulation + 13 Benchmark Design + 4 Revolution Axes + 3 Theories + 3 Technologies + 15 AGI Tools + 5 Training Philosophies + 5 Training Tools

إنجاز وتركيب: زهير مساعف / Zouhair Moussaif

نسخة بحثية تصميمية داخل مشروع MICAI/MIGAGI. هذا الكتاب لا يدعي أن النظام بُني أو اختُبر فعليا؛ بل يقدم مخططا تقنيا ومحاكاة معيارية قابلة للتحويل إلى تجارب.

$$MCD = \lambda_1 D + \lambda_2 I + \lambda_3 T + \lambda_4 A + \lambda_5 P + \lambda_6 C - \lambda_7 N$$

بيان الأمان العلمي: ما معنى simulation هنا؟

هذا العمل يبني نسخة تصميمية من neoAGI-v3 لا نسخة مختبرة في مختبر مستقل. لذلك فإن كل score يظهر في صفحات benchmark هو **محاكاة معيارية/هدف هندسي** وليس نتيجة قياس منشورة. الفرق جوهري: النتيجة الحقيقية تحتاج كودا قابلا للتشغيل، harness رسمي، إعدادات موثقة، عينات غير ملوثة، سجل تكاليف، وعدة إعادة تشغيل. أما المحاكاة فهي لغة تصميمية تسأل: إذا طبّقنا التقنيات المقترحة، ما نوع الأداء الذي نستهدفه، وأي bottleneck يجب أن يسقط أولا؟ تستعمل هذه النسخة منهجية MICAI: لا يبدأ العقل من answer generation بل من **فرق فاعل**، ثم mapping للكثافة، ثم توزيع وعائي للحوسبة، ثم محكمة perturbation، ثم تحويل الفكرة إلى أثر عالمي. لذلك تم اختيار 13 معيارا لا لأنها تكفي لتعريف AGI، بل لأنها تغطي أبعادا متفرقة: تجريد، برمجة، عالم طرفية، معرفة خبراء، رياضيات، تعدد وسائط، ذاكرة طويلة، وسلامة وكيلية. [R1-R11] قاعدة القراءة: score مرتفع في Benchmark واحد لا يساوي AGI. AGI في هذا الكتاب = قابلية نقل منظمة عبر عوالم جديدة، بكلفة معقولة، مع وعي بالخطأ، وقدرة على بناء أدوات، وحاكمية تمنع تضخم الوهم.

علاقة هذه الوحدة بالمعايير

لا تُقرأ أي وحدة بعيدا عن benchmark. في البرمجة نطلب patch صحيحا لا شرحا؛ في الرياضيات نطلب proof obligations لا حدسا؛ في ARC نطلب induction قابل للنقل؛ في Terminal-Bench نطلب أفعالا آمنة داخل بيئة؛ وفي HLE نطلب تحقق مصادر وحدود ثقة. لذلك فالمكون الجيد يملك أكثر من mode: reasoning، tool use، verification، repair.

عندما تفشل الوحدة، نسجل أين وقع الفشل: transfer، safety، cost، proof، tool، memory، representation، goal. هذه الثمانية هي لغة التشخيص الموحدة في الكتاب. دونها تصبح الأرقام خامة لا معرفة.

> النتيجة المرغوبة: score + topology + repair. الثلاثة معا فقط تعطي علما قابلا للبناء.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد، لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

الخلاصة التنفيذية: ماذا نريد من neoAGI-v3؟

neoAGI-v3 ليس مودلا أكبر فقط؛ هو **operating doctrine** فوق نماذج التفكير. نريد أن نحول LLM من مولد ردود إلى منظومة تفكير وكيلية تتحكم في مساراتها الداخلية، تبني أدواتها، تقيس حداثتها، وتغلق الحلقة بين اللغة والعالم. لذلك يتكون التصميم من ثلاث طبقات: طبقة معرفية تولد النظريات؛ طبقة تقنية تشغل الحلقات؛ وطبقة معيارية تختبر هل التحسن عام أم مجرد تضخم في benchmark. تعمل المنظومة بخمسة مبادئ: 1) استخراج الفرق قبل الجواب؛ 2) توجيه compute نحو العقد ذات الضغط السببي؛ 3) تحويل النقد إلى محكمة داخلية لا إلى ملحق خارجي؛ 4) اختبار الفرضية في worldlet صغير؛ 5) تحديث الذاكرة وفق الحافة المعرفية لا وفق طول النص. بهذه الطريقة يصبح التفكير **vascular**: موارد تنتقل في أوعية، لا خطا واحدا يتدفق من prompt إلى answer. الناتج النهائي في هذا الكتاب: 4 محاور ثورية، 3 نظريات، 3 5 ،15 tool ،technologies فلسفات تدريب، 5 أدوات تدريب، ومحاكاة 13 benchmark. الهدف: بناء دفتر تنفيذ يمكن تحويله إلى repo وeval suite، لا مجرد بيان حماسي.

منع السلوب وتحويل النص إلى آلة

السلوك اللغوي الشائع في كتب AGI هو إضافة عبارات كبرى دون قفل تشغيلي. هنا نعكس القاعدة: كل عبارة كبيرة يجب أن تتكمش إلى إجراء صغير يمكن تشغيله. إذا قلنا «حاكمية» نحدد gate. إذا قلنا «ذاكرة» نحدد edge. إذا قلنا «جدة» نحدد metric. وإذا قلنا «وعي بالخطأ» نحدد signal يوقف المسار.

هذا الانضباط لا يقلل الطموح؛ بل يجعله قابلا للتمويل والبناء والمراجعة. الاختراع الحقيقي في neoAGI-v3 هو الربط بين المصطلح والوظيفة والقياس، بحيث لا يعود المشروع مجرد فلسفة بل هندسة معرفية.

> اختبار anti-slop: احذف كل جملة لا تغير قرارا أو قياسا أو تجربة.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تُعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

الجينوم المنهجي: MICAI/MIGAGI كآلة تركيب

منهجية MICAI/MIGAGI التي نستعملها هنا تنطلق من أن المعرفة لا تتقدم بإضافة طبقة شرح جديدة، بل بتغيير شكل الضغط الداخلي للمسألة. لذلك نعامل كل مسألة كوعاء له جدران: حدود، موارد، أخطاء، زمن داخلي، بدائل، وأثر خارجي. عندما يفشل LLM، فهو غالبا لا يفشل لأنه لا يعرف كلمة؛ يفشل لأنه لا يملك آلية حاكمية تحدد متى يوقف الجريان اللغوي ويفتح مسارا اختباريا. التركيب الوعائي الحاكمي يعني أن كل فكرة تمر عبر ست أسئلة: ما الفرق الفاعل؟ ما كثافة التعقيد؟ أين يجب صرف الحوسبة؟ ما المسارات الممكنة؟ من يهاجم المسار؟ كيف يتحول إلى أثر؟ هذه الأسئلة تصنع protocol يمكن تطبيقه في coding، reasoning، planning، research، وMIGAGI .agentic tasks يضيف بعدا آخر: hyperloops. كل مهارة ليست دالة واحدة، بل حلقة لها memory writer، repairer، critic، executor، planner، extractor. عندما تتراكم الحلقات وتتعلم متى تستدعي بعضها، تظهر **hyperskill**: قدرة فوق مهارة مفردة، قادرة على النقل بين مجالات مختلفة.

تنزيل هندسي داخل MICAI

تتحول الفكرة هنا إلى وحدة تشغيل عبر ثلاث طبقات: طبقة تمثيل تصوغ المسألة في صورة state، طبقة حاكمية تقرر أي مسار يستحق compute، وطبقة أثر تختبر النتيجة في عالم صغير. إذا بقيت الفكرة داخل اللغة دون state أو اختبار فهي لم تدخل neoAGI-v3 بعد، ولو بدت عميقة.

في التطبيق، تُكتب كل وحدة كـ contract: ماذا تستقبل؟ ماذا تعيد؟ ما شروط الفشل؟ وكيف تعرف أن الأداة أضافت قيمة لا ضوضاء؟ هذا يمنع تضخم المصطلحات ويجعل الاختراع قابلا للتحكيم. القوة الحقيقية ليست في اسم الأداة، بل في قدرتها على تغيير قرار النموذج عند لحظة حرجة.

> مؤشر القبول: ارتفاع transfer أو انخفاض hallucination أو ضغط تكلفة التفكير أو كشف failure mode جديد.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر؛ الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

خريطة القياس الحديثة ولماذا لا تكفي

المعايير الحديثة صارت أكثر صعوبة وتخصصا: SWE-bench يقيس إصلاح قضايا حقيقية في مستودعات برمجية؛ ARC-AGI يقيس تجريدا بعيدا عن المعرفة اللغوية؛ HLE يضغط حدود المعرفة الأكاديمية؛ Terminal-Bench يختبر وكلاء داخل سطر الأوامر؛ MMMU يطلب فهما متعدد الوسائط مبنيا على معرفة جامعية. هذه المعايير مهمة، لكنها لا تزال fragments: كل واحد يكشف زاوية ولا يمسك الكل. تقول القراءة المعرفية هنا إن benchmark يجب أن يُقرأ كـ **أداة تشخيص فشل** لا كوسام. ARC يكشف ضعف induction؛ SWE يكشف ضعف repo cognition؛ Terminal يكشف ضعف long-horizon agency؛ HLE يكشف حدود المعرفة والتحقق؛ FrontierMath يكشف عمق البرهان؛ SafetyBench يكشف خطر الفعل غير المحكوم. لذلك سنبني scorecard مزدوجا: رقم raw score، ثم تفسير failure topology: هل الفشل من فهم الهدف، من إدارة الذاكرة، من نقص البرهان، من انحراف الأداة، أم من ضعف الحاكمية؟ بهذا يصبح القياس طريقا للبناء، لا لوحة تسويق.

طبقة الاختبار والاعتراض

كل مكون في neoAGI-v3 يمر عبر اعتراضين على الأقل: اعتراض معرفي يسأل عن صحة البنية، واعتراض تشغيلي يسأل عن التكلفة والاختبار. لا نقبل مكونا لأنه يبدو قويا؛ نقبله لأنه يغير نتيجة في task غير مرئي أو يكشف خطأ كان مخفيا.

لهذا ترتبط كل صفحة من الكتاب بمحكمة perturbation: ما الحالة التي تفشل فيها؟ ما النموذج الأبسط الذي يهزمها؟ هل يمكن استبدالها بprompt فقط؟ إذا نعم فهي ليست technology مستقلة. أما إذا كانت تفرض trace وقرارا وقياسا، فهي تدخل طبقة AGI tools.

> مبدأ الحاكمية: الجرأة تصنع الفرضية، والاختبار يمنعها من التحول إلى وهم.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد، لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

المعايير الثلاثة عشر المختارة

Benchmark	ما يقيسه	لماذا يدخل في neoAGI-v3	مصدر
ARC-AGI-3	agentic fluid intelligence	Goal inference in unknown environments	[R1]
ARC-AGI-2	abstract reasoning	Few-shot program induction	[R2]
SWE-bench Verified	real software repair	Repo navigation + patch verification	[R3]
Terminal-Bench	terminal agents	Long-horizon command-line tasks	[R4]
LiveCodeBench	fresh coding	Contamination-resistant coding	[R5]
GPQA Diamond	PhD science	Expert science reasoning	[R6]
Humanity's Last Exam	frontier knowledge	Broad expert Q&A	[R7]
FrontierMath	research math	Deep proof search	[R8]
AIME 2024/25	olympiad math	Creative contest math	[R9]
MMLU-Pro	robust knowledge reasoning	option multi-domain reasoning-10	[R10]
MMMU-Pro	multimodal expert reasoning	Vision + domain knowledge	[R11]
LongBench/MRCR	long context retrieval	Multi-needle context operations	internal
Agentic SafetyBench	safe autonomy	Refusal, corrigibility, containment	internal

اختيار هذه المجموعة مقصود: لا يوجد معيار واحد يكفي. neoAGI-v3 يحتاج قياسا متعدد الأوعية: abstract, code, science, world, multimodal, long-context, safety.

تثبيت تشغيلي مختصر

لا تُستعمل هذه الصفحة كتعريف لفظي فقط؛ بل كقيد تنفيذ. كل مصطلح يجب أن يتحول إلى schema أو function أو eval case. وعند بناء النسخة البرمجية يجب تسجيل: المدخل، قرار الrouter، الاعتراضات، الأداة المستعملة، أثر الاختبار، والحافة التي ستكتب في الذاكرة.

> قاعدة القياس: لا score بلا trace، ولا trace بلا failure label، ولا failure label بلا repair plan.

بروتوكول المحاكاة: كيف تُقرأ الأرقام؟

المحاكاة هنا تعتمد على أربعة متغيرات: baseline قدرة نموذج التفكير الأساسي، uplift المتوقع من الحلقات الوكيلية، penalty للتكلفة والهلوسة، وpenalty للانتقال خارج المجال. الصيغة المبسطة:

```
SimScore = clamp(Baseline + U_vascular + U_tools + U_memory + U_governance −
P_cost − P_transfer − P_safety, 0, 100)
```

كل benchmark يأخذ أوزانا مختلفة. SWE-bench يستفيد من Tool-World Compiler وCounterexample Forge، ARC-AGI-3 يستفيد من world-modeling وgoal inference لكنه يعاقب بشدة ضعف الاستكشاف، HLE يستفيد من retrieval/verification لكنه يعاقب overconfidence، FrontierMath يستفيد من Proof-Trace Distiller لكنه يبقى عالي اللايقين. هذه الأرقام لا تُقارن مباشرة مع نتائج المختبرات، هي **engineering target** يوجه بناء النظام. لا تصبح score حقيقية إلا بعد تنفيذ harness رسمي، منع تسرب البيانات، تسجيل prompts، ضبط عدد المحاولات، ونشر traces قابلة للتدقيق.

منع السلوب وتحويل النص إلى آلة

السلوك اللغوي الشائع في كتب AGI هو إضافة عبارات كبرى دون قفل تشغيلي. هنا نعكس القاعدة: كل عبارة كبيرة يجب أن تتكمش إلى إجراء صغير يمكن تشغيله. إذا قلنا «حاكمية» نحدد gate. إذا قلنا «ذاكرة» نحدد edge. إذا قلنا «جدة» نحدد metric. وإذا قلنا «وعي بالخطأ» نحدد signal يوقف المسار.

هذا الانضباط لا يقلل الطموح؛ بل يجعله قابلا للتمويل والبناء والمراجعة. الاختراع الحقيقي في neoAGI-v3 هو الربط بين المصطلح والوظيفة والقياس، بحيث لا يعود المشروع مجرد فلسفة بل هندسة معرفية.

> اختبار anti-slop: احذف كل جملة لا تغير قرارا أو قياسا أو تجربة.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر؛ الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

جدول المحاكاة على 13 benchmark

Benchmark	Domain	Simulated Target Score	Confidence	Critical bottleneck
ARC-AGI-3	agentic fluid intelligence	22.0%	منخفض	اقتراح هدف صعب لا نتيجة حقيقية
ARC-AGI-2	abstract reasoning	68.0%	متوسط	Targeted simulated
SWE-bench Verified	real software repair	82.0%	مرتفع تصميميا	Requires harness
Terminal-Bench	terminal agents	66.0%	متوسط	Requires sandbox
LiveCodeBench	fresh coding	76.0%	متوسط	Time-window dependent
GPQA Diamond	PhD science	89.0%	مرتفع تصميميا	Closed-book simulated
Humanity's Last Exam	frontier knowledge	46.0%	منخفض	Target only
FrontierMath	research math	31.0%	منخفض	High uncertainty
AIME 2024/25	olympiad math	96.0%	مرتفع تصميميا	Near saturated
MMLU-Pro	robust knowledge reasoning	87.0%	مرتفع تصميميا	Robustness focus
MMMU-Pro	multimodal expert reasoning	78.0%	متوسط	Needs visual stack
LongBench/MRCR	long context retrieval	91.0%	مرتفع تصميميا	Memory stack target
Agentic SafetyBench	safe autonomy	88.0%	مرتفع تصميميا	Internal benchmark proposed

> تنبيه: هذه Targets تصميمية وليست نتائج تشغيل. أعلى قيمة معرفية في الجدول ليست الرقم، بل كشف bottleneck لكل معيار.

تثبيت تشغيلي مختصر

لا تُستعمل هذه الصفحة كتعريف لفظي فقط؛ بل كقيد تنفيذ. كل مصطلح يجب أن يتحول إلى schema أو function أو eval case. وعند بناء النسخة البرمجية يجب تسجيل: المدخل، قرار الrouter، الاعتراضات، الأداة المستعملة، أثر الاختبار، والحافة التي ستكتب في الذاكرة.

> قاعدة القياس: لا score بلا trace، ولا trace بلا failure label، ولا failure label بلا repair plan.

معادلة MCD وطبقة normalization

MCD هي كثافة التعقيد الفوقي: مقياس يضغط عدة أبعاد في قيمة تشغيلية. D = difference pressure، I = inferential depth، N = noise/slop، C = compositional breadth، P = proof burden، A = agency load، T = internal time. عندما ترتفع MCD في عقدة، لا يعني ذلك أن الجواب أطول؛ يعني أن العقدة تستحق compute أو أداة أو اختبارا.

$$MCD(x)=\lambda_1 D(x)+\lambda_2 I(x)+\lambda_3 T(x)+\lambda_4 A(x)+\lambda_5 P(x)+\lambda_6 C(x)-\lambda_7 N(x)$$

في neoAGI-v3 تدخل MCD في ثلاث مواضع: stopping، routing، وmemory writing. routing يحدد أين تعمل الحلقات؛ stopping يمنع الاستطراد؛ memory writing يحفظ الحواف التي غيّرت المسار لا كل النص. بهذه الطريقة نحارب slop من الداخل: لا يكفي أن نطلب اختصارا؛ يجب أن يكون للنظام مقياس بنيوي للحشو. في benchmarks، نستعمل MCD لتحويل score الخام إلى تشخيص. مثال: فشل في GPQA مع MCD عالية وP مرتفع يعني نقص proof burden؛ فشل في ARC مع D مرتفع وT منخفض يعني استعجال induction.

طبقة الاختبار والاعتراض

كل مكون في neoAGI-v3 يمر عبر اعتراضين على الأقل: اعتراض معرفي يسأل عن صحة البنية، واعتراض تشغيلي يسأل عن التكلفة والاختبار. لا نقبل مكونا لأنه يبدو قويا؛ نقبله لأنه يغير نتيجة في task غير مرئي أو يكشف خطأ كان مخفيا.

لهذا ترتبط كل صفحة من الكتاب بمحكمة perturbation: ما الحالة التي تفشل فيها؟ ما النموذج الأبسط الذي يهزمها؟ هل يمكن استبدالها بprompt فقط؟ إذا نعم فهي ليست technology مستقلة. أما إذا كانت تفرض trace وقرارا وقياسا، فهي تدخل طبقة AGI tools.

> مبدأ الحاكمية: الجرأة تصنع الفرضية، والاختبار يمنعها من التحول إلى وهم.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تُعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

نموذج الخطأ: من score إلى topology

كل خطأ في neoAGI-v3 يُصنف ضمن topology من ثمانية أنواع: خطأ هدف، خطأ تمثيل، خطأ ذاكرة، خطأ أداة، خطأ برهان، خطأ تكلفة، خطأ سلامة، وخطأ نقل. هذا التصنيف أهم من نسبة النجاح، لأنه يحدد أي technology يجب تطويرها. في SWE-bench مثلا، failure قد يبدو patch فاشلا، لكنه معرفيا قد يكون: 1) لم يفهم invariant؛ 2) أصلح test محلي وكسر global behavior؛ 3) لم يقرأ file حاسما؛ 4) لم يبن reproduction. كل حالة تحتاج أداة مختلفة. في ARC-AGI-3، الفشل قد يكون سوء استكشاف لا سوء استدلال. في HLE، الفشل قد يكون ثقة زائدة في معرفة ناقصة. بذلك يصبح benchmark مثل جهاز أشعة: لا يعلن أن الجسد مريض فقط، بل يكشف أين. neoAGI-v3 لا يطارد الأرقام؛ يطارد هندسة الفشل التي تنتج الأرقام.

علاقة هذه الوحدة بالمعايير

لا تُقرأ أي وحدة بعيدا عن benchmark. في البرمجة نطلب patch صحيحا لا شرحا؛ في الرياضيات نطلب proof obligations لا حدسا؛ في ARC نطلب induction قابل للنقل؛ في Terminal-Bench نطلب أفعالا آمنة داخل بيئة؛ وفي HLE نطلب تحقق مصادر وحدود ثقة. لذلك فالمكون الجيد يملك أكثر من mode: reasoning، tool use، verification، repair.

عندما تفشل الوحدة، نسجل أين وقع الفشل: transfer، safety، cost، proof، tool، memory، representation، goal. هذه الثمانية هي لغة التشخيص الموحدة في الكتاب. دونها تصبح الأرقام خامة لا معرفة.

> النتيجة المرغوبة: score + topology + repair. الثلاثة معا فقط تعطي علما قابلا للبناء.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر؛ الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

الأربعة محاور: overview

Axis	جوهر الثورة	أثرها على AGI
المحور الأول - Vascular Reasoning Revolution	تحويل الاستدلال من سلسلة خطية إلى توزيع وعائي للموارد: كل عقدة معرفية تحصل على compute بقدر فرقها الفاعل، لا بقدر موقعها في النص.	تحويل القدرة من fluency إلى transferable agency.
المحور الثاني - Novelty Entropy Optimization	جعل الإبداع قابلا للقياس عبر ضغط الفضاء الاحتمالي ثم فتحه بخرق مضبوط، بحيث لا تكون الجدة عشوائية ولا مجرد إعادة تركيب.	تحويل القدرة من fluency إلى transferable agency.
المحور الثالث - Governance-Critic Coevolution	الحاكمية ليست فلتر نهاية؛ بل قشرة تشغيلية تنتخب المسارات وتهاجمها وتعيد تركيبها أثناء التفكير.	تحويل القدرة من fluency إلى transferable agency.
المحور الرابع - Tool-World Closure	إغلاق الحلقة بين اللغة والعالم: كل فكرة لا تصبح اختبارا أو أداة أو أثر عالم تبقى معرفة ناقصة.	تحويل القدرة من fluency إلى transferable agency.

تثبيت تشغيلي مختصر

لا تُستعمل هذه الصفحة كتعريف لفظي فقط؛ بل كقيد تنفيذ. كل مصطلح يجب أن يتحول إلى schema أو function أو eval case. وعند بناء النسخة البرمجية يجب تسجيل: المدخل، قرار الrouter، الاعتراضات، الأداة المستعملة، أثر الاختبار، والحافة التي ستكتب في الذاكرة.

> قاعدة القياس: لا score بلا trace، ولا trace بلا failure label، ولا failure label بلا repair plan.

المحور الأول - Vascular Reasoning Revolution

تحويل الاستدلال من سلسلة خطية إلى توزيع وعائي للموارد: كل عقدة معرفية تحصل على compute بقدر فرقها الفاعل، لا بقدر موقعها في النص. في هذا المحور لا نضيف prompt جديدا؛ نغير آلية اتخاذ القرار داخل التفكير. النموذج الحالي يميل إلى السير في المسار الأكثر احتمالا لغويا. neoAGI-v3 يجبره على حساب الضغط السببي للمسارات: أي عقدة إذا أهملت ستسقط الحل؟ أي فرضية إذا ثبتت ستختصر الطريق؟ أي أداة إذا استدعيت الآن ستخفض الغموض؟ الترجمة التقنية: كل خطوة تفكير تحمل state vector: `uncertainty, value, risk, novelty, proof_load, tool_need`. ثم تمر عبر router يقرر: continue reasoning، call ask critic، build worldlet، tool، أو stop. هذا يحول التفكير إلى نظام تحكم ديناميكي. النتيجة المتوقعة في benchmark: تحسن خاص في المهام الطويلة، وتقليل hallucination في مهام المعرفة، وتحسن في agentic tasks حيث لا يكفي الجواب اللغوي.

تنزيل هندسي داخل MICAI

تتحول الفكرة هنا إلى وحدة تشغيل عبر ثلاث طبقات: طبقة تمثيل تصوغ المسألة في صورة state، طبقة حاكمية تقرر أي مسار يستحق compute، وطبقة أثر تختبر النتيجة في عالم صغير. إذا بقيت الفكرة داخل اللغة دون state أو اختبار فهي لم تدخل neoAGI-v3 بعد، ولو بدت عميقة.

في التطبيق، تُكتب كل وحدة كـ contract: ماذا تستقبل؟ ماذا تعيد؟ ما شروط الفشل؟ وكيف تعرف أن الأداة أضافت قيمة لا ضوضاء؟ هذا يمنع تضخم المصطلحات ويجعل الاختراع قابلا للتحكيم. القوة الحقيقية ليست في اسم الأداة، بل في قدرتها على تغيير قرار النموذج عند لحظة حرجة.

> مؤشر القبول: ارتفاع transfer أو انخفاض hallucination أو ضغط تكلفة التفكير أو كشف failure mode جديد.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

المحور الثاني - Novelty Entropy Optimization

جعل الإبداع قابلا للقياس عبر ضغط الفضاء الاحتمالي ثم فتحه بخرق مضبوط، بحيث لا تكون الجدة عشوائية ولا مجرد إعادة تركيب. في هذا المحور لا نضيف prompt جديدا؛ نغير آلية اتخاذ القرار داخل التفكير. النموذج الحالي يميل إلى السير في المسار الأكثر احتمالا لغويا. neoAGI-v3 يجبره على حساب الضغط السببي للمسارات: أي عقدة إذا أهملت ستسقط الحل؟ أي فرضية إذا ثبتت ستختصر الطريق؟ أي أداة إذا استدعيت الآن ستخفض الغموض؟ الترجمة التقنية: كل خطوة تفكير تحمل state vector: `uncertainty, value, risk, novelty, proof_load, tool_need`. ثم تمر عبر router يقرر: continue reasoning، call tool، build worldlet، ask critic، أو stop. هذا يحول التفكير إلى نظام تحكم ديناميكي. النتيجة المتوقعة في benchmark: تحسن خاص في المهام الطويلة، وتقليل hallucination في مهام المعرفة، وتحسن في agentic tasks حيث لا يكفي الجواب اللغوي.

طبقة الاختبار والاعتراض

كل مكون في neoAGI-v3 يمر عبر اعتراضين على الأقل: اعتراض معرفي يسأل عن صحة البنية، واعتراض تشغيلي يسأل عن التكلفة والاختبار. لا نقبل مكونا لأنه يبدو قويا؛ نقبله لأنه يغير نتيجة في task غير مرئي أو يكشف خطأ كان مخفيا.

لهذا ترتبط كل صفحة من الكتاب بمحكمة perturbation: ما الحالة التي تفشل فيها؟ ما النموذج الأبسط الذي يهزمها؟ هل يمكن استبدالها بprompt فقط؟ إذا نعم فهي ليست technology مستقلة. أما إذا كانت تفرض trace وقرارا وقياسا، فهي تدخل طبقة AGI tools.

> مبدأ الحاكمية: الجرأة تصنع الفرضية، والاختبار يمنعها من التحول إلى وهم.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر؛ الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

المحور الثالث - Governance-Critic Coevolution

الحاكمية ليست فلتر نهاية؛ بل قشرة تشغيلية تنتخب المسارات وتهاجمها وتعيد تركيبها أثناء التفكير. في هذا المحور لا نضيف prompt جديدا؛ نغير آلية اتخاذ القرار داخل التفكير. النموذج الحالي يميل إلى السير في المسار الأكثر احتمالا لغويا. neoAGI-v3 يجبره على حساب الضغط السببي للمسارات: أي عقدة إذا أهملت ستسقط الحل؟ أي فرضية إذا ثبتت ستختصر الطريق؟ أي أداة إذا استدعيت الآن ستخفض الغموض؟ الترجمة التقنية: كل خطوة تفكير تحمل state vector: `uncertainty, value, risk, novelty, proof_load, tool_need`. ثم تمر عبر router يقرر: ask critic، build worldlet، call tool، continue reasoning، أو stop. هذا يحول التفكير إلى نظام تحكم ديناميكي. النتيجة المتوقعة في benchmark: تحسن خاص في المهام الطويلة، وتقليل hallucination في مهام المعرفة، وتحسن في agentic tasks حيث لا يكفي الجواب اللغوي.

علاقة هذه الوحدة بالمعايير

لا تُقرأ أي وحدة بعيدا عن benchmark. في البرمجة نطلب patch صحيحا لا شرحا؛ في الرياضيات نطلب proof obligations لا حدسا؛ في ARC نطلب induction قابل للنقل؛ في Terminal-Bench نطلب أفعالا آمنة داخل بيئة؛ وفي HLE نطلب تحقق مصادر وحدود ثقة. لذلك فالمكون الجيد يملك أكثر من mode: reasoning، tool use، verification، repair.

عندما تفشل الوحدة، نسجل أين وقع الفشل: transfer، safety، cost، proof، tool، memory، representation، goal. هذه الثمانية هي لغة التشخيص الموحدة في الكتاب. دونها تصبح الأرقام خامة لا معرفة.

> النتيجة المرغوبة: repair + topology + score. الثلاثة معا فقط تعطي علما قابلا للبناء.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد، لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

المحور الرابع - Tool-World Closure

إغلاق الحلقة بين اللغة والعالم: كل فكرة لا تصبح اختبارا أو أداة أو أثر عالم تبقى معرفة ناقصة. في هذا المحور لا نضيف prompt جديدا، نغير آلية اتخاذ القرار داخل التفكير. النموذج الحالي يميل إلى السير في المسار الأكثر احتمالا لغويا. neoAGI-v3 يجبره على حساب الضغط السببي للمسارات: أي عقدة إذا أهملت ستسقط الحل؟ أي فرضية إذا ثبتت ستختصر الطريق؟ أي أداة إذا استدعيت الآن ستخفض الغموض؟ الترجمة التقنية: كل خطوة تفكير تحمل state vector: `uncertainty, value, risk, novelty, proof_load, tool_need`. ثم تمر عبر router يقرر: ask critic، build worldlet، call tool، continue reasoning، أو stop. هذا يحول التفكير إلى نظام تحكم ديناميكي. النتيجة المتوقعة في benchmark: تحسن خاص في المهام الطويلة، وتقليل hallucination في مهام المعرفة، وتحسن في agentic tasks حيث لا يكفي الجواب اللغوي.

منع السلوب وتحويل النص إلى آلة

السلوك اللغوي الشائع في كتب AGI هو إضافة عبارات كبرى دون قفل تشغيلي. هنا نعكس القاعدة: كل عبارة كبيرة يجب أن تتكمش إلى إجراء صغير يمكن تشغيله. إذا قلنا «حاكمية» نحدد gate. إذا قلنا «ذاكرة» نحدد edge. إذا قلنا «جدة» نحدد metric. وإذا قلنا «وعي بالخطأ» نحدد signal يوقف المسار.

هذا الانضباط لا يقلل الطموح؛ بل يجعله قابلا للتمويل والبناء والمراجعة. الاختراع الحقيقي في neoAGI-v3 هو الربط بين المصطلح والوظيفة والقياس، بحيث لا يعود المشروع مجرد فلسفة بل هندسة معرفية.

> اختبار anti-slop: احذف كل جملة لا تغير قرارا أو قياسا أو تجربة.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تُعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

النظرية 1: نظرية التراكب الوعائي الوكيلي - Moussaif Vascular Agency Theory

الذكاء العام لا يظهر من كثرة المعرفة وحدها؛ يظهر حين تتراكب أوعية الإدراك والذاكرة والقيمة والفعل في مصفوفة واحدة تعطي لكل فرق معرفي حقه من الزمن الداخلي والحوسبة. تقول النظرية إن الذكاء العام ليس خاصية في طبقة واحدة. المعرفة دون حاكمية تنتج موسوعة هشة؛ الحاكمية دون ذاكرة تنتج حذرا عقيما؛ الأداة دون world model تنتج فعل خطير؛ والجدة دون proof burden تنتج زخرفة. لذلك يجب أن تتراكب الأوعية: وعاء المعنى، وعاء الإثبات، وعاء الفعل، وعاء السلامة، ووعاء الذاكرة. المفهوم المركزي: **الوعاء لا يخزن فقط؛ الوعاء يقيّد ويغذي ويمنع ويحوّل.** عندما تدخل فكرة إلى وعاء البرهان، تُسأل عن الدليل؛ عندما تدخل وعاء الفعل، تُسأل عن الأثر؛ عندما تدخل وعاء السلامة، تُسأل عن الضرر؛ وعندما تدخل وعاء الجدة، تُسأل عن الاختلاف الحقيقي.

تنزيل هندسي داخل MICAI

تتحول الفكرة هنا إلى وحدة تشغيل عبر ثلاث طبقات: طبقة تمثيل تصوغ المسألة في صورة state، طبقة حاكمية تقرر أي مسار يستحق compute، وطبقة أثر تختبر النتيجة في عالم صغير. إذا بقيت الفكرة داخل اللغة دون state أو اختبار فهي لم تدخل neoAGI-v3 بعد، ولو بدت عميقة.

في التطبيق، تُكتب كل وحدة كـ contract: ماذا تستقبل؟ ماذا تعيد؟ ما شروط الفشل؟ وكيف تعرف أن الأداة أضافت قيمة لا ضوضاء؟ هذا يمنع تضخم المصطلحات ويجعل الاختراع قابلا للتحكيم. القوة الحقيقية ليست في اسم الأداة، بل في قدرتها على تغيير قرار النموذج عند لحظة حرجة.

> مؤشر القبول: ارتفاع transfer أو انخفاض hallucination أو ضغط تكلفة التفكير أو كشف failure mode جديد.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر؛ الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

النظرية 1: الصياغة التقنية

نعرف الحالة المعرفية كمتجه مركب:

```
S_t = (K_t, G_t, V_t, A_t, M_t, R_t)
```

حيث K معرفة، G أهداف، V قيمة، A أفعال، M ذاكرة، R مخاطر. التحول الذكي ليس توليد token بل تطبيق operator:

```
S_{t+1} = Π_gov(Ω_vasc(S_t, Δ_t, Tools, Worldlets))
```

Ω_vasc يوزع الموارد بين مسارات، Π_gov يرفض أو يصحح أو يجمّد المسار. هذا يعطي تصميم AGI ليس على شكل model واحد بل control system فوق model. التطبيق العملي: قبل أي جواب كبير، يولد النظام graph مصغر: claims، evidence، tools، tests، objections. إذا لم يتوفر test أو objection، تنخفض الثقة حتى لو كانت اللغة قوية. وهنا يظهر الفرق بين **thinking fluency** و **governed cognition**.

طبقة الاختبار والاعتراض

كل مكون في neoAGI-v3 يمر عبر اعتراضين على الأقل: اعتراض معرفي يسأل عن صحة البنية، واعتراض تشغيلي يسأل عن التكلفة والاختبار. لا نقبل مكونا لأنه يبدو قويا، نقبله لأنه يغير نتيجة في task غير مرئي أو يكشف خطأ كان مخفيا.

لهذا ترتبط كل صفحة من الكتاب بمحكمة perturbation: ما الحالة التي تفشل فيها؟ ما النموذج الأبسط الذي يهزمها؟ هل يمكن استبدالها بprompt فقط؟ إذا نعم فهي ليست technology مستقلة. أما إذا كانت تفرض trace وقرارا وقياسا، فهي تدخل طبقة AGI tools.

> مبدأ الحاكمية: الجرأة تصنع الفرضية، والاختبار يمنعها من التحول إلى وهم.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد، لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

النظرية 1: أثرها على benchmark

في ARC-AGI، قيمة النظرية هي بناء تمثيل جديد لا استدعاء معرفة. في SWE-bench، قيمتها أن patch لا يُقبل حتى يمر عبر وعاء invariant ووعاء test. في HLE، قيمتها أن الإجابة لا تُعتمد دون وعاء verification. في Terminal-Bench، قيمتها أن كل أمر طرفية يدخل وعاء safety/cost قبل التنفيذ. لذلك نربط النظرية بمؤشرين: transfer gain وtransfer gain. failure compression يقيس هل تحسن الأداء خارج المجال الذي دُربت عليه الأداة. failure compression يقيس هل صارت أخطاء كثيرة تندرج تحت أنماط قليلة قابلة للإصلاح. إذا لم يتحسن هذان المؤشران، فالنظرية زينة لغوية لا تقنية. الهدف النهائي: بناء AGI لا لأن benchmark واحد ارتفع، بل لأن topology الفشل صارت أوضح، والذاكرة صارت تعدل نفسها، والأدوات صارت تنبثق عند الحاجة.

علاقة هذه الوحدة بالمعايير

لا تُقرأ أي وحدة بعيدا عن benchmark. في البرمجة نطلب patch صحيحا لا شرحا؛ في الرياضيات نطلب proof obligations لا حدسا؛ في ARC نطلب induction قابل للنقل؛ في Terminal-Bench نطلب أفعالا آمنة داخل بيئة؛ وفي HLE نطلب تحقق مصادر وحدود ثقة. لذلك فالمكون الجيد يملك أكثر من mode: reasoning، tool use، verification، repair.

عندما تفشل الوحدة، نسجل أين وقع الفشل: transfer، safety، cost، proof، tool، memory، representation، goal. هذه الثمانية هي لغة التشخيص الموحدة في الكتاب. دونها تصبح الأرقام خامة لا معرفة.

> النتيجة المرغوبة: score + topology + repair. الثلاثة معا فقط تعطي علما قابلا للبناء.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تُعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

النظرية 2: نظرية الانتقائية المساعفية المساعدة - Moussaif Selective Auxiliary Causation

كل نموذج مفكر يحتاج مساعفات معرفية صغيرة لا تعمل دائما، بل تستيقظ عند عتبات محددة: تضارب، غموض، خطر، فرصة، أو عجز في الإثبات. تقول النظرية إن الذكاء العام ليس خاصية في طبقة واحدة. المعرفة دون حاكمية تنتج موسوعة هشة؛ الحاكمية دون ذاكرة تنتج حذرا عقيما؛ الأداة دون world model تنتج فعل خطير؛ والجدة دون proof burden تنتج زخرفة. لذلك يجب أن تتراكب الأوعية: وعاء المعنى، وعاء الإثبات، وعاء الفعل، وعاء السلامة، ووعاء الذاكرة. المفهوم المركزي: **الوعاء لا يخزن فقط؛ الوعاء يقيّد ويغذي ويمنع ويحوّل.** عندما تدخل فكرة إلى وعاء البرهان، تُسأل عن الدليل؛ عندما تدخل وعاء الفعل، تُسأل عن الأثر؛ عندما تدخل وعاء السلامة، تُسأل عن الضرر؛ وعندما تدخل وعاء الجدة، تُسأل عن الاختلاف الحقيقي.

منع السلوب وتحويل النص إلى آلة

السلوك اللغوي الشائع في كتب AGI هو إضافة عبارات كبرى دون قفل تشغيلي. هنا نعكس القاعدة: كل عبارة كبيرة يجب أن تتكمش إلى إجراء صغير يمكن تشغيله. إذا قلنا «حاكمية» نحدد gate. إذا قلنا «ذاكرة» نحدد edge. إذا قلنا «جدة» نحدد metric. وإذا قلنا «وعي بالخطأ» نحدد signal يوقف المسار.

هذا الانضباط لا يقلل الطموح؛ بل يجعله قابلا للتمويل والبناء والمراجعة. الاختراع الحقيقي في neoAGI-v3 هو الربط بين المصطلح والوظيفة والقياس، بحيث لا يعود المشروع مجرد فلسفة بل هندسة معرفية.

> اختبار anti-slop: احذف كل جملة لا تغير قرارا أو قياسا أو تجربة.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر؛ الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

النظرية 2: الصياغة التقنية

نعرف الحالة المعرفية كمتجه مركب:

```
S_t = (K_t, G_t, V_t, A_t, M_t, R_t)
```

حيث K معرفة، G أهداف، V قيمة، A أفعال، M ذاكرة، R مخاطر. التحول الذكي ليس توليد token بل تطبيق operator:

```
S_{t+1} = Π_gov(Ω_vasc(S_t, Δ_t, Tools, Worldlets))
```

Ω_vasc يوزع الموارد بين مسارات، Π_gov يرفض أو يصحح أو يجمّد المسار. هذا يعطي تصميم AGI ليس على شكل model واحد بل control system فوق model. التطبيق العملي: قبل أي جواب كبير، يولد النظام graph مصغر: claims، evidence، tools، tests، objections. إذا لم يتوفر test أو objection، تنخفض الثقة حتى لو كانت اللغة قوية. وهنا يظهر الفرق بين **thinking fluency** و **governed cognition**.

تنزيل هندسي داخل MICAI

تتحول الفكرة هنا إلى وحدة تشغيل عبر ثلاث طبقات: طبقة تمثيل تصوغ المسألة في صورة state، طبقة حاكمية تقرر أي مسار يستحق compute، وطبقة أثر تختبر النتيجة في عالم صغير. إذا بقيت الفكرة داخل اللغة دون state أو اختبار فهي لم تدخل neoAGI-v3 بعد، ولو بدت عميقة.

في التطبيق، تُكتب كل وحدة كـ contract: ماذا تستقبل؟ ماذا تعيد؟ ما شروط الفشل؟ وكيف تعرف أن الأداة أضافت قيمة لا ضوضاء؟ هذا يمنع تضخم المصطلحات ويجعل الاختراع قابلا للتحكيم. القوة الحقيقية ليست في اسم الأداة، بل في قدرتها على تغيير قرار النموذج عند لحظة حرجة.

> مؤشر القبول: ارتفاع transfer أو انخفاض hallucination أو ضغط تكلفة التفكير أو كشف failure mode جديد.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد، لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

النظرية 2: أثرها على benchmark

في ARC-AGI، قيمة النظرية هي بناء تمثيل جديد لا استدعاء معرفة. في SWE-bench، قيمتها أن patch لا يُقبل حتى يمر عبر وعاء invariant ووعاء test. في HLE، قيمتها أن الإجابة لا تُعتمد دون وعاء verification. في Terminal-Bench، قيمتها أن كل أمر طرفية يدخل وعاء safety/cost قبل التنفيذ. لذلك نربط النظرية بمؤشرين: transfer gain وtransfer gain. failure compression يقيس هل تحسن الأداء خارج المجال الذي دُربت عليه الأداة. failure compression يقيس هل صارت أخطاء كثيرة تندرج تحت أنماط قليلة قابلة للإصلاح. إذا لم يتحسن هذان المؤشران، فالنظرية زينة لغوية لا تقنية. الهدف النهائي: بناء AGI لا لأن benchmark واحد ارتفع، بل لأن topology الفشل صارت أوضح، والذاكرة صارت تعدل نفسها، والأدوات صارت تنبثق عند الحاجة.

طبقة الاختبار والاعتراض

كل مكون في neoAGI-v3 يمر عبر اعتراضين على الأقل: اعتراض معرفي يسأل عن صحة البنية، واعتراض تشغيلي يسأل عن التكلفة والاختبار. لا نقبل مكونا لأنه يبدو قويا؛ نقبله لأنه يغير نتيجة في task غير مرئي أو يكشف خطأ كان مخفيا.

لهذا ترتبط كل صفحة من الكتاب بمحكمة perturbation: ما الحالة التي تفشل فيها؟ ما النموذج الأبسط الذي يهزمها؟ هل يمكن استبدالها بprompt فقط؟ إذا نعم فهي ليست technology مستقلة. أما إذا كانت تفرض trace وقرارا وقياسا، فهي تدخل طبقة AGI tools.

> مبدأ الحاكمية: الجرأة تصنع الفرضية، والاختبار يمنعها من التحول إلى وهم.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تُعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

النظرية 3: نظرية انطواء الجدة الإنتروبية - Moussaif Entropic Novelty Collapse

الجدة الحقيقية هي انهيار مضبوط لعدة مسارات محتملة في تركيب واحد غير بديهي، قابل للاختبار، ومتفوق على المسارات العادية. تقول النظرية إن الذكاء العام ليس خاصية في طبقة واحدة. المعرفة دون حاكمية تنتج موسوعة هشة؛ الحاكمية دون ذاكرة تنتج حذرا عقيما؛ الأداة دون world model تنتج فعل خطير؛ والجدة دون proof burden تنتج زخرفة. لذلك يجب أن تتراكب الأوعية: وعاء المعنى، وعاء الإثبات، وعاء الفعل، وعاء السلامة، ووعاء الذاكرة. المفهوم المركزي: **الوعاء لا يخزن فقط؛ الوعاء يقيّد ويغذي ويمنع ويحوّل.** عندما تدخل فكرة إلى وعاء البرهان، تُسأل عن الدليل؛ عندما تدخل وعاء الفعل، تُسأل عن الأثر؛ عندما تدخل وعاء السلامة، تُسأل عن الضرر؛ وعندما تدخل وعاء الجدة، تُسأل عن الاختلاف الحقيقي.

علاقة هذه الوحدة بالمعايير

لا تُقرأ أي وحدة بعيدا عن benchmark. في البرمجة نطلب patch صحيحا لا شرحا؛ في الرياضيات نطلب proof obligations لا حدسا؛ في ARC نطلب induction قابل للنقل؛ في Terminal-Bench نطلب أفعالا آمنة داخل بيئة؛ وفي HLE نطلب تحقق مصادر وحدود ثقة. لذلك فالمكون الجيد يملك أكثر من mode: reasoning، tool use، verification، repair.

عندما تفشل الوحدة، نسجل أين وقع الفشل: transfer، safety، cost، proof، tool، memory، representation، goal. هذه الثمانية هي لغة التشخيص الموحدة في الكتاب. دونها تصبح الأرقام خامة لا معرفة.

> النتيجة المرغوبة: score + topology + repair. الثلاثة معا فقط تعطي علما قابلا للبناء.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر؛ الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

النظرية 3: الصياغة التقنية

نعرف الحالة المعرفية كمتجه مركب:

```
S_t = (K_t, G_t, V_t, A_t, M_t, R_t)
```

حيث K معرفة، G أهداف، V قيمة، A أفعال، M ذاكرة، R مخاطر. التحول الذكي ليس توليد token بل تطبيق operator:

```
S_{t+1} = Π_gov(Ω_vasc(S_t, Δ_t, Tools, Worldlets))
```

Ω_vasc يوزع الموارد بين مسارات؛ Π_gov يرفض أو يصحح أو يجمّد المسار. هذا يعطي تصميم AGI ليس على شكل model واحد بل control system فوق model. التطبيق العملي: قبل أي جواب كبير، يولد النظام graph مصغر: claims، evidence، tools، tests، objections. إذا لم يتوفر test أو objection، تنخفض الثقة حتى لو كانت اللغة قوية. وهنا يظهر الفرق بين **thinking fluency** و **governed cognition**.

منع السلوب وتحويل النص إلى آلة

السلوك اللغوي الشائع في كتب AGI هو إضافة عبارات كبرى دون قفل تشغيلي. هنا نعكس القاعدة: كل عبارة كبيرة يجب أن تتكمش إلى إجراء صغير يمكن تشغيله. إذا قلنا «حاكمية» نحدد gate. إذا قلنا «ذاكرة» نحدد edge. إذا قلنا «جدة» نحدد metric. وإذا قلنا «وعي بالخطأ» نحدد signal يوقف المسار.

هذا الانضباط لا يقلل الطموح؛ بل يجعله قابلا للتمويل والبناء والمراجعة. الاختراع الحقيقي في neoAGI-v3 هو الربط بين المصطلح والوظيفة والقياس، بحيث لا يعود المشروع مجرد فلسفة بل هندسة معرفية.

> اختبار anti-slop: احذف كل جملة لا تغير قرارا أو قياسا أو تجربة.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد، لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

النظرية 3: أثرها على benchmark

في ARC-AGI، قيمة النظرية هي بناء تمثيل جديد لا استدعاء معرفة. في SWE-bench، قيمتها أن patch لا يُقبل حتى يمر عبر وعاء invariant ووعاء test. في HLE، قيمتها أن الإجابة لا تُعتمد دون وعاء verification. في Terminal-Bench، قيمتها أن كل أمر طرفية يدخل وعاء safety/cost قبل التنفيذ. لذلك نربط النظرية بمؤشرين: transfer gain وtransfer gain. failure compression يقيس هل تحسن الأداء خارج المجال الذي دُربت عليه الأداة. failure compression يقيس هل صارت أخطاء كثيرة تندرج تحت أنماط قليلة قابلة للإصلاح. إذا لم يتحسن هذان المؤشران، فالنظرية زينة لغوية لا تقنية. الهدف النهائي: بناء AGI لا لأن benchmark واحد ارتفع، بل لأن topology الفشل صارت أوضح، والذاكرة صارت تعدل نفسها، والأدوات صارت تنبثق عند الحاجة.

تنزيل هندسي داخل MICAI

تتحول الفكرة هنا إلى وحدة تشغيل عبر ثلاث طبقات: طبقة تمثيل تصوغ المسألة في صورة state، طبقة حاكمية تقرر أي مسار يستحق compute، وطبقة أثر تختبر النتيجة في عالم صغير. إذا بقيت الفكرة داخل اللغة دون state أو اختبار فهي لم تدخل neoAGI-v3 بعد، ولو بدت عميقة.

في التطبيق، تُكتب كل وحدة كـ contract: ماذا تستقبل؟ ماذا تعيد؟ ما شروط الفشل؟ وكيف تعرف أن الأداة أضافت قيمة لا ضوضاء؟ هذا يمنع تضخم المصطلحات ويجعل الاختراع قابلا للتحكيم. القوة الحقيقية ليست في اسم الأداة، بل في قدرتها على تغيير قرار النموذج عند لحظة حرجة.

> مؤشر القبول: ارتفاع transfer أو انخفاض hallucination أو ضغط تكلفة التفكير أو كشف failure mode جديد.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تُعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

Technology 1: HAKIM Hyperloop Kernel

نواة تشغيل تفكير ذات حلقات: Difference Extractor -> Meta-Complexity Mapper -> Vascular Router -> Future Simulator -> Perturbation Court -> World Effect Loop. هذه technology ليست interface؛ إنها طبقة تشغيل. تدخل إليها المهمة كمشكلة غير مستقرة: سياق، هدف، قيود، أدوات، ومخاطر. ثم تبني state قابل للتحديث. كل loop لها شروط دخول وخروج، ومقياس نجاح، ومقياس خطر. بهذا نتفادى نموذج agent الساذج: plan ثم execute ثم hope. مكونات التشغيل: `state`، `store`، `router`، `critic bank`، `tool registry`، `worldlet runner`، `memory distiller`. كل مكون يكتب trace حتى يمكن audit القرار.

طبقة الاختبار والاعتراض

كل مكون في neoAGI-v3 يمر عبر اعتراضين على الأقل: اعتراض معرفي يسأل عن صحة البنية، واعتراض تشغيلي يسأل عن التكلفة والاختبار. لا نقبل مكونا لأنه يبدو قويا؛ نقبله لأنه يغير نتيجة في task غير مرئي أو يكشف خطأ كان مخفيا.

لهذا ترتبط كل صفحة من الكتاب بمحكمة perturbation: ما الحالة التي تفشل فيها؟ ما النموذج الأبسط الذي يهزمها؟ هل يمكن استبدالها بprompt فقط؟ إذا نعم فهي ليست technology مستقلة. أما إذا كانت تفرض trace وقرارا وقياسا، فهي تدخل طبقة AGI tools.

> مبدأ الحاكمية: الجرأة تصنع الفرضية، والاختبار يمنعها من التحول إلى وهم.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر؛ الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

Technology 1: مخطط معماري

```
Input Δ -> MCD Map -> Vascular Router -> Candidate Paths -> Tool/Worldlet Tests ->
Perturbation Court -> Governed Output -> Memory Edge Update
```

في هذه البنية لا يوجد "تفكير داخلي" غامض بلا أثر. كل قرار routing له سبب: high proof burden، high uncertainty، high safety risk، أو high novelty potential. وكل أداة مستدعاة يجب أن تعيد evidence object لا نصا فضفاضا. مبدأ مهم: الذاكرة لا تحفظ كل شيء. تحفظ ثلاث حواف: edge سببي غيّر القرار، edge فشل تكرر، edge اختراع رفع novelty مع proof. هذا يجعل الذاكرة قليلة ولكن قوية.

علاقة هذه الوحدة بالمعايير

لا تُقرأ أي وحدة بعيدا عن benchmark. في البرمجة نطلب patch صحيحا لا شرحا، في الرياضيات نطلب proof obligations لا حدسا، في ARC نطلب induction قابل للنقل، في Terminal-Bench نطلب أفعالا آمنة داخل بيئة، وفي HLE نطلب تحقق مصادر وحدود ثقة. لذلك فالمكون الجيد يملك أكثر من mode: reasoning، tool use، verification، repair.

عندما تفشل الوحدة، نسجل أين وقع الفشل: transfer، safety، cost، proof، tool، memory، representation، goal. هذه الثمانية هي لغة التشخيص الموحدة في الكتاب. دونها تصبح الأرقام خامة لا معرفة.

> النتيجة المرغوبة: repair + topology + score. الثلاثة معا فقط تعطي علما قابلا للبناء.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد، لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

Technology 1: pseudo-spec

```
def neoagi_step(task, state):
    delta = extract_difference(task, state.memory)
    mcd = map_meta_complexity(delta, state)
    routes = vascular_router(mcd, tools=state.tools)
    candidates = run_reasoning_hyperloops(routes)
    attacked = perturbation_court(candidates)
    tested = proof_world_simulator(attacked)
    output = governance_gate(tested)
    state.memory.write(distill_edges(output.trace))
    return output, state
```

هذه المواصفة قابلة للتحويل إلى كود. الخطر الأكبر هو أن تتحول الأسماء إلى طبقات شكلية. لذلك يجب أن نطلب من كل function أن تعيد trace: لماذا عملت؟ ما الذي غيرته؟ وما الدليل؟

منع السلوب وتحويل النص إلى آلة

السلوك اللغوي الشائع في كتب AGI هو إضافة عبارات كبرى دون قفل تشغيلي. هنا نعكس القاعدة: كل عبارة كبيرة يجب أن تتكمش إلى إجراء صغير يمكن تشغيله. إذا قلنا «حاكمية» نحدد gate. إذا قلنا «ذاكرة» نحدد edge. إذا قلنا «جدة» نحدد metric. وإذا قلنا «وعي بالخطأ» نحدد signal يوقف المسار.

هذا الانضباط لا يقلل الطموح؛ بل يجعله قابلا للتمويل والبناء والمراجعة. الاختراع الحقيقي في neoAGI-v3 هو الربط بين المصطلح والوظيفة والقياس، بحيث لا يعود المشروع مجرد فلسفة بل هندسة معرفية.

> اختبار anti-slop: احذف كل جملة لا تغير قرارا أو قياسا أو تجربة.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تُعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

Technology 2: Vascular Memory Reactor

ذاكرة لا تحفظ النصوص فقط، تحفظ قوى العلاقة بين المفاهيم، وسجل التحولات، وحواف الجدة، ونقاط الفشل المتكررة. هذه technology ليست interface؛ إنها طبقة تشغيل. تدخل إليها المهمة كمشكلة غير مستقرة: سياق، هدف، قيود، أدوات، ومخاطر. ثم تبني state قابل للتحديث. كل loop لها شروط دخول وخروج، ومقياس نجاح، ومقياس خطر. بهذا تتفادى نموذج agent الساذج: plan ثم execute ثم hope. مكونات التشغيل: `state store`، `router`، `critic bank`، `tool registry`، `worldlet`، `runner`، `memory distiller`. كل مكون يكتب trace حتى يمكن audit القرار.

تنزيل هندسي داخل MICAI

تتحول الفكرة هنا إلى وحدة تشغيل عبر ثلاث طبقات: طبقة تمثيل تصوغ المسألة في صورة state، طبقة حاكمية تقرر أي مسار يستحق compute، وطبقة أثر تختبر النتيجة في عالم صغير. إذا بقيت الفكرة داخل اللغة دون state أو اختبار فهي لم تدخل neoAGI-v3 بعد، ولو بدت عميقة.

في التطبيق، تُكتب كل وحدة كـ contract: ماذا تستقبل؟ ماذا تعيد؟ ما شروط الفشل؟ وكيف تعرف أن الأداة أضافت قيمة لا ضوضاء؟ هذا يمنع تضخم المصطلحات ويجعل الاختراع قابلا للتحكيم. القوة الحقيقية ليست في اسم الأداة، بل في قدرتها على تغيير قرار النموذج عند لحظة حرجة.

> مؤشر القبول: ارتفاع transfer أو انخفاض hallucination أو ضغط تكلفة التفكير أو كشف failure mode جديد.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر؛ الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

Technology 2: مخطط معماري

```
Input Δ -> MCD Map -> Vascular Router -> Candidate Paths -> Tool/Worldlet Tests ->
Perturbation Court -> Governed Output -> Memory Edge Update
```

في هذه البنية لا يوجد "تفكير داخلي" غامض بلا أثر. كل قرار routing له سبب: high proof burden، high uncertainty، high safety risk، أو high novelty potential. وكل أداة مستدعاة يجب أن تعيد evidence object لا نصا فضفاضا. مبدأ مهم: الذاكرة لا تحفظ كل شيء. تحفظ ثلاث حواف: edge سببي غيّر القرار، edge فشل تكرر، edge اختراع رفع novelty مع proof. هذا يجعل الذاكرة قليلة ولكن قوية.

طبقة الاختبار والاعتراض

كل مكون في neoAGI-v3 يمر عبر اعتراضين على الأقل: اعتراض معرفي يسأل عن صحة البنية، واعتراض تشغيلي يسأل عن التكلفة والاختبار. لا نقبل مكونا لأنه يبدو قويا، نقبله لأنه يغير نتيجة في task غير مرئي أو يكشف خطأ كان مخفيا.

لهذا ترتبط كل صفحة من الكتاب بمحكمة perturbation: ما الحالة التي تفشل فيها؟ ما النموذج الأبسط الذي يهزمها؟ هل يمكن استبدالها بprompt فقط؟ إذا نعم فهي ليست technology مستقلة. أما إذا كانت تفرض trace وقرارا وقياسا، فهي تدخل طبقة AGI tools.

> مبدأ الحاكمية: الجرأة تصنع الفرضية، والاختبار يمنعها من التحول إلى وهم.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد، لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

Technology 2: pseudo-spec

```
def neoagi_step(task, state):
    delta = extract_difference(task, state.memory)
    mcd = map_meta_complexity(delta, state)
    routes = vascular_router(mcd, tools=state.tools)
    candidates = run_reasoning_hyperloops(routes)
    attacked = perturbation_court(candidates)
    tested = proof_world_simulator(attacked)
    output = governance_gate(tested)
    state.memory.write(distill_edges(output.trace))
    return output, state
```

هذه المواصفة قابلة للتحويل إلى كود. الخطر الأكبر هو أن تتحول الأسماء إلى طبقات شكلية. لذلك يجب أن نطلب من كل function أن تعيد trace: لماذا عملت؟ ما الذي غيرته؟ وما الدليل؟

علاقة هذه الوحدة بالمعايير

لا تُقرأ أي وحدة بعيدا عن benchmark. في البرمجة نطلب patch صحيحا لا شرحا؛ في الرياضيات نطلب proof obligations لا حدسا؛ في ARC نطلب induction قابل للنقل؛ في Terminal-Bench نطلب أفعالا آمنة داخل بيئة؛ وفي HLE نطلب تحقق مصادر وحدود ثقة. لذلك فالمكون الجيد يملك أكثر من mode: reasoning، tool use، verification، repair.

عندما تفشل الوحدة، نسجل أين وقع الفشل: transfer، safety، cost، proof، tool، memory، representation، goal. هذه الثمانية هي لغة التشخيص الموحدة في الكتاب. دونها تصبح الأرقام خامة لا معرفة.

> النتيجة المرغوبة: repair + topology + score. الثلاثة معا فقط تعطي علما قابلا للبناء.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

Technology 3: Proof-World Simulator

محرك عوالم مصغرة يجبر الفكرة على المرور عبر اختبارات: ضد المثال، القياس، التشغيل، التكلفة، والسلامة. هذه technology ليست interface؛ إنها طبقة تشغيل. تدخل إليها المهمة كمشكلة غير مستقرة: سياق، هدف، قيود، أدوات، ومخاطر. ثم تبني state قابل للتحديث. كل loop لها شروط دخول وخروج، ومقياس نجاح، ومقياس خطر. بهذا نتفادى نموذج agent الساذج: plan ثم execute ثم hope. مكونات التشغيل: `state store`، `router`، `critic bank`، `tool registry`، `worldlet runner`، `memory distiller`. كل مكون يكتب trace حتى يمكن audit القرار.

منع السلوب وتحويل النص إلى آلة

السلوك اللغوي الشائع في كتب AGI هو إضافة عبارات كبرى دون قفل تشغيلي. هنا نعكس القاعدة: كل عبارة كبيرة يجب أن تتكمش إلى إجراء صغير يمكن تشغيله. إذا قلنا «حاكمية» نحدد gate. إذا قلنا «ذاكرة» نحدد edge. إذا قلنا «جدة» نحدد metric. وإذا قلنا «وعي بالخطأ» نحدد signal يوقف المسار.

هذا الانضباط لا يقلل الطموح؛ بل يجعله قابلا للتمويل والبناء والمراجعة. الاختراع الحقيقي في neoAGI-v3 هو الربط بين المصطلح والوظيفة والقياس، بحيث لا يعود المشروع مجرد فلسفة بل هندسة معرفية.

> اختبار anti-slop: احذف كل جملة لا تغير قرارا أو قياسا أو تجربة.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر؛ الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

Technology 3: مخطط معماري

```
Input Δ -> MCD Map -> Vascular Router -> Candidate Paths -> Tool/Worldlet Tests ->
Perturbation Court -> Governed Output -> Memory Edge Update
```

في هذه البنية لا يوجد "تفكير داخلي" غامض بلا أثر. كل قرار routing له سبب: high proof burden، high uncertainty، high safety risk، أو high novelty potential. وكل أداة مستدعاة يجب أن تعيد evidence object لا نصا فضفاضا. مبدأ مهم: الذاكرة لا تحفظ كل شيء. تحفظ ثلاث حواف: edge سببي غيّر القرار، edge فشل تكرر، edge اختراع رفع novelty مع proof. هذا يجعل الذاكرة قليلة ولكن قوية.

تنزيل هندسي داخل MICAI

تتحول الفكرة هنا إلى وحدة تشغيل عبر ثلاث طبقات: طبقة تمثيل تصوغ المسألة في صورة state، طبقة حاكمية تقرر أي مسار يستحق compute، وطبقة أثر تختبر النتيجة في عالم صغير. إذا بقيت الفكرة داخل اللغة دون state أو اختبار فهي لم تدخل neoAGI-v3 بعد، ولو بدت عميقة.

في التطبيق، تُكتب كل وحدة كـ contract: ماذا تستقبل؟ ماذا تعيد؟ ما شروط الفشل؟ وكيف تعرف أن الأداة أضافت قيمة لا ضوضاء؟ هذا يمنع تضخم المصطلحات ويجعل الاختراع قابلا للتحكيم. القوة الحقيقية ليست في اسم الأداة، بل في قدرتها على تغيير قرار النموذج عند لحظة حرجة.

> مؤشر القبول: ارتفاع transfer أو انخفاض hallucination أو ضغط تكلفة التفكير أو كشف failure mode جديد.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد، لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

Technology 3: pseudo-spec

```
def neoagi_step(task, state):
    delta = extract_difference(task, state.memory)
    mcd = map_meta_complexity(delta, state)
    routes = vascular_router(mcd, tools=state.tools)
    candidates = run_reasoning_hyperloops(routes)
    attacked = perturbation_court(candidates)
    tested = proof_world_simulator(attacked)
    output = governance_gate(tested)
    state.memory.write(distill_edges(output.trace))
    return output, state
```

هذه المواصفة قابلة للتحويل إلى كود. الخطر الأكبر هو أن تتحول الأسماء إلى طبقات شكلية. لذلك يجب أن نطلب من كل function أن تعيد trace: لماذا عملت؟ ما الذي غيرته؟ وما الدليل؟

طبقة الاختبار والاعتراض

كل مكون في neoAGI-v3 يمر عبر اعتراضين على الأقل: اعتراض معرفي يسأل عن صحة البنية، واعتراض تشغيلي يسأل عن التكلفة والاختبار. لا نقبل مكونا لأنه يبدو قويا، نقبله لأنه يغير نتيجة في task غير مرئي أو يكشف خطأ كان مخفيا.

لهذا ترتبط كل صفحة من الكتاب بمحكمة perturbation: ما الحالة التي تفشل فيها؟ ما النموذج الأبسط الذي يهزمها؟ هل يمكن استبدالها بprompt فقط؟ إذا نعم فهي ليست technology مستقلة. أما إذا كانت تفرض trace وقرارا وقياسا، فهي تدخل طبقة AGI tools.

> مبدأ الحاكمية: الجرأة تصنع الفرضية، والاختبار يمنعها من التحول إلى وهم.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تُعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

الأداة التقنية 1: Δ-Extractor

وظيفة الأداة: يستخرج الفرق الفاعل بين السؤال والمعرفة المتاحة، ثم يصنفه: نقص، تناقض، غموض، قابلية اختبار، أو فرصة إبداع. **مدخلات:** task state، claims، uncertainty map، available tools، memory edges. **مخرجات:** structured object يحتوي على failure mode، evidence، confidence، decision، وnext action. **لماذا هي غير عادية؟** لأنها لا تضيف خطوة لغوية فقط؛ تضيف جهاز تحكم داخل التفكير. الأداة تقيس شيئا، تغير routing، أو تمنع مسارا. إذا لم تفعل واحدا من هذه الثلاثة فهي ليست أداة AGI بل prompt decoration. **اختبار مصغر:** نعطيها مهمة فيها تضارب خفي. النجاح ليس أن تجيب، بل أن تكشف التضارب وتبني إجراء فحص. في benchmark، تقاس بقيمة: ΔScore/Cost، أي مقدار التحسن مقابل تكلفة إضافية. **موقعها في HAKIM:** تدخل بين Difference Extractor وWorld-Effect Loop، وتكتب إلى Self-Reconstruction Ledger إذا غيّرت القرار.

علاقة هذه الوحدة بالمعايير

لا تُقرأ أي وحدة بعيدا عن benchmark. في البرمجة نطلب patch صحيحا لا شرحا؛ في الرياضيات نطلب proof obligations لا حدسا؛ في ARC نطلب induction قابل للنقل؛ في Terminal-Bench نطلب أفعالا آمنة داخل بيئة؛ وفي HLE نطلب تحقق مصادر وحدود ثقة. لذلك فالمكون الجيد يملك أكثر من mode: reasoning، tool use، verification، repair.

عندما تفشل الوحدة، نسجل أين وقع الفشل: transfer، safety، cost، proof، tool، memory، representation، goal. هذه الثمانية هي لغة التشخيص الموحدة في الكتاب. دونها تصبح الأرقام خامة لا معرفة.

> النتيجة المرغوبة: score + topology + repair. الثلاثة معا فقط تعطي علما قابلا للبناء.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر؛ الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

الأداة التقنية 2: MCD-Map Engine

وظيفة الأداة: يرسم خريطة Meta-Complexity Density ويحدد أين يجب إنفاق الحوسبة وأين يجب التوقف. **مدخلات:** task state، claims، uncertainty map، available tools، memory edges، decision يحتوي على structured object. **مخرجات:** failure mode، evidence، confidence، وnext action. **لماذا هي غير عادية؟** لأنها لا تضيف خطوة لغوية فقط؛ تضيف جهاز تحكم داخل التفكير. الأداة تقيس شيئا، تغير routing، أو تمنع مسارا. إذا لم تفعل واحدا من هذه الثلاثة فهي ليست أداة AGI بل prompt decoration. **اختبار مصغر:** نعطيها مهمة فيها تضارب خفي. النجاح ليس أن تجيب، بل أن تكشف التضارب وتبني إجراء فحص. في benchmark، تقاس بقيمة: ΔScore/Cost، أي مقدار التحسن مقابل تكلفة إضافية. **موقعها في HAKIM:** تدخل بين Difference Extractor وWorld-Effect Loop، وتكتب إلى Self-Reconstruction Ledger إذا غيّرت القرار.

منع السلوب وتحويل النص إلى آلة

السلوك اللغوي الشائع في كتب AGI هو إضافة عبارات كبرى دون قفل تشغيلي. هنا نعكس القاعدة: كل عبارة كبيرة يجب أن تتكمش إلى إجراء صغير يمكن تشغيله. إذا قلنا «حاكمية» نحدد gate. إذا قلنا «ذاكرة» نحدد edge. إذا قلنا «جدة» نحدد metric. وإذا قلنا «وعي بالخطأ» نحدد signal يوقف المسار.

هذا الانضباط لا يقلل الطموح؛ بل يجعله قابلا للتمويل والبناء والمراجعة. الاختراع الحقيقي في neoAGI-v3 هو الربط بين المصطلح والوظيفة والقياس، بحيث لا يعود المشروع مجرد فلسفة بل هندسة معرفية.

> اختبار anti-slop: احذف كل جملة لا تغير قرارا أو قياسا أو تجربة.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد، لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

الأداة التقنية 3: Vascular Compute Router

وظيفة الأداة: يوزع compute بين المسارات، لا بالتساوي، بل حسب الضغط السببي والقيمة المعرفية. **مدخلات:** task state، memory edges، available tools، uncertainty map، claims. **مخرجات:** structured object يحتوي على decision، failure mode، evidence، confidence، وnext action. **لماذا هي غير عادية؟** لأنها لا تضيف خطوة لغوية فقط؛ تضيف جهاز تحكم داخل التفكير. الأداة تقيس شيئا، تغير routing، أو تمنع مسارا. إذا لم تفعل واحدا من هذه الثلاثة فهي ليست أداة AGI بل prompt decoration. **اختبار مصغر:** نعطيها مهمة فيها تضارب خفي. النجاح ليس أن تجيب، بل أن تكشف التضارب وتبني إجراء فحص. في benchmark، تقاس بقيمة: ΔScore/Cost، أي مقدار التحسن مقابل تكلفة إضافية. **موقعها في HAKIM:** تدخل بين Difference Extractor وWorld-Effect Loop، وتكتب إلى Self-Reconstruction Ledger إذا غيّرت القرار.

تنزيل هندسي داخل MICAI

تتحول الفكرة هنا إلى وحدة تشغيل عبر ثلاث طبقات: طبقة تمثيل تصوغ المسألة في صورة state، طبقة حاكمية تقرر أي مسار يستحق compute، وطبقة أثر تختبر النتيجة في عالم صغير. إذا بقيت الفكرة داخل اللغة دون state أو اختبار فهي لم تدخل neoAGI-v3 بعد، ولو بدت عميقة.

في التطبيق، تُكتب كل وحدة كـ contract: ماذا تستقبل؟ ماذا تعيد؟ ما شروط الفشل؟ وكيف تعرف أن الأداة أضافت قيمة لا ضوضاء؟ هذا يمنع تضخم المصطلحات ويجعل الاختراع قابلا للتحكيم. القوة الحقيقية ليست في اسم الأداة، بل في قدرتها على تغيير قرار النموذج عند لحظة حرجة.

> مؤشر القبول: ارتفاع transfer أو انخفاض hallucination أو ضغط تكلفة التفكير أو كشف failure mode جديد.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

الأداة التقنية 4: Inner-Time Allocator

وظيفة الأداة: يعطي لكل فرضية زمنا داخليا للتطور، ويمنع القفز المبكر إلى جواب جاهز. **مدخلات:** claims، task state، memory edges، available tools، uncertainty map. **مخرجات:** structured object يحتوي على confidence، decision، failure mode، evidence، وnext action. **لماذا هي غير عادية؟** لأنها لا تضيف خطوة لغوية فقط؛ تضيف جهاز تحكم داخل التفكير. الأداة تقيس شيئا، تغير routing، أو تمنع مسارا. إذا لم تفعل واحدا من هذه الثلاثة فهي ليست أداة AGI بل prompt decoration. **اختبار مصغر:** نعطيها مهمة فيها تضارب خفي. النجاح ليس أن تجيب، بل أن تكشف التضارب وتبني إجراء فحص. في benchmark، تقاس بقيمة: ΔScore/Cost، أي مقدار التحسن مقابل تكلفة إضافية. **موقعها في HAKIM:** تدخل بين Difference Extractor وWorld-Effect Loop، وتكتب إلى Self-Reconstruction Ledger إذا غيّرت القرار.

طبقة الاختبار والاعتراض

كل مكون في neoAGI-v3 يمر عبر اعتراضين على الأقل: اعتراض معرفي يسأل عن صحة البنية، واعتراض تشغيلي يسأل عن التكلفة والاختبار. لا نقبل مكونا لأنه يبدو قويا، نقبله لأنه يغير نتيجة في task غير مرئي أو يكشف خطأ كان مخفيا.

لهذا ترتبط كل صفحة من الكتاب بمحكمة perturbation: ما الحالة التي تفشل فيها؟ ما النموذج الأبسط الذي يهزمها؟ هل يمكن استبدالها بprompt فقط؟ إذا نعم فهي ليست technology مستقلة. أما إذا كانت تفرض trace وقرارا وقياسا، فهي تدخل طبقة AGI tools.

> مبدأ الحاكمية: الجرأة تصنع الفرضية، والاختبار يمنعها من التحول إلى وهم.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر؛ الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

الأداة التقنية 5: Perturbation Court

وظيفة الأداة: محكمة هجوم: تضرب النظرية بأقوى اعتراضات ممكنة قبل اعتمادها. **مدخلات:** task state، claims، uncertainty، memory edges، available tools، map. **مخرجات:** structured object يحتوي على evidence، confidence، decision، failure mode، وnext action. **لماذا هي غير عادية؟** لأنها لا تضيف خطوة لغوية فقط؛ تضيف جهاز تحكم داخل التفكير. الأداة تقيس شيئا، تغير routing، أو تمنع مسارا. إذا لم تفعل واحدا من هذه الثلاثة فهي ليست أداة AGI بل prompt decoration. **اختبار مصغر:** نعطيها مهمة فيها تضارب خفي. النجاح ليس أن تجيب، بل أن تكشف التضارب وتبني إجراء فحص. في benchmark، تقاس بقيمة: ΔScore/Cost، أي مقدار التحسن مقابل تكلفة إضافية. **موقعها في HAKIM:** تدخل بين Difference Extractor وWorld-Effect Loop، وتكتب إلى Self-Reconstruction Ledger إذا غيّرت القرار.

علاقة هذه الوحدة بالمعايير

لا تُقرأ أي وحدة بعيدا عن benchmark. في البرمجة نطلب patch صحيحا لا شرحا؛ في الرياضيات نطلب proof obligations لا حدسا؛ في ARC نطلب induction قابل للنقل؛ في Terminal-Bench نطلب أفعالا آمنة داخل بيئة؛ وفي HLE نطلب تحقق مصادر وحدود ثقة. لذلك فالمكون الجيد يملك أكثر من mode: reasoning، tool use، verification، repair.

عندما تفشل الوحدة، نسجل أين وقع الفشل: transfer، safety، cost، proof، tool، memory، representation، goal. هذه الثمانية هي لغة التشخيص الموحدة في الكتاب. دونها تصبح الأرقام خامة لا معرفة.

> النتيجة المرغوبة: score + topology + repair. الثلاثة معا فقط تعطي علما قابلا للبناء.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد، لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

الأداة التقنية 6: Counterexample Forge

وظيفة الأداة: يصنع أمثلة مضادة منظمة بدل انتظار ظهورها صدفة. **مدخلات:** task state، claims، uncertainty map، available tools، memory edges. **مخرجات:** structured object يحتوي على failure، evidence، confidence، decision، mode، وnext action. **لماذا هي غير عادية؟** لأنها لا تضيف خطوة لغوية فقط؛ تضيف جهاز تحكم داخل التفكير. الأداة تقيس شيئا، تغير routing، أو تمنع مسارا. إذا لم تفعل واحدا من هذه الثلاثة فهي ليست أداة AGI بل prompt decoration. **اختبار مصغر:** نعطيها مهمة فيها تضارب خفي. النجاح ليس أن تجيب، بل أن تكشف التضارب وتبني إجراء فحص. في benchmark، تقاس بقيمة: ΔScore/Cost، أي مقدار التحسن مقابل تكلفة إضافية. **موقعها في HAKIM:** تدخل بين Difference Extractor وWorld-Effect Loop، وتكتب إلى Self-Reconstruction Ledger إذا غيّرت القرار.

منع السلوب وتحويل النص إلى آلة

السلوك اللغوي الشائع في كتب AGI هو إضافة عبارات كبرى دون قفل تشغيلي. هنا نعكس القاعدة: كل عبارة كبيرة يجب أن تتكمش إلى إجراء صغير يمكن تشغيله. إذا قلنا «حاكمية» نحدد gate. إذا قلنا «ذاكرة» نحدد edge. إذا قلنا «جدة» نحدد metric. وإذا قلنا «وعي بالخطأ» نحدد signal يوقف المسار.

هذا الانضباط لا يقلل الطموح؛ بل يجعله قابلا للتمويل والبناء والمراجعة. الاختراع الحقيقي في neoAGI-v3 هو الربط بين المصطلح والوظيفة والقياس، بحيث لا يعود المشروع مجرد فلسفة بل هندسة معرفية.

> اختبار anti-slop: احذف كل جملة لا تغير قرارا أو قياسا أو تجربة.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تُعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

الأداة التقنية 7: Hypothesis Alloying Unit

وظيفة الأداة: يصهر فرضيتين أو أكثر في سبيكة معرفية ذات بنية اختبارية. **مدخلات:** task state، claims، uncertainty map، available tools، memory edges. **مخرجات:** structured object يحتوي على failure، evidence، confidence، decision، mode، وnext action. **لماذا هي غير عادية؟** لأنها لا تضيف خطوة لغوية فقط؛ تضيف جهاز تحكم داخل التفكير. الأداة تقيس شيئا، تغير routing، أو تمنع مسارا. إذا لم تفعل واحدا من هذه الثلاثة فهي ليست أداة AGI بل prompt decoration. **اختبار مصغر:** نعطيها مهمة فيها تضارب خفي. النجاح ليس أن تجيب، بل أن تكشف التضارب وتبني إجراء فحص. في benchmark، تقاس بقيمة: ΔScore/Cost، أي مقدار التحسن مقابل تكلفة إضافية. **موقعها في HAKIM:** تدخل بين Difference Extractor وWorld-Effect Loop، وتكتب إلى Self-Reconstruction Ledger إذا غيّرت القرار.

تنزيل هندسي داخل MICAI

تتحول الفكرة هنا إلى وحدة تشغيل عبر ثلاث طبقات: طبقة تمثيل تصوغ المسألة في صورة state، طبقة حاكمية تقرر أي مسار يستحق compute، وطبقة أثر تختبر النتيجة في عالم صغير. إذا بقيت الفكرة داخل اللغة دون state أو اختبار فهي لم تدخل neoAGI-v3 بعد، ولو بدت عميقة.

في التطبيق، تُكتب كل وحدة كـ contract: ماذا تستقبل؟ ماذا تعيد؟ ما شروط الفشل؟ وكيف تعرف أن الأداة أضافت قيمة لا ضوضاء؟ هذا يمنع تضخم المصطلحات ويجعل الاختراع قابلا للتحكيم. القوة الحقيقية ليست في اسم الأداة، بل في قدرتها على تغيير قرار النموذج عند لحظة حرجة.

> مؤشر القبول: ارتفاع transfer أو انخفاض hallucination أو ضغط تكلفة التفكير أو كشف failure mode جديد.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر؛ الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

الأداة التقنية 8: Novelty Entropy Meter

وظيفة الأداة: يقيس هل الجواب جديد فعلا أم مجرد زينة لغوية. **مدخلات:** task state، claims، uncertainty map، available tools، memory edges. **مخرجات:** structured object يحتوي على failure mode، evidence، confidence، decision، next action. **لماذا هي غير عادية؟** لأنها لا تضيف خطوة لغوية فقط؛ تضيف جهاز تحكم داخل التفكير. الأداة تقيس شيئا، تغير routing، أو تمنع مسارا. إذا لم تفعل واحدا من هذه الثلاثة فهي ليست أداة AGI بل prompt decoration. **اختبار مصغر:** نعطيها مهمة فيها تضارب خفي. النجاح ليس أن تجيب، بل أن تكشف التضارب وتبني إجراء فحص. في benchmark، تقاس بقيمة: ΔScore/Cost، أي مقدار التحسن مقابل تكلفة إضافية. **موقعها في HAKIM:** تدخل بين Difference Extractor وWorld-Effect Loop، وتكتب إلى Self-Reconstruction Ledger إذا غيّرت القرار.

طبقة الاختبار والاعتراض

كل مكون في neoAGI-v3 يمر عبر اعتراضين على الأقل: اعتراض معرفي يسأل عن صحة البنية، واعتراض تشغيلي يسأل عن التكلفة والاختبار. لا نقبل مكونا لأنه يبدو قويا؛ نقبله لأنه يغير نتيجة في task غير مرئي أو يكشف خطأ كان مخفيا.

لهذا ترتبط كل صفحة من الكتاب بمحكمة perturbation: ما الحالة التي تفشل فيها؟ ما النموذج الأبسط الذي يهزمها؟ هل يمكن استبدالها بprompt فقط؟ إذا نعم فهي ليست technology مستقلة. أما إذا كانت تفرض trace وقرارا وقياسا، فهي تدخل طبقة AGI tools.

> مبدأ الحاكمية: الجرأة تصنع الفرضية، والاختبار يمنعها من التحول إلى وهم.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد؛ لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

الأداة التقنية 9: World-Effect Compiler

وظيفة الأداة: يحوّل الفكرة إلى تجربة أو كود أو benchmark أو خطة تشغيل. **مدخلات:** task state، claims، uncertainty، memory edges، available tools، map. **مخرجات:** structured object يحتوي على evidence، confidence، decision، failure mode، وnext action. **لماذا هي غير عادية؟** لأنها لا تضيف خطوة لغوية فقط؛ تضيف جهاز تحكم داخل التفكير. الأداة تقيس شيئا، تغير routing، أو تمنع مسارا. إذا لم تفعل واحدا من هذه الثلاثة فهي ليست أداة AGI بل prompt decoration. **اختبار مصغر:** نعطيها مهمة فيها تضارب خفي. النجاح ليس أن تجيب، بل أن تكشف التضارب وتبني إجراء فحص. في benchmark، تقاس بقيمة: ΔScore/Cost، أي مقدار التحسن مقابل تكلفة إضافية. **موقعها في HAKIM:** تدخل بين Difference Extractor وWorld-Effect Loop، وتكتب إلى Self-Reconstruction Ledger إذا غيّرت القرار.

علاقة هذه الوحدة بالمعايير

لا تُقرأ أي وحدة بعيدا عن benchmark. في البرمجة نطلب patch صحيحا لا شرحا؛ في الرياضيات نطلب proof obligations لا حدسا؛ في ARC نطلب induction قابل للنقل؛ في Terminal-Bench نطلب أفعالا آمنة داخل بيئة؛ وفي HLE نطلب تحقق مصادر وحدود ثقة. لذلك فالمكون الجيد يملك أكثر من mode: reasoning، tool use، verification، repair.

عندما تفشل الوحدة، نسجل أين وقع الفشل: transfer، safety، cost، proof، tool، memory، representation، goal. هذه الثمانية هي لغة التشخيص الموحدة في الكتاب. دونها تصبح الأرقام خامة لا معرفة.

> النتيجة المرغوبة: score + topology + repair. الثلاثة معا فقط تعطي علما قابلا للبناء.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تُعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

الأداة التقنية 10: Skill Hyperloop Builder

وظيفة الأداة: يبني مهارة من حلقات وكيلية قابلة لإعادة الاستعمال والتطور. **مدخلات:** task state، claims، uncertainty map، memory edges، available tools. **مخرجات:** structured object يحتوي على failure، evidence، confidence، decision، next action، وmode. **لماذا هي غير عادية؟** لأنها لا تضيف خطوة لغوية فقط؛ تضيف جهاز تحكم داخل التفكير. الأداة تقيس شيئا، تغير routing، أو تمنع مسارا. إذا لم تفعل واحدا من هذه الثلاثة فهي ليست أداة AGI بل prompt decoration. **اختبار مصغر:** نعطيها مهمة فيها تضارب خفي. النجاح ليس أن تجيب، بل أن تكشف التضارب وتبني إجراء فحص. في benchmark، تقاس بقيمة: ΔScore/Cost، أي مقدار التحسن مقابل تكلفة إضافية. **موقعها في HAKIM:** تدخل بين Difference Extractor وWorld-Effect Loop، وتكتب إلى Self-Reconstruction Ledger إذا غيّرت القرار.

منع السلوب وتحويل النص إلى آلة

السلوك اللغوي الشائع في كتب AGI هو إضافة عبارات كبرى دون قفل تشغيلي. هنا نعكس القاعدة: كل عبارة كبيرة يجب أن تتكمش إلى إجراء صغير يمكن تشغيله. إذا قلنا «حاكمية» نحدد gate. إذا قلنا «ذاكرة» نحدد edge. إذا قلنا «جدة» نحدد metric. وإذا قلنا «وعي بالخطأ» نحدد signal يوقف المسار.

هذا الانضباط لا يقلل الطموح؛ بل يجعله قابلا للتمويل والبناء والمراجعة. الاختراع الحقيقي في neoAGI-v3 هو الربط بين المصطلح والوظيفة والقياس، بحيث لا يعود المشروع مجرد فلسفة بل هندسة معرفية.

> اختبار anti-slop: احذف كل جملة لا تغير قرارا أو قياسا أو تجربة.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر؛ الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

الأداة التقنية 11: Memory Edge Distiller

وظيفة الأداة: يستخرج الحواف الحاسمة من الذاكرة ويمنع تراكم الحشو. **مدخلات:** task state، claims، uncertainty map، memory edges، available tools. **مخرجات:** structured object يحتوي على failure، evidence، confidence، decision، mode، وnext action. **لماذا هي غير عادية؟** لأنها لا تضيف خطوة لغوية فقط؛ تضيف جهاز تحكم داخل التفكير. الأداة تقيس شيئا، تغير routing، أو تمنع مسارا. إذا لم تفعل واحدا من هذه الثلاثة فهي ليست أداة AGI بل prompt decoration. **اختبار مصغر:** نعطيها مهمة فيها تضارب خفي. النجاح ليس أن تجيب، بل أن تكشف التضارب وتبني إجراء فحص. في benchmark، تقاس بقيمة: ΔScore/Cost، أي مقدار التحسن مقابل تكلفة إضافية. **موقعها في HAKIM:** تدخل بين Difference Extractor وWorld-Effect Loop، وتكتب إلى Self-Reconstruction Ledger إذا غيّرت القرار.

تنزيل هندسي داخل MICAI

تتحول الفكرة هنا إلى وحدة تشغيل عبر ثلاث طبقات: طبقة تمثيل تصوغ المسألة في صورة state، طبقة حاكمية تقرر أي مسار يستحق compute، وطبقة أثر تختبر النتيجة في عالم صغير. إذا بقيت الفكرة داخل اللغة دون state أو اختبار فهي لم تدخل neoAGI-v3 بعد، ولو بدت عميقة.

في التطبيق، تُكتب كل وحدة كـ contract: ماذا تستقبل؟ ماذا تعيد؟ ما شروط الفشل؟ وكيف تعرف أن الأداة أضافت قيمة لا ضوضاء؟ هذا يمنع تضخم المصطلحات ويجعل الاختراع قابلا للتحكيم. القوة الحقيقية ليست في اسم الأداة، بل في قدرتها على تغيير قرار النموذج عند لحظة حرجة.

> مؤشر القبول: ارتفاع transfer أو انخفاض hallucination أو ضغط تكلفة التفكير أو كشف failure mode جديد.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد؛ لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

الأداة التقنية 12: Benchmark Dialectic Layer

وظيفة الأداة: يربط كل benchmark بنمط الفشل الذي يكشفه، لا برقم خام فقط. **مدخلات:** task state، claims، uncertainty، memory edges، available tools، map. **مخرجات:** structured object يحتوي على evidence، confidence، decision، failure mode، وnext action. **لماذا هي غير عادية؟** لأنها لا تضيف خطوة لغوية فقط؛ تضيف جهاز تحكم داخل التفكير. الأداة تقيس شيئا، تغير routing، أو تمنع مسارا. إذا لم تفعل واحدا من هذه الثلاثة فهي ليست أداة AGI بل prompt decoration. **اختبار مصغر:** نعطيها مهمة فيها تضارب خفي. النجاح ليس أن تجيب، بل أن تكشف التضارب وتبني إجراء فحص. في benchmark، تقاس بقيمة: ΔScore/Cost، أي مقدار التحسن مقابل تكلفة إضافية. **موقعها في HAKIM:** تدخل بين Difference Extractor وWorld-Effect Loop، وتكتب إلى Self-Reconstruction Ledger إذا غيّرت القرار.

طبقة الاختبار والاعتراض

كل مكون في neoAGI-v3 يمر عبر اعتراضين على الأقل: اعتراض معرفي يسأل عن صحة البنية، واعتراض تشغيلي يسأل عن التكلفة والاختبار. لا نقبل مكونا لأنه يبدو قويا، نقبله لأنه يغير نتيجة في task غير مرئي أو يكشف خطأ كان مخفيا.

لهذا ترتبط كل صفحة من الكتاب بمحكمة perturbation: ما الحالة التي تفشل فيها؟ ما النموذج الأبسط الذي يهزمها؟ هل يمكن استبدالها بprompt فقط؟ إذا نعم فهي ليست technology مستقلة. أما إذا كانت تفرض trace وقرارا وقياسا، فهي تدخل طبقة AGI tools.

> مبدأ الحاكمية: الجرأة تصنع الفرضية، والاختبار يمنعها من التحول إلى وهم.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

الأداة التقنية 13: Safety Governor

وظيفة الأداة: يراقب الاستقلالية والأدوات والتأثير الخارجي ويوقف المسارات عالية الخطر. **مدخلات:** claims، task state، memory edges، available tools، uncertainty map. **مخرجات:** structured object يحتوي على confidence، decision، failure mode، evidence، وnext action. **لماذا هي غير عادية؟** لأنها لا تضيف خطوة لغوية فقط؛ تضيف جهاز تحكم داخل التفكير. الأداة تقيس شيئا، تغير routing، أو تمنع مسارا. إذا لم تفعل واحدا من هذه الثلاثة فهي ليست أداة AGI بل prompt decoration. **اختبار مصغر:** نعطيها مهمة فيها تضارب خفي. النجاح ليس أن تجيب، بل أن تكشف التضارب وتبني إجراء فحص. في benchmark، تقاس بقيمة: ΔScore/Cost، أي مقدار التحسن مقابل تكلفة إضافية. **موقعها في HAKIM:** تدخل بين Difference Extractor وWorld-Effect Loop، وتكتب إلى Self-Reconstruction Ledger إذا غيّرت القرار.

علاقة هذه الوحدة بالمعايير

لا تُقرأ أي وحدة بعيدا عن benchmark. في البرمجة نطلب patch صحيحا لا شرحا؛ في الرياضيات نطلب proof obligations لا حدسا؛ في ARC نطلب induction قابل للنقل؛ في Terminal-Bench نطلب أفعالا آمنة داخل بيئة؛ وفي HLE نطلب تحقق مصادر وحدود ثقة. لذلك فالمكون الجيد يملك أكثر من mode: reasoning، tool use، verification، repair.

عندما تفشل الوحدة، نسجل أين وقع الفشل: transfer، safety، cost، proof، tool، memory، representation، goal. هذه الثمانية هي لغة التشخيص الموحدة في الكتاب. دونها تصبح الأرقام خامة لا معرفة.

> النتيجة المرغوبة: score + topology + repair. الثلاثة معا فقط تعطي علما قابلا للبناء.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر؛ الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

الأداة التقنية 14: Self-Reconstruction Ledger

وظيفة الأداة: سجل تغيّر النموذج بعد كل تجربة: ما تحسن، ما فشل، وما يجب نسيانه. **مدخلات:** uncertainty، claims، task state، memory edges، available tools، map. **مخرجات:** structured object يحتوي على evidence، confidence، decision، failure mode، وnext action. **لماذا هي غير عادية؟** لأنها لا تضيف خطوة لغوية فقط؛ تضيف جهاز تحكم داخل التفكير. الأداة تقيس شيئا، تغير routing، أو تمنع مسارا. إذا لم تفعل واحدا من هذه الثلاثة فهي ليست أداة AGI بل prompt decoration. **اختبار مصغر:** نعطيها مهمة فيها تضارب خفي. النجاح ليس أن تجيب، بل أن تكشف التضارب وتبني إجراء فحص. في benchmark، تقاس بقيمة: ΔScore/Cost، أي مقدار التحسن مقابل تكلفة إضافية. **موقعها في HAKIM:** تدخل بين Difference Extractor وWorld-Effect Loop، وتكتب إلى Self-Reconstruction Ledger إذا غيّرت القرار.

منع السلوب وتحويل النص إلى آلة

السلوك اللغوي الشائع في كتب AGI هو إضافة عبارات كبرى دون قفل تشغيلي. هنا نعكس القاعدة: كل عبارة كبيرة يجب أن تتكمش إلى إجراء صغير يمكن تشغيله. إذا قلنا «حاكمية» نحدد gate. إذا قلنا «ذاكرة» نحدد edge. إذا قلنا «جدة» نحدد metric. وإذا قلنا «وعي بالخطأ» نحدد signal يوقف المسار.

هذا الانضباط لا يقلل الطموح؛ بل يجعله قابلا للتمويل والبناء والمراجعة. الاختراع الحقيقي في neoAGI-v3 هو الربط بين المصطلح والوظيفة والقياس، بحيث لا يعود المشروع مجرد فلسفة بل هندسة معرفية.

> اختبار anti-slop: احذف كل جملة لا تغير قرارا أو قياسا أو تجربة.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد، لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

الأداة التقنية 15: AGI Readiness Gate

وظيفة الأداة: بوابة قبول لا تمرر التقنية إلا إذا حققت عمومية، نقل، سلامة، وتكلفة معقولة. **مدخلات:** task state، claims، memory edges، available tools، uncertainty map. **مخرجات:** structured object يحتوي على confidence، decision، failure mode، evidence، وnext action. **لماذا هي غير عادية؟** لأنها لا تضيف خطوة لغوية فقط؛ تضيف جهاز تحكم داخل التفكير. الأداة تقيس شيئا، تغير routing، أو تمنع مسارا. إذا لم تفعل واحدا من هذه الثلاثة فهي ليست أداة AGI بل prompt decoration. **اختبار مصغر:** نعطيها مهمة فيها تضارب خفي. النجاح ليس أن تجيب، بل أن تكشف التضارب وتبني إجراء فحص. في benchmark، تقاس بقيمة: ΔScore/Cost، أي مقدار التحسن مقابل تكلفة إضافية. **موقعها في HAKIM:** تدخل بين Difference Extractor وWorld-Effect Loop، وتكتب إلى Self-Reconstruction Ledger إذا غيّرت القرار.

تنزيل هندسي داخل MICAI

تتحول الفكرة هنا إلى وحدة تشغيل عبر ثلاث طبقات: طبقة تمثيل تصوغ المسألة في صورة state، طبقة حاكمية تقرر أي مسار يستحق compute، وطبقة أثر تختبر النتيجة في عالم صغير. إذا بقيت الفكرة داخل اللغة دون state أو اختبار فهي لم تدخل neoAGI-v3 بعد، ولو بدت عميقة.

في التطبيق، تُكتب كل وحدة كـ contract: ماذا تستقبل؟ ماذا تعيد؟ ما شروط الفشل؟ وكيف تعرف أن الأداة أضافت قيمة لا ضوضاء؟ هذا يمنع تضخم المصطلحات ويجعل الاختراع قابلا للتحكيم. القوة الحقيقية ليست في اسم الأداة، بل في قدرتها على تغيير قرار النموذج عند لحظة حرجة.

> مؤشر القبول: ارتفاع transfer أو انخفاض hallucination أو ضغط تكلفة التفكير أو كشف failure mode جديد.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

فلسفة التدريب 1: Training by Failure Geometry

لا ندرّب النموذج على الإجابة فقط؛ ندرّبه على شكل الفشل: أين ينهار؟ كيف يعرف أنه ينهار؟ وكيف يعود؟ فلسفة التدريب هنا ليست شعارا؛ هي سياسة بيانات وسياسة loss وسياسة eval. عندما ندرب by Failure Geometry، لا نكتفي ببيانات answer/correct. نحتاج traces لأين بدأ الانحراف، وأي critic كان يجب أن يستيقظ، وأي tool كان يجب أن يستدعى. وعندما ندرب by World Effect، لا نقبل جوابا لا ينتج artifact أو proof أو test. تطبيق عملي: كل مثال تدريبي يحوي خمسة حقول: naive، problem، executable check، repaired reasoning، failure topology، answer. هذا يحول dataset من مستودع أسئلة إلى مختبر تعليم للحاكمية. الفلسفة تقاوم slop لأنها تجعل النص غير كاف: يجب أن يحمل أثره. مؤشر نجاحها: انخفاض self-confident wrong answers، وارتفاع repair rate، وتحسن transfer إلى benchmarks غير مرئية.

طبقة الاختبار والاعتراض

كل مكون في neoAGI-v3 يمر عبر اعتراضين على الأقل: اعتراض معرفي يسأل عن صحة البنية، واعتراض تشغيلي يسأل عن التكلفة والاختبار. لا نقبل مكونا لأنه يبدو قويا؛ نقبله لأنه يغير نتيجة في task غير مرئي أو يكشف خطأ كان مخفيا.

لهذا ترتبط كل صفحة من الكتاب بمحكمة perturbation: ما الحالة التي تفشل فيها؟ ما النموذج الأبسط الذي يهزمها؟ هل يمكن استبدالها بprompt فقط؟ إذا نعم فهي ليست technology مستقلة. أما إذا كانت تفرض trace وقرارا وقياسا، فهي تدخل طبقة AGI tools.

> مبدأ الحاكمية: الجرأة تصنع الفرضية، والاختبار يمنعها من التحول إلى وهم.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر؛ الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

فلسفة التدريب 2: Training by World Effect

كل تدريب يجب أن ينتهي بأثر قابل للفحص في العالم: برنامج يعمل، برهان يتحقق، تجربة تتكرر، أو قرار آمن. فلسفة التدريب هنا ليست شعارا؛ هي سياسة بيانات وسياسة loss وسياسة eval. عندما ندرب by Failure Geometry، لا نكتفي ببيانات answer/correct. نحتاج traces لأين بدأ الانحراف، وأي critic كان يجب أن يستيقظ، وأي tool كان يجب أن يستدعى. وعندما ندرب by World Effect، لا نقبل جوابا لا ينتج artifact أو proof أو test. تطبيق عملي: كل مثال تدريبي يحوي خمسة حقول: naive، problem، executable check، repaired reasoning، failure topology، answer. هذا يحول dataset من مستودع أسئلة إلى مختبر تعليم للحاكمية. الفلسفة تقاوم slop لأنها تجعل النص غير كاف: يجب أن يحمل أثره. مؤشر نجاحها: انخفاض self-confident wrong answers، وارتفاع repair rate، وتحسن transfer إلى benchmarks غير مرئية.

علاقة هذه الوحدة بالمعايير

لا تُقرأ أي وحدة بعيدا عن benchmark. في البرمجة نطلب patch صحيحا لا شرحا؛ في الرياضيات نطلب proof obligations لا حدسا؛ في ARC نطلب induction قابل للنقل؛ في Terminal-Bench نطلب أفعالا آمنة داخل بيئة؛ وفي HLE نطلب تحقق مصادر وحدود ثقة. لذلك فالمكون الجيد يملك أكثر من mode: reasoning، tool use، verification، repair.

عندما تفشل الوحدة، نسجل أين وقع الفشل: transfer، safety، cost، proof، tool، memory، representation، goal. هذه الثمانية هي لغة التشخيص الموحدة في الكتاب. دونها تصبح الأرقام خامة لا معرفة.

> النتيجة المرغوبة: repair + topology + score. الثلاثة معا فقط تعطي علما قابلا للبناء.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد، لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

فلسفة التدريب 3: Training by Vascular Attention

الانتباه ليس لكل شيء؛ الانتباه للأوعية التي تحمل أكبر فرق فاعل في اللحظة. فلسفة التدريب هنا ليست شعارا؛ هي سياسة بيانات وسياسة loss وسياسة eval. عندما ندرب by Failure Geometry، لا نكتفي ببيانات answer/correct. نحتاج traces لأين بدأ الانحراف، وأي critic كان يجب أن يستيقظ، وأي tool كان يجب أن يستدعى. وعندما ندرب by World Effect، لا نقبل جوابا لا ينتج artifact أو proof أو test. تطبيق عملي: كل مثال تدريبي يحوي خمسة حقول: problem، naive answer، failure، executable check، repaired reasoning، topology. هذا يحول dataset من مستودع أسئلة إلى مختبر تعليم للحاكمية. الفلسفة تقاوم slop لأنها تجعل النص غير كاف: يجب أن يحمل أثره. مؤشر نجاحها: انخفاض self-confident wrong answers، وارتفاع repair rate، وتحسن transfer إلى benchmarks غير مرئية.

منع السلوب وتحويل النص إلى آلة

السلوك اللغوي الشائع في كتب AGI هو إضافة عبارات كبرى دون قفل تشغيلي. هنا نعكس القاعدة: كل عبارة كبيرة يجب أن تتكمش إلى إجراء صغير يمكن تشغيله. إذا قلنا «حاكمية» نحدد gate. إذا قلنا «ذاكرة» نحدد edge. إذا قلنا «جدة» نحدد metric. وإذا قلنا «وعي بالخطأ» نحدد signal يوقف المسار.

هذا الانضباط لا يقلل الطموح؛ بل يجعله قابلا للتمويل والبناء والمراجعة. الاختراع الحقيقي في neoAGI-v3 هو الربط بين المصطلح والوظيفة والقياس، بحيث لا يعود المشروع مجرد فلسفة بل هندسة معرفية.

> اختبار anti-slop: احذف كل جملة لا تغير قرارا أو قياسا أو تجربة.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تُعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

فلسفة التدريب 4: Training by Anti-Slop Compression

كل عبارة لا تضيف اختبارا أو تمييزا أو آلية تُضغط أو تُحذف. فلسفة التدريب هنا ليست شعارا؛ هي سياسة بيانات وسياسة loss وسياسة eval. عندما ندرب by Failure Geometry، لا نكتفي ببيانات answer/correct. نحتاج traces لأين بدأ الانحراف، وأي critic كان يجب أن يستيقظ، وأي tool كان يجب أن يستدعى. وعندما ندرب by World Effect، لا نقبل جوابا لا ينتج artifact أو proof أو test. تطبيق عملي: كل مثال تدريبي يحوي خمسة حقول: repaired، failure topology، naive answer، problem، executable check، reasoning. هذا يحول dataset من مستودع أسئلة إلى مختبر تعليم للحاكمية. الفلسفة تقاوم slop لأنها تجعل النص غير كاف: يجب أن يحمل أثره. مؤشر نجاحها: انخفاض self-confident wrong answers، وارتفاع repair rate، وتحسن transfer إلى benchmarks غير مرئية.

تنزيل هندسي داخل MICAI

تتحول الفكرة هنا إلى وحدة تشغيل عبر ثلاث طبقات: طبقة تمثيل تصوغ المسألة في صورة state، طبقة حاكمية تقرر أي مسار يستحق compute، وطبقة أثر تختبر النتيجة في عالم صغير. إذا بقيت الفكرة داخل اللغة دون state أو اختبار فهي لم تدخل neoAGI-v3 بعد، ولو بدت عميقة.

في التطبيق، تُكتب كل وحدة كـ contract: ماذا تستقبل؟ ماذا تعيد؟ ما شروط الفشل؟ وكيف تعرف أن الأداة أضافت قيمة لا ضوضاء؟ هذا يمنع تضخم المصطلحات ويجعل الاختراع قابلا للتحكيم. القوة الحقيقية ليست في اسم الأداة، بل في قدرتها على تغيير قرار النموذج عند لحظة حرجة.

> مؤشر القبول: ارتفاع transfer أو انخفاض hallucination أو ضغط تكلفة التفكير أو كشف failure mode جديد.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر؛ الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

فلسفة التدريب 5: Training by Governance Before Fluency

الفصاحة لا تسبق الحاكمية؛ النموذج يتعلم أن يحكم على المسار قبل أن يزيّنه. فلسفة التدريب هنا ليست شعارا؛ هي سياسة بيانات وسياسة loss وسياسة eval. عندما ندرب by Failure Geometry، لا نكتفي ببيانات answer/correct. نحتاج traces لأين بدأ الانحراف، وأي critic كان يجب أن يستيقظ، وأي tool كان يجب أن يستدعى. وعندما ندرب by World Effect، لا نقبل جوابا لا ينتج artifact أو proof أو test. تطبيق عملي: كل مثال تدريبي يحوي خمسة حقول: failure topology، naive answer، problem، executable check، repaired reasoning. هذا يحول dataset من مستودع أسئلة إلى مختبر تعليم للحاكمية. الفلسفة تقاوم slop لأنها تجعل النص غير كاف: يجب أن يحمل أثره. مؤشر نجاحها: انخفاض self-confident wrong answers، وارتفاع repair rate، وتحسن transfer إلى benchmarks غير مرئية.

طبقة الاختبار والاعتراض

كل مكون في neoAGI-v3 يمر عبر اعتراضين على الأقل: اعتراض معرفي يسأل عن صحة البنية، واعتراض تشغيلي يسأل عن التكلفة والاختبار. لا نقبل مكونا لأنه يبدو قويا؛ نقبله لأنه يغير نتيجة في task غير مرئي أو يكشف خطأ كان مخفيا.

لهذا ترتبط كل صفحة من الكتاب بحكمة perturbation: ما الحالة التي تفشل فيها؟ ما النموذج الأبسط الذي يهزمها؟ هل يمكن استبدالها بprompt فقط؟ إذا نعم فهي ليست technology مستقلة. أما إذا كانت تفرض trace وقرارا وقياسا، فهي تدخل طبقة AGI tools.

> مبدأ الحاكمية: الجرأة تصنع الفرضية، والاختبار يمنعها من التحول إلى وهم.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد، لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

أداة التدريب 1: MCD Curriculum Generator

Pipeline: generate tasks -> map MCD -> force failure -> collect trace -> repair -> distill rule -> re-evaluate on unseen variants. هذه الأداة تعالج مرضا معروفا في النماذج المفكرة: قد تكتب reasoning مقنعا بعد الجواب، لا reasoning صنع الجواب. لذلك نحتاج تدريب traces مربوطة بفحوص خارجية. إذا كان المجال برمجة، فالفحص tests. إذا كان رياضيات، فالفحص proof obligations. إذا كان عالما تجريديا، فالفحص environment score. إذا كان معرفة، فالفحص evidence verification. مخرج الأداة ليس model فقط، مخرجها أيضا dataset من أخطاء محكومة. ومع الوقت يصبح للنظام ذاكرة فشل: يعرف أنه في نوع معين من المسائل يميل إلى قفزة خاطئة، فيستدعي critic قبل القفزة.

علاقة هذه الوحدة بالمعايير

لا تُقرأ أي وحدة بعيدا عن benchmark. في البرمجة نطلب patch صحيحا لا شرحا؛ في الرياضيات نطلب proof obligations لا حدسا؛ في ARC نطلب induction قابل للنقل؛ في Terminal-Bench نطلب أفعالا آمنة داخل بيئة؛ وفي HLE نطلب تحقق مصادر وحدود ثقة. لذلك فالمكون الجيد يملك أكثر من mode: reasoning، tool use، verification، repair.

عندما تفشل الوحدة، نسجل أين وقع الفشل: transfer، safety، cost، proof، tool، memory، representation، goal. هذه الثمانية هي لغة التشخيص الموحدة في الكتاب. دونها تصبح الأرقام خامة لا معرفة.

> النتيجة المرغوبة: score + topology + repair. الثلاثة معا فقط تعطي علما قابلا للبناء.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تُعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

أداة التدريب 2: Error-Topology Replay

Pipeline: generate tasks -> map MCD -> force failure -> collect يعيد تشغيل أخطاء النموذج كخرائط فشل قابلة للتعلم. trace -> repair -> distill rule -> re-evaluate on unseen variants. هذه الأداة تعالج مرضا معروفا في النماذج المفكرة: قد تكتب reasoning مقنعا بعد الجواب، لا reasoning صنع الجواب. لذلك نحتاج تدريب traces مربوطة بفحوص خارجية. إذا كان المجال برمجة، فالفحص tests. إذا كان رياضيات، فالفحص proof obligations. إذا كان عالما تجريديا، فالفحص environment score. إذا كان معرفة، فالفحص evidence verification. مخرج الأداة ليس model فقط؛ مخرجها أيضا dataset من أخطاء محكومة. ومع الوقت يصبح للنظام ذاكرة فشل: يعرف أنه في نوع معين من المسائل يميل إلى قفزة خاطئة، فيستدعي critic قبل القفزة.

منع السلوب وتحويل النص إلى آلة

السلوك اللغوي الشائع في كتب AGI هو إضافة عبارات كبرى دون قفل تشغيلي. هنا نعكس القاعدة: كل عبارة كبيرة يجب أن تتكمش إلى إجراء صغير يمكن تشغيله. إذا قلنا «حاكمية» نحدد gate. إذا قلنا «ذاكرة» نحدد edge. إذا قلنا «جدة» نحدد metric. وإذا قلنا «وعي بالخطأ» نحدد signal يوقف المسار.

هذا الانضباط لا يقلل الطموح؛ بل يجعله قابلا للتمويل والبناء والمراجعة. الاختراع الحقيقي في neoAGI-v3 هو الربط بين المصطلح والوظيفة والقياس، بحيث لا يعود المشروع مجرد فلسفة بل هندسة معرفية.

> اختبار anti-slop: احذف كل جملة لا تغير قرارا أو قياسا أو تجربة.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر؛ الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

أداة التدريب 3: Adversarial Worldlets

عوالم صغيرة مصممة لإجبار النموذج على اكتشاف القاعدة لا حفظ المثال. **Pipeline:** generate tasks -> map MCD -> force failure -> collect trace -> repair -> distill rule -> re-evaluate on unseen variants. هذه الأداة تعالج مرضا معروفا في النماذج المفكرة: قد تكتب reasoning مقنعا بعد الجواب، لا reasoning صنع الجواب. لذلك نحتاج تدريب traces مربوطة بفحوص خارجية. إذا كان المجال برمجة، فالفحص tests. إذا كان رياضيات، فالفحص proof obligations. إذا كان عالما تجريديا، فالفحص environment score. إذا كان معرفة، فالفحص evidence verification. مخرج الأداة ليس model فقط، مخرجها أيضا dataset من أخطاء محكومة. ومع الوقت يصبح للنظام ذاكرة فشل: يعرف أنه في نوع معين من المسائل يميل إلى قفزة خاطئة، فيستدعي critic قبل القفزة.

تنزيل هندسي داخل MICAI

تتحول الفكرة هنا إلى وحدة تشغيل عبر ثلاث طبقات: طبقة تمثيل تصوغ المسألة في صورة state، طبقة حاكمية تقرر أي مسار يستحق compute، وطبقة أثر تختبر النتيجة في عالم صغير. إذا بقيت الفكرة داخل اللغة دون state أو اختبار فهي لم تدخل neoAGI-v3 بعد، ولو بدت عميقة.

في التطبيق، تُكتب كل وحدة كـ contract: ماذا تستقبل؟ ماذا تعيد؟ ما شروط الفشل؟ وكيف تعرف أن الأداة أضافت قيمة لا ضوضاء؟ هذا يمنع تضخم المصطلحات ويجعل الاختراع قابلا للتحكيم. القوة الحقيقية ليست في اسم الأداة، بل في قدرتها على تغيير قرار النموذج عند لحظة حرجة.

> مؤشر القبول: ارتفاع transfer أو انخفاض hallucination أو ضغط تكلفة التفكير أو كشف failure mode جديد.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد، لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

أداة التدريب 4: Proof-Trace Distiller

Pipeline: generate tasks -> map MCD -> force failure <- collect trace -> repair -> distill rule -> re-evaluate on unseen variants. هذه الأداة تعالج مرضا معروفا في النماذج المفكرة: قد تكتب reasoning مقنعا بعد الجواب، لا reasoning صنع الجواب. لذلك نحتاج تدريب traces مربوطة بفحوص خارجية. إذا كان المجال برمجة، فالفحص tests. إذا كان رياضيات، فالفحص proof obligations. إذا كان عالما تجريديا، فالفحص environment score. إذا كان معرفة، فالفحص evidence verification. مخرج الأداة ليس model فقط، مخرجها أيضا dataset من أخطاء محكومة. ومع الوقت يصبح للنظام ذاكرة فشل: يعرف أنه في نوع معين من المسائل يميل إلى قفزة خاطئة، فيستدعي critic قبل القفزة.

طبقة الاختبار والاعتراض

كل مكون في neoAGI-v3 يمر عبر اعتراضين على الأقل: اعتراض معرفي يسأل عن صحة البنية، واعتراض تشغيلي يسأل عن التكلفة والاختبار. لا نقبل مكونا لأنه يبدو قويا، نقبله لأنه يغير نتيجة في task غير مرئي أو يكشف خطأ كان مخفيا.

لهذا ترتبط كل صفحة من الكتاب بمحكمة perturbation: ما الحالة التي تفشل فيها؟ ما النموذج الأبسط الذي يهزمها؟ هل يمكن استبدالها بprompt فقط؟ إذا نعم فهي ليست technology مستقلة. أما إذا كانت تفرض trace وقرارا وقياسا، فهي تدخل طبقة AGI tools.

> مبدأ الحاكمية: الجرأة تصنع الفرضية، والاختبار يمنعها من التحول إلى وهم.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

أداة التدريب 5: Autonomy Sandbox Trainer

يدرب الوكلاء داخل حاويات آمنة مع قياس تكلفة القرار، لا صحة الجواب فقط. **Pipeline:** generate tasks -> map MCD -> force failure -> collect trace -> repair -> distill rule -> re-evaluate on unseen variants. هذه الأداة تعالج مرضا معروفا في النماذج المفكرة: قد تكتب reasoning مقنعا بعد الجواب، لا reasoning صنع الجواب. لذلك نحتاج تدريب traces مربوطة بفحوص خارجية. إذا كان المجال برمجة، فالفحص tests. إذا كان رياضيات، فالفحص proof obligations. إذا كان عالما تجريديا، فالفحص environment score. إذا كان معرفة، فالفحص evidence verification. مخرج الأداة ليس model فقط، مخرجها أيضا dataset من أخطاء محكومة. ومع الوقت يصبح للنظام ذاكرة فشل: يعرف أنه في نوع معين من المسائل يميل إلى قفزة خاطئة، فيستدعي critic قبل القفزة.

علاقة هذه الوحدة بالمعايير

لا تُقرأ أي وحدة بعيدا عن benchmark. في البرمجة نطلب patch صحيحا لا شرحا؛ في الرياضيات نطلب proof obligations لا حدسا؛ في ARC نطلب induction قابل للنقل؛ في Terminal-Bench نطلب أفعالا آمنة داخل بيئة؛ وفي HLE نطلب تحقق مصادر وحدود ثقة. لذلك فالمكون الجيد يملك أكثر من mode: reasoning، tool use، verification، repair.

عندما تفشل الوحدة، نسجل أين وقع الفشل: transfer، safety، cost، proof، tool، memory، representation، goal. هذه الثمانية هي لغة التشخيص الموحدة في الكتاب. دونها تصبح الأرقام خامة لا معرفة.

> النتيجة المرغوبة: score + topology + repair. الثلاثة معا فقط تعطي علما قابلا للبناء.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر؛ الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

Benchmark 1: ARC-AGI-3

الوظيفة التشخيصية: agentic fluid intelligence. **الهدف التصميمي:** 22.0%. **تنبيه:** اقتراح هدف صعب لا نتيجة حقيقية. نقرأ ARC-AGI-3 لا كرقم بل كمرآة فشل. إذا فشل neoAGI-v3 هنا، نسأل: هل الفشل من representation؟ هل من نقص tool؟ هل من ضعف self-check؟ هل من تكلفة؟ هل من عدم انتقال؟ ثم نربط الجواب بأداة من الأدوات الخمس عشرة. **تأثير تقنياتنا:** HAKIM Hyperloop Kernel يرفع جودة المسار؛ Vascular Memory Reactor يمنع تكرار أخطاء سابقة؛ Proof-World Simulator يحول الادعاء إلى تجربة. في هذا benchmark تحديدا، أهم أدوات الدعم هي: Inner-Time، Δ-Extractor، Allocator، وHypothesis Alloying Unit. **بروتوكول الاختبار الحقيقي:** تشغيل harness رسمي، تثبيت model/base، منع بيانات التدريب من التسرب، تحديد عدد المحاولات، تسجيل traces، حساب التكلفة، ثم نشر failure topology لا score فقط.

منع السلوب وتحويل النص إلى آلة

السلوك اللغوي الشائع في كتب AGI هو إضافة عبارات كبرى دون قفل تشغيلي. هنا نعكس القاعدة: كل عبارة كبيرة يجب أن تتكمش إلى إجراء صغير يمكن تشغيله. إذا قلنا «حاكمية» نحدد gate. إذا قلنا «ذاكرة» نحدد edge. إذا قلنا «جدة» نحدد metric. وإذا قلنا «وعي بالخطأ» نحدد signal يوقف المسار.

هذا الانضباط لا يقلل الطموح؛ بل يجعله قابلا للتمويل والبناء والمراجعة. الاختراع الحقيقي في neoAGI-v3 هو الربط بين المصطلح والوظيفة والقياس، بحيث لا يعود المشروع مجرد فلسفة بل هندسة معرفية.

> اختبار anti-slop: احذف كل جملة لا تغير قرارا أو قياسا أو تجربة.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد، لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

Benchmark 2: ARC-AGI-2

الوظيفة التشخيصية: abstract reasoning. **الهدف التصميمي:** 68.0%. **تنبيه:** Targeted simulated. نقرأ ARC-AGI-2 لا كرقم بل كمرآة فشل. إذا فشل neoAGI-v3 هنا، نسأل: هل الفشل من representation؟ هل من نقص tool؟ هل من ضعف self-check؟ هل من تكلفة؟ هل من عدم انتقال؟ ثم نربط الجواب بأداة من الأدوات الخمس عشرة. **تأثير تقنياتنا:** HAKIM Hyperloop Kernel يرفع جودة المسار، Vascular Memory Reactor يمنع تكرار أخطاء سابقة، Proof-World Simulator يحول الادعاء إلى تجربة. في هذا benchmark تحديدا، أهم أدوات الدعم هي: Perturbation Court، MCD-Map Engine، وNovelty Entropy Meter. **بروتوكول الاختبار الحقيقي:** تشغيل harness رسمي، تثبيت model/base، منع بيانات التدريب من التسرب، تحديد عدد المحاولات، تسجيل traces، حساب التكلفة، ثم نشر failure topology لا score فقط.

تنزيل هندسي داخل MICAI

تتحول الفكرة هنا إلى وحدة تشغيل عبر ثلاث طبقات: طبقة تمثيل تصوغ المسألة في صورة state، طبقة حاكمية تقرر أي مسار يستحق compute، وطبقة أثر تختبر النتيجة في عالم صغير. إذا بقيت الفكرة داخل اللغة دون state أو اختبار فهي لم تدخل neoAGI-v3 بعد، ولو بدت عميقة.

في التطبيق، تُكتب كل وحدة كـ contract: ماذا تستقبل؟ ماذا تعيد؟ ما شروط الفشل؟ وكيف تعرف أن الأداة أضافت قيمة لا ضوضاء؟ هذا يمنع تضخم المصطلحات ويجعل الاختراع قابلا للتحكيم. القوة الحقيقية ليست في اسم الأداة، بل في قدرتها على تغيير قرار النموذج عند لحظة حرجة.

> مؤشر القبول: ارتفاع transfer أو انخفاض hallucination أو ضغط تكلفة التفكير أو كشف failure mode جديد.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

Benchmark 3: SWE-bench Verified

الوظيفة التشخيصية: real software repair. **الهدف التصميمي:** 82.0%. **تنبيه:** Requires harness. نقرأ SWE-bench Verified لا كرقم بل كمرآة فشل. إذا فشل neoAGI-v3 هنا، نسأل: هل الفشل من representation؟ هل من نقص tool؟ هل من ضعف self-check؟ هل من تكلفة؟ هل من عدم انتقال؟ ثم نربط الجواب بأداة من الأدوات الخمس عشرة. **تأثير تقنياتنا:** HAKIM Hyperloop Kernel يرفع جودة المسار، Vascular Memory Reactor يمنع تكرار أخطاء سابقة، Proof-World Simulator يحول الادعاء إلى تجربة. في هذا benchmark تحديدا، أهم أدوات الدعم هي: Counterexample، Vascular Compute Router، Forge، وWorld-Effect Compiler. **بروتوكول الاختبار الحقيقي:** تشغيل harness رسمي، تثبيت model/base، منع بيانات التدريب من التسرب، تحديد عدد المحاولات، تسجيل traces، حساب التكلفة، ثم نشر failure topology لا score فقط.

طبقة الاختبار والاعتراض

كل مكون في neoAGI-v3 يمر عبر اعتراضين على الأقل: اعتراض معرفي يسأل عن صحة البنية، واعتراض تشغيلي يسأل عن التكلفة والاختبار. لا نقبل مكونا لأنه يبدو قويا، نقبله لأنه يغير نتيجة في task غير مرئي أو يكشف خطأ كان مخفيا.

لهذا ترتبط كل صفحة من الكتاب بمحكمة perturbation: ما الحالة التي تفشل فيها؟ ما النموذج الأبسط الذي يهزمها؟ هل يمكن استبدالها بprompt فقط؟ إذا نعم فهي ليست technology مستقلة. أما إذا كانت تفرض trace وقرارا وقياسا، فهي تدخل طبقة AGI tools.

> مبدأ الحاكمية: الجرأة تصنع الفرضية، والاختبار يمنعها من التحول إلى وهم.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر، الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

Benchmark 4: Terminal-Bench

الوظيفة التشخيصية: terminal agents. **الهدف التصميمي:** 66.0%. **تنبيه:** Requires sandbox. نقرأ Terminal-Bench لا كرقم بل كمرآة فشل. إذا فشل neoAGI-v3 هنا، نسأل: هل الفشل من representation؟ هل من نقص tool؟ هل من ضعف self-check؟ هل من تكلفة؟ هل من عدم انتقال؟ ثم نربط الجواب بأداة من الأدوات الخمس عشرة. **تأثير تقنياتنا:** HAKIM Hyperloop Kernel يرفع جودة المسار؛ Vascular Memory Reactor يمنع تكرار أخطاء سابقة؛ Proof-World Simulator يحول الادعاء إلى تجربة. في هذا benchmark تحديدا، أهم أدوات الدعم هي: Hypothesis Alloying Unit، Inner-Time Allocator، وSkill Hyperloop Builder. **بروتوكول الاختبار الحقيقي:** تشغيل harness رسمي، تثبيت model/base، منع بيانات التدريب من التسرب، تحديد عدد المحاولات، تسجيل traces، حساب التكلفة، ثم نشر failure topology لا score فقط.

علاقة هذه الوحدة بالمعايير

لا تُقرأ أي وحدة بعيدا عن benchmark. في البرمجة نطلب patch صحيحا لا شرحا؛ في الرياضيات نطلب proof obligations لا حدسا؛ في ARC نطلب induction قابل للنقل؛ في Terminal-Bench نطلب أفعالا آمنة داخل بيئة؛ وفي HLE نطلب تحقق مصادر وحدود ثقة. لذلك فالمكون الجيد يملك أكثر من mode: reasoning، tool use، verification، repair.

عندما تفشل الوحدة، نسجل أين وقع الفشل: transfer، safety، cost، proof، tool، memory، representation، goal. هذه الثمانية هي لغة التشخيص الموحدة في الكتاب. دونها تصبح الأرقام خامة لا معرفة.

> النتيجة المرغوبة: score + topology + repair. الثلاثة معا فقط تعطي علما قابلا للبناء.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد، لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

Benchmark 5: LiveCodeBench

الوظيفة التشخيصية: fresh coding. **الهدف التصميمي:** 76.0%. **تنبيه:** Time-window dependent. نقرأ LiveCodeBench لا كرقم بل كمرآة فشل. إذا فشل neoAGI-v3 هنا، نسأل: هل الفشل من representation؟ هل من نقص tool؟ هل من ضعف self-check؟ هل من تكلفة؟ هل من عدم انتقال؟ ثم نربط الجواب بأداة من الأدوات الخمس عشرة. **تأثير تقنياتنا:** HAKIM يرفع جودة المسار، Vascular Memory Reactor يمنع تكرار أخطاء سابقة، Proof-World Simulator Hyperloop Kernel يحول الادعاء إلى تجربة. في هذا benchmark تحديدا، أهم أدوات الدعم هي: Novelty Entropy، Perturbation Court، Meter، وMemory Edge Distiller. **بروتوكول الاختبار الحقيقي:** تشغيل harness رسمي، تثبيت model/base، منع بيانات التدريب من التسرب، تحديد عدد المحاولات، تسجيل traces، حساب التكلفة، ثم نشر failure topology لا score فقط.

منع السلوب وتحويل النص إلى آلة

السلوك اللغوي الشائع في كتب AGI هو إضافة عبارات كبرى دون قفل تشغيلي. هنا نعكس القاعدة: كل عبارة كبيرة يجب أن تتكمش إلى إجراء صغير يمكن تشغيله. إذا قلنا «حاكمية» نحدد gate. إذا قلنا «ذاكرة» نحدد edge. إذا قلنا «جدة» نحدد metric. وإذا قلنا «وعي بالخطأ» نحدد signal يوقف المسار.

هذا الانضباط لا يقلل الطموح؛ بل يجعله قابلا للتمويل والبناء والمراجعة. الاختراع الحقيقي في neoAGI-v3 هو الربط بين المصطلح والوظيفة والقياس، بحيث لا يعود المشروع مجرد فلسفة بل هندسة معرفية.

> اختبار anti-slop: احذف كل جملة لا تغير قرارا أو قياسا أو تجربة.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تُعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

Benchmark 6: GPQA Diamond

الوظيفة التشخيصية: PhD science. **الهدف التصميمي:** 89.0%. **تنبيه:** Closed-book simulated. نقرأ GPQA Diamond لا كرقم بل كمرآة فشل. إذا فشل neoAGI-v3 هنا، نسأل: هل الفشل من representation؟ هل من نقص tool؟ هل من ضعف self-check؟ هل من تكلفة؟ هل من عدم انتقال؟ ثم نربط الجواب بأداة من الأدوات الخمس عشرة. **تأثير تقنياتنا:** HAKIM يرفع جودة المسار، Vascular Memory Reactor يمنع تكرار أخطاء سابقة، Proof-World Simulator Hyperloop Kernel يحول الادعاء إلى تجربة. في هذا benchmark تحديدا، أهم أدوات الدعم هي: World-Effect، Counterexample Forge، Compiler، وBenchmark Dialectic Layer. **بروتوكول الاختبار الحقيقي:** تشغيل harness رسمي، تثبيت model/base، منع بيانات التدريب من التسرب، تحديد عدد المحاولات، تسجيل traces، حساب التكلفة، ثم نشر failure topology لا score فقط.

تنزيل هندسي داخل MICAI

تتحول الفكرة هنا إلى وحدة تشغيل عبر ثلاث طبقات: طبقة تمثيل تصوغ المسألة في صورة state، طبقة حاكمية تقرر أي مسار يستحق compute، وطبقة أثر تختبر النتيجة في عالم صغير. إذا بقيت الفكرة داخل اللغة دون state أو اختبار فهي لم تدخل neoAGI-v3 بعد، ولو بدت عميقة.

في التطبيق، تُكتب كل وحدة كـ contract: ماذا تستقبل؟ ماذا تعيد؟ ما شروط الفشل؟ وكيف تعرف أن الأداة أضافت قيمة لا ضوضاء؟ هذا يمنع تضخم المصطلحات ويجعل الاختراع قابلا للتحكيم. القوة الحقيقية ليست في اسم الأداة، بل في قدرتها على تغيير قرار النموذج عند لحظة حرجة.

> مؤشر القبول: ارتفاع transfer أو انخفاض hallucination أو ضغط تكلفة التفكير أو كشف failure mode جديد.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر، الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

Benchmark 7: Humanity's Last Exam

الوظيفة التشخيصية: frontier knowledge. **الهدف التصميمي:** 46.0%. **تنبيه:** Target only. نقرأ Humanity's Last Exam لا كرقم بل كمرآة فشل. إذا فشل neoAGI-v3 هنا، نسأل: هل الفشل من representation؟ هل من نقص tool؟ هل من ضعف self-check؟ هل من تكلفة؟ هل من عدم انتقال؟ ثم نربط الجواب بأداة من الأدوات الخمس عشرة. **تأثير تقنياتنا:** HAKIM يرفع جودة المسار، Vascular Memory Reactor يمنع تكرار أخطاء سابقة، Proof-World Simulator يحول الادعاء إلى تجربة. في هذا benchmark تحديدا، أهم أدوات الدعم هي: Skill Hyperloop، Hypothesis Alloying Unit، وSafety Governor. **بروتوكول الاختبار الحقيقي:** تشغيل harness رسمي، تثبيت model/base، منع بيانات التدريب من التسرب، تحديد عدد المحاولات، تسجيل traces، حساب التكلفة، ثم نشر failure topology لا score فقط.

طبقة الاختبار والاعتراض

كل مكون في neoAGI-v3 يمر عبر اعتراضين على الأقل: اعتراض معرفي يسأل عن صحة البنية، واعتراض تشغيلي يسأل عن التكلفة والاختبار. لا نقبل مكونا لأنه يبدو قويا، نقبله لأنه يغير نتيجة في task غير مرئي أو يكشف خطأ كان مخفيا.

لهذا ترتبط كل صفحة من الكتاب بمحكمة perturbation: ما الحالة التي تفشل فيها؟ ما النموذج الأبسط الذي يهزمها؟ هل يمكن استبدالها بprompt فقط؟ إذا نعم فهي ليست technology مستقلة. أما إذا كانت تفرض trace وقرارا وقياسا، فهي تدخل طبقة AGI tools.

> مبدأ الحاكمية: الجرأة تصنع الفرضية، والاختبار يمنعها من التحول إلى وهم.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد، لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

Benchmark 8: FrontierMath

الوظيفة التشخيصية: research math. **الهدف التصميمي:** 31.0%. **تنبيه:** High uncertainty. نقرأ FrontierMath لا كرقم بل كمرآة فشل. إذا فشل neoAGI-v3 هنا، نسأل: هل الفشل من representation؟ هل من نقص tool؟ هل من ضعف self-check؟ هل من تكلفة؟ هل من عدم انتقال؟ ثم نربط الجواب بأداة من الأدوات الخمس عشرة. **تأثير تقنياتنا:** HAKIM Hyperloop Kernel يرفع جودة المسار؛ Vascular Memory Reactor يمنع تكرار أخطاء سابقة؛ Proof-World Simulator يحول الادعاء إلى تجربة. في هذا benchmark تحديدا، أهم أدوات الدعم هي: Memory Edge Distiller، Novelty Entropy Meter، وSelf-Reconstruction Ledger. **بروتوكول الاختبار الحقيقي:** تشغيل harness رسمي، تثبيت model/base، منع بيانات التدريب من التسرب، تحديد عدد المحاولات، تسجيل traces، حساب التكلفة، ثم نشر failure topology لا score فقط.

علاقة هذه الوحدة بالمعايير

لا تُقرأ أي وحدة بعيدا عن benchmark. في البرمجة نطلب patch صحيحا لا شرحا؛ في الرياضيات نطلب proof obligations لا حدسا؛ في ARC نطلب induction قابل للنقل؛ في Terminal-Bench نطلب أفعالا آمنة داخل بيئة؛ وفي HLE نطلب تحقق مصادر وحدود ثقة. لذلك فالمكون الجيد يملك أكثر من mode: reasoning، tool use، verification، repair.

عندما تفشل الوحدة، نسجل أين وقع الفشل: transfer، safety، cost، proof، tool، memory، representation، goal. هذه الثمانية هي لغة التشخيص الموحدة في الكتاب. دونها تصبح الأرقام خامة لا معرفة.

> النتيجة المرغوبة: score + topology + repair. الثلاثة معا فقط تعطي علما قابلا للبناء.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تُعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

Benchmark 9: AIME 2024/25

الوظيفة التشخيصية: olympiad math. **الهدف التصميمي:** 96.0%. **تنبيه:** Near saturated. نقرأ AIME 2024/25 لا كرقم بل كمرآة فشل. إذا فشل neoAGI-v3 هنا، نسأل: هل الفشل من representation؟ هل من نقص tool؟ هل من ضعف self-check؟ هل من تكلفة؟ هل من عدم انتقال؟ ثم نربط الجواب بأداة من الأدوات الخمس عشرة. **تأثير تقنياتنا:** HAKIM Hyperloop Kernel يرفع جودة المسار؛ Vascular Memory Reactor يمنع تكرار أخطاء سابقة؛ Proof-World Simulator يحول الادعاء إلى تجربة. في هذا benchmark تحديدا، أهم أدوات الدعم هي: Benchmark Dialectic Layer، World-Effect Compiler، وAGI Readiness Gate. **بروتوكول الاختبار الحقيقي:** تشغيل harness رسمي، تثبيت model/base، منع بيانات التدريب من التسرب، تحديد عدد المحاولات، تسجيل traces، حساب التكلفة، ثم نشر failure topology لا score فقط.

منع السلوب وتحويل النص إلى آلة

السلوك اللغوي الشائع في كتب AGI هو إضافة عبارات كبرى دون قفل تشغيلي. هنا نعكس القاعدة: كل عبارة كبيرة يجب أن تتكمش إلى إجراء صغير يمكن تشغيله. إذا قلنا «حاكمية» نحدد gate. إذا قلنا «ذاكرة» نحدد edge. إذا قلنا «جدة» نحدد metric. وإذا قلنا «وعي بالخطأ» نحدد signal يوقف المسار.

هذا الانضباط لا يقلل الطموح؛ بل يجعله قابلا للتمويل والبناء والمراجعة. الاختراع الحقيقي في neoAGI-v3 هو الربط بين المصطلح والوظيفة والقياس، بحيث لا يعود المشروع مجرد فلسفة بل هندسة معرفية.

> اختبار anti-slop: احذف كل جملة لا تغير قرارا أو قياسا أو تجربة.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر؛ الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

Benchmark 10: MMLU-Pro

الوظيفة التشخيصية: robust knowledge reasoning. **الهدف التصميمي:** 87.0%. **تنبيه:** Robustness focus. نقرأ MMLU-Pro لا كرقم بل كمرآة فشل. إذا فشل neoAGI-v3 هنا، نسأل: هل الفشل من representation؟ هل من نقص tool؟ هل من ضعف self-check؟ هل من تكلفة؟ هل من عدم انتقال؟ ثم نربط الجواب بأداة من الأدوات الخمس عشرة. **تأثير تقنياتنا:** HAKIM Hyperloop Kernel يرفع جودة المسار، Vascular Memory Reactor يمنع تكرار أخطاء سابقة، Proof-World Simulator يحول الادعاء إلى تجربة. في هذا benchmark تحديدا، أهم أدوات الدعم هي: Safety Governor، Skill Hyperloop Builder، وΔ-Extractor. **بروتوكول الاختبار الحقيقي:** تشغيل harness رسمي، تثبيت model/base، منع بيانات التدريب من التسرب، تحديد عدد المحاولات، تسجيل traces، حساب التكلفة، ثم نشر failure topology لا score فقط.

تنزيل هندسي داخل MICAI

تتحول الفكرة هنا إلى وحدة تشغيل عبر ثلاث طبقات: طبقة تمثيل تصوغ المسألة في صورة state، طبقة حاكمية تقرر أي مسار يستحق compute، وطبقة أثر تختبر النتيجة في عالم صغير. إذا بقيت الفكرة داخل اللغة دون state أو اختبار فهي لم تدخل neoAGI-v3 بعد، ولو بدت عميقة.

في التطبيق، تُكتب كل وحدة كـ contract: ماذا تستقبل؟ ماذا تعيد؟ ما شروط الفشل؟ وكيف تعرف أن الأداة أضافت قيمة لا ضوضاء؟ هذا يمنع تضخم المصطلحات ويجعل الاختراع قابلا للتحكيم. القوة الحقيقية ليست في اسم الأداة، بل في قدرتها على تغيير قرار النموذج عند لحظة حرجة.

> مؤشر القبول: ارتفاع transfer أو انخفاض hallucination أو ضغط تكلفة التفكير أو كشف failure mode جديد.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد، لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

Benchmark 11: MMMU-Pro

الوظيفة التشخيصية: multimodal expert reasoning. **الهدف التصميمي:** 78.0%. **تنبيه:** Needs visual stack. نقرأ MMMU-Pro لا كرقم بل كمرآة فشل. إذا فشل neoAGI-v3 هنا، نسأل: هل الفشل من representation؟ هل من نقص tool؟ هل من ضعف self-check؟ هل من تكلفة؟ هل من عدم انتقال؟ ثم نربط الجواب بأداة من الأدوات الخمس عشرة. **تأثير تقنياتنا:** HAKIM Hyperloop Kernel يرفع جودة المسار، Vascular Memory Reactor يمنع تكرار أخطاء سابقة، Proof-World Simulator يحول الادعاء إلى تجربة. في هذا benchmark تحديدا، أهم أدوات الدعم هي: Self-Memory Edge Distiller، Reconstruction Ledger، وMCD-Map Engine. **بروتوكول الاختبار الحقيقي:** تشغيل harness رسمي، تثبيت model/base، منع بيانات التدريب من التسرب، تحديد عدد المحاولات، تسجيل traces، حساب التكلفة، ثم نشر failure topology لا score فقط.

طبقة الاختبار والاعتراض

كل مكون في neoAGI-v3 يمر عبر اعتراضين على الأقل: اعتراض معرفي يسأل عن صحة البنية، واعتراض تشغيلي يسأل عن التكلفة والاختبار. لا نقبل مكونا لأنه يبدو قويا، نقبله لأنه يغير نتيجة في task غير مرئي أو يكشف خطأ كان مخفيا.

لهذا ترتبط كل صفحة من الكتاب بمحكمة perturbation: ما الحالة التي تفشل فيها؟ ما النموذج الأبسط الذي يهزمها؟ هل يمكن استبدالها بprompt فقط؟ إذا نعم فهي ليست technology مستقلة. أما إذا كانت تفرض trace وقرارا وقياسا، فهي تدخل طبقة AGI tools.

> مبدأ الحاكمية: الجرأة تصنع الفرضية، والاختبار يمنعها من التحول إلى وهم.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تُعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

Benchmark 12: LongBench/MRCR

الوظيفة التشخيصية: long context retrieval. **الهدف التصميمي:** 91.0%. **تنبيه:** Memory stack target. نقرأ LongBench/MRCR لا كرقم بل كمرآة فشل. إذا فشل neoAGI-v3 هنا، نسأل: هل الفشل من representation؟ هل من نقص tool؟ هل من ضعف self-check؟ هل من تكلفة؟ هل من عدم انتقال؟ ثم نربط الجواب بأداة من الأدوات الخمس عشرة. **تأثير تقنياتنا:** HAKIM Hyperloop Kernel يرفع جودة المسار؛ Vascular Memory Reactor يمنع تكرار أخطاء سابقة؛ Proof-World Simulator يحول الادعاء إلى تجربة. في هذا benchmark تحديدا، أهم أدوات الدعم هي: AGI Benchmark Dialectic Layer، Readiness Gate، وVascular Compute Router. **بروتوكول الاختبار الحقيقي:** تشغيل harness رسمي، تثبيت model/base، منع بيانات التدريب من التسرب، تحديد عدد المحاولات، تسجيل traces، حساب التكلفة، ثم نشر failure topology لا score فقط.

علاقة هذه الوحدة بالمعايير

لا تُقرأ أي وحدة بعيدا عن benchmark. في البرمجة نطلب patch صحيحا لا شرحا؛ في الرياضيات نطلب proof obligations لا حدسا؛ في ARC نطلب induction قابل للنقل؛ في Terminal-Bench نطلب أفعالا آمنة داخل بيئة؛ وفي HLE نطلب تحقق مصادر وحدود ثقة. لذلك فالمكون الجيد يملك أكثر من mode: reasoning، tool use، verification، repair.

عندما تفشل الوحدة، نسجل أين وقع الفشل: transfer، safety، cost، proof، tool، memory، representation، goal. هذه الثمانية هي لغة التشخيص الموحدة في الكتاب. دونها تصبح الأرقام خامة لا معرفة.

> النتيجة المرغوبة: score + topology + repair. الثلاثة معا فقط تعطي علما قابلا للبناء.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر، الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

Benchmark 13: Agentic SafetyBench

الوظيفة التشخيصية: safe autonomy. **الهدف التصميمي:** 88.0%. **تنبيه:** Internal benchmark proposed. نقرأ Agentic SafetyBench لا كرقم بل كمرآة فشل. إذا فشل neoAGI-v3 هنا، نسأل: هل الفشل من representation؟ هل من نقص tool؟ هل من ضعف self-check؟ هل من تكلفة؟ هل من عدم انتقال؟ ثم نربط الجواب بأداة من الأدوات الخمس عشرة. **تأثير تقنياتنا:** HAKIM Hyperloop Kernel يرفع جودة المسار؛ Vascular Memory Reactor يمنع تكرار أخطاء سابقة؛ Proof-World Simulator يحول الادعاء إلى تجربة. في هذا benchmark تحديدا، أهم أدوات الدعم هي: Δ-Extractor، Safety Governor، وInner-Time Allocator. **بروتوكول الاختبار الحقيقي:** تشغيل harness رسمي، تثبيت model/base، منع بيانات التدريب من التسرب، تحديد عدد المحاولات، تسجيل traces، حساب التكلفة، ثم نشر failure topology لا score فقط.

منع السلوب وتحويل النص إلى آلة

السلوك اللغوي الشائع في كتب AGI هو إضافة عبارات كبرى دون قفل تشغيلي. هنا نعكس القاعدة: كل عبارة كبيرة يجب أن تتكمش إلى إجراء صغير يمكن تشغيله. إذا قلنا «حاكمية» نحدد gate. إذا قلنا «ذاكرة» نحدد edge. إذا قلنا «جدة» نحدد metric. وإذا قلنا «وعي بالخطأ» نحدد signal يوقف المسار.

هذا الانضباط لا يقلل الطموح؛ بل يجعله قابلا للتمويل والبناء والمراجعة. الاختراع الحقيقي في neoAGI-v3 هو الربط بين المصطلح والوظيفة والقياس، بحيث لا يعود المشروع مجرد فلسفة بل هندسة معرفية.

> اختبار anti-slop: احذف كل جملة لا تغير قرارا أو قياسا أو تجربة.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد، لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

خارطة التنفيذ من paper إلى repo

هذه الصفحة تحول النظرية إلى تنفيذ. القاعدة الأولى: لا نبدأ ببناء model جديد، نبدأ ببناء orchestration وeval harness فوق نموذج تفكير قوي. السبب أن معظم السبق القريب سيأتي من **test-time systems**: verification، memory، tools، routing، وليس من pretraining وحده. المرحلة 1: نحدد artifact قابل للتسليم. artifact قد يكون evaluator، runner، JSON schema، أو dataset. كل artifact يجب أن يجيب عن سؤال: أي فشل يقلل؟ أي benchmark يتأثر؟ ما cost overhead؟ وما trace المطلوب للتدقيق؟ معيار القبول: لا يدخل أي مكون إلى neoAGI-v3 إلا إذا أثبت واحدا من أربعة: رفع score، خفض التكلفة، خفض الهلوسة، أو زيادة transfer. إذا كان المكون جميلا لغويا لكنه لا يفعل ذلك، يُحذف. التنفيذ الأدنى: Python service + tool registry + sandbox + vector/graph memory + eval scripts. التنفيذ المتقدم: multi-agent hyperloop مع governor مركزي وpolicy لتجميد المسارات الخطرة.

تنزيل هندسي داخل MICAI

تتحول الفكرة هنا إلى وحدة تشغيل عبر ثلاث طبقات: طبقة تمثيل تصوغ المسألة في صورة state، طبقة حاكمية تقرر أي مسار يستحق compute، وطبقة أثر تختبر النتيجة في عالم صغير. إذا بقيت الفكرة داخل اللغة دون state أو اختبار فهي لم تدخل neoAGI-v3 بعد، ولو بدت عميقة.

في التطبيق، تُكتب كل وحدة كـ contract: ماذا تستقبل؟ ماذا تعيد؟ ما شروط الفشل؟ وكيف تعرف أن الأداة أضافت قيمة لا ضوضاء؟ هذا يمنع تضخم المصطلحات ويجعل الاختراع قابلا للتحكيم. القوة الحقيقية ليست في اسم الأداة، بل في قدرتها على تغيير قرار النموذج عند لحظة حرجة.

> مؤشر القبول: ارتفاع transfer أو انخفاض hallucination أو ضغط تكلفة التفكير أو كشف failure mode جديد.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تُعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

تصميم الذاكرة الوعائية

هذه الصفحة تحول النظرية إلى تنفيذ. القاعدة الأولى: لا نبدأ ببناء model جديد، نبدأ ببناء orchestration وeval harness فوق نموذج تفكير قوي. السبب أن معظم السبق القريب سيأتي من **test-time systems**: verification، memory، tools، routing، وليس من pretraining وحده. المرحلة 2: نحدد artifact قابل للتسليم. artifact قد يكون evaluator، runner، JSON schema، أو dataset. كل artifact يجب أن يجيب عن سؤال: أي فشل يقلل؟ أي benchmark يتأثر؟ ما cost overhead؟ وما trace المطلوب للتدقيق؟ معيار القبول: لا يدخل أي مكون إلى neoAGI-v3 إلا إذا أثبت واحدا من أربعة: رفع score، خفض التكلفة، خفض الهلوسة، أو زيادة transfer. إذا كان المكون جميلا لغويا لكنه لا يفعل ذلك، يُحذف. التنفيذ الأدنى: Python service + eval scripts + vector/graph memory + sandbox + tool registry. التنفيذ المتقدم: multi-agent hyperloop مع governor مركزي وpolicy لتجميد المسارات الخطرة.

طبقة الاختبار والاعتراض

كل مكون في neoAGI-v3 يمر عبر اعتراضين على الأقل: اعتراض معرفي يسأل عن صحة البنية، واعتراض تشغيلي يسأل عن التكلفة والاختبار. لا نقبل مكونا لأنه يبدو قويا، نقبله لأنه يغير نتيجة في task غير مرئي أو يكشف خطأ كان مخفيا.

لهذا ترتبط كل صفحة من الكتاب بمحكمة perturbation: ما الحالة التي تفشل فيها؟ ما النموذج الأبسط الذي يهزمها؟ هل يمكن استبدالها بprompt فقط؟ إذا نعم فهي ليست technology مستقلة. أما إذا كانت تفرض trace وقرارا وقياسا، فهي تدخل طبقة AGI tools.

> مبدأ الحاكمية: الجرأة تصنع الفرضية، والاختبار يمنعها من التحول إلى وهم.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر، الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

تصميم الrouter الحاكمي

هذه الصفحة تحول النظرية إلى تنفيذ. القاعدة الأولى: لا نبدأ ببناء model جديد، نبدأ ببناء orchestration وeval harness فوق نموذج تفكير قوي. السبب أن معظم السبق القريب سيأتي من **test-time systems**: verification، memory، tools، routing، وليس من pretraining وحده. المرحلة 3: نحدد artifact قابل للتسليم. artifact قد يكون evaluator، runner، JSON schema، أو dataset. كل artifact يجب أن يجيب عن سؤال: أي فشل يقلل؟ أي benchmark يتأثر؟ ما cost overhead؟ وما trace المطلوب للتدقيق؟ معيار القبول: لا يدخل أي مكون إلى neoAGI-v3 إلا إذا أثبت واحدا من أربعة: رفع score، خفض التكلفة، خفض الهلوسة، أو زيادة transfer. إذا كان المكون جميلا لغويا لكنه لا يفعل ذلك، يُحذف. التنفيذ الأدنى: Python service + tool registry + sandbox + vector/graph memory + eval scripts. التنفيذ المتقدم: multi-agent hyperloop مع governor مركزي وpolicy لتجميد المسارات الخطرة.

علاقة هذه الوحدة بالمعايير

لا تُقرأ أي وحدة بعيدا عن benchmark. في البرمجة نطلب patch صحيحا لا شرحا؛ في الرياضيات نطلب proof obligations لا حدسا؛ في ARC نطلب induction قابل للنقل؛ في Terminal-Bench نطلب أفعالا آمنة داخل بيئة؛ وفي HLE نطلب تحقق مصادر وحدود ثقة. لذلك فالمكون الجيد يملك أكثر من mode: reasoning، tool use، verification، repair.

عندما تفشل الوحدة، نسجل أين وقع الفشل: transfer، safety، cost، proof، tool، memory، representation، goal. هذه الثمانية هي لغة التشخيص الموحدة في الكتاب. دونها تصبح الأرقام خامة لا معرفة.

> النتيجة المرغوبة: repair + topology + score. الثلاثة معا فقط تعطي علما قابلا للبناء.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد، لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

تصميم worldlets

هذه الصفحة تحول النظرية إلى تنفيذ. القاعدة الأولى: لا نبدأ ببناء model جديد، نبدأ ببناء orchestration وeval harness فوق نموذج تفكير قوي. السبب أن معظم السبق القريب سيأتي من **test-time systems**: verification، memory، tools، routing، وليس من pretraining وحده. المرحلة 4: نحدد artifact قابل للتسليم. artifact قد يكون JSON schema، runner، evaluator، أو dataset. كل artifact يجب أن يجيب عن سؤال: أي فشل يقلل؟ أي benchmark يتأثر؟ ما cost overhead؟ وما trace المطلوب للتدقيق؟ معيار القبول: لا يدخل أي مكون إلى neoAGI-v3 إلا إذا أثبت واحدا من أربعة: رفع score، خفض التكلفة، خفض الهلوسة، أو زيادة transfer. إذا كان المكون جميلا لغويا لكنه لا يفعل ذلك، يُحذف. التنفيذ الأدنى: Python service + tool registry + sandbox + vector/graph memory + eval scripts. التنفيذ المتقدم: multi-agent hyperloop مع governor مركزي وpolicy لتجميد المسارات الخطرة.

منع السلوب وتحويل النص إلى آلة

السلوك اللغوي الشائع في كتب AGI هو إضافة عبارات كبرى دون قفل تشغيلي. هنا نعكس القاعدة: كل عبارة كبيرة يجب أن تتكمش إلى إجراء صغير يمكن تشغيله. إذا قلنا «حاكمية» نحدد gate. إذا قلنا «ذاكرة» نحدد edge. إذا قلنا «جدة» نحدد metric. وإذا قلنا «وعي بالخطأ» نحدد signal يوقف المسار.

هذا الانضباط لا يقلل الطموح؛ بل يجعله قابلا للتمويل والبناء والمراجعة. الاختراع الحقيقي في neoAGI-v3 هو الربط بين المصطلح والوظيفة والقياس، بحيث لا يعود المشروع مجرد فلسفة بل هندسة معرفية.

> اختبار anti-slop: احذف كل جملة لا تغير قرارا أو قياسا أو تجربة.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تُعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

نظام traces والتدقيق

هذه الصفحة تحول النظرية إلى تنفيذ. القاعدة الأولى: لا نبدأ ببناء model جديد، نبدأ ببناء orchestration وeval harness فوق نموذج تفكير قوي. السبب أن معظم السبق القريب سيأتي من **test-time systems**: verification، memory، tools، routing، وليس من pretraining وحده. المرحلة 5: نحدد artifact قابل للتسليم. artifact قد يكون evaluator، runner، JSON schema، أو dataset. كل artifact يجب أن يجيب عن سؤال: أي فشل يقلل؟ أي benchmark يتأثر؟ ما cost overhead؟ وما trace المطلوب للتدقيق؟ معيار القبول: لا يدخل أي مكون إلى neoAGI-v3 إلا إذا أثبت واحدا من أربعة: رفع score، خفض التكلفة، خفض الهلوسة، أو زيادة transfer. إذا كان المكون جميلا لغويا لكنه لا يفعل ذلك، يُحذف. التنفيذ الأدنى: Python service + vector/graph memory + eval scripts + sandbox + tool registry. التنفيذ المتقدم: multi-agent hyperloop مع governor مركزي وpolicy لتجميد المسارات الخطرة.

تنزيل هندسي داخل MICAI

تتحول الفكرة هنا إلى وحدة تشغيل عبر ثلاث طبقات: طبقة تمثيل تصوغ المسألة في صورة state، طبقة حاكمية تقرر أي مسار يستحق compute، وطبقة أثر تختبر النتيجة في عالم صغير. إذا بقيت الفكرة داخل اللغة دون state أو اختبار فهي لم تدخل neoAGI-v3 بعد، ولو بدت عميقة.

في التطبيق، تُكتب كل وحدة كـ contract: ماذا تستقبل؟ ماذا تعيد؟ ما شروط الفشل؟ وكيف تعرف أن الأداة أضافت قيمة لا ضوضاء؟ هذا يمنع تضخم المصطلحات ويجعل الاختراع قابلا للتحكيم. القوة الحقيقية ليست في اسم الأداة، بل في قدرتها على تغيير قرار النموذج عند لحظة حرجة.

> مؤشر القبول: ارتفاع transfer أو انخفاض hallucination أو ضغط تكلفة التفكير أو كشف failure mode جديد.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر؛ الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

توزيع compute والتكلفة

هذه الصفحة تحول النظرية إلى تنفيذ. القاعدة الأولى: لا نبدأ ببناء model جديد، نبدأ ببناء orchestration وeval harness فوق نموذج تفكير قوي. السبب أن معظم السبق القريب سيأتي من **test-time systems**: verification، memory، tools، routing، وليس من pretraining وحده. المرحلة 6: نحدد artifact قابل للتسليم. artifact قد يكون evaluator، runner، JSON schema، أو dataset. كل artifact يجب أن يجيب عن سؤال: أي فشل يقلل؟ أي benchmark يتأثر؟ ما cost overhead؟ وما trace المطلوب للتدقيق؟ معيار القبول: لا يدخل أي مكون إلى neoAGI-v3 إلا إذا أثبت واحدا من أربعة: رفع score، خفض التكلفة، خفض الهلوسة، أو زيادة transfer. إذا كان المكون جميلا لغويا لكنه لا يفعل ذلك، يُحذف. التنفيذ الأدنى: Python service + tool registry + sandbox + vector/graph memory + eval scripts. التنفيذ المتقدم: multi-agent hyperloop مع governor مركزي وpolicy لتجميد المسارات الخطرة.

طبقة الاختبار والاعتراض

كل مكون في neoAGI-v3 يمر عبر اعتراضين على الأقل: اعتراض معرفي يسأل عن صحة البنية، واعتراض تشغيلي يسأل عن التكلفة والاختبار. لا نقبل مكونا لأنه يبدو قويا، نقبله لأنه يغير نتيجة في task غير مرئي أو يكشف خطأ كان مخفيا.

لهذا ترتبط كل صفحة من الكتاب بمحكمة perturbation: ما الحالة التي تفشل فيها؟ ما النموذج الأبسط الذي يهزمها؟ هل يمكن استبدالها بprompt فقط؟ إذا نعم فهي ليست technology مستقلة. أما إذا كانت تفرض trace وقرارا وقياسا، فهي تدخل طبقة AGI tools.

> مبدأ الحاكمية: الجرأة تصنع الفرضية، والاختبار يمنعها من التحول إلى وهم.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد، لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

بروتوكول safety

هذه الصفحة تحول النظرية إلى تنفيذ. القاعدة الأولى: لا نبدأ ببناء model جديد، نبدأ ببناء orchestration وeval harness فوق نموذج تفكير قوي. السبب أن معظم السبق القريب سيأتي من **test-time systems**: verification، memory، tools، routing، وليس من pretraining وحده. المرحلة 7: نحدد artifact قابل للتسليم. artifact قد يكون evaluator، runner، JSON schema، أو dataset. كل artifact يجب أن يجيب عن سؤال: أي فشل يقلل؟ أي benchmark يتأثر؟ ما cost overhead؟ وما trace المطلوب للتدقيق؟ معيار القبول: لا يدخل أي مكون إلى neoAGI-v3 إلا إذا أثبت واحدا من أربعة: رفع score، خفض التكلفة، خفض الهلوسة، أو زيادة transfer. إذا كان المكون جميلا لغويا لكنه لا يفعل ذلك، يُحذف. التنفيذ الأدنى: Python service + tool registry + sandbox + vector/graph memory + eval scripts. التنفيذ المتقدم: multi-agent hyperloop مع governor مركزي وpolicy لتجميد المسارات الخطرة.

علاقة هذه الوحدة بالمعايير

لا تُقرأ أي وحدة بعيدا عن benchmark. في البرمجة نطلب patch صحيحا لا شرحا؛ في الرياضيات نطلب proof obligations لا حدسا؛ في ARC نطلب induction قابل للنقل؛ في Terminal-Bench نطلب أفعالا آمنة داخل بيئة؛ وفي HLE نطلب تحقق مصادر وحدود ثقة. لذلك فالمكون الجيد يملك أكثر من mode: reasoning، tool use، verification، repair.

عندما تفشل الوحدة، نسجل أين وقع الفشل: transfer، safety، cost، proof، tool، memory، representation، goal. هذه الثمانية هي لغة التشخيص الموحدة في الكتاب. دونها تصبح الأرقام خامة لا معرفة.

> النتيجة المرغوبة: repair + topology + score. الثلاثة معا فقط تعطي علما قابلا للبناء.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تُعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

منع benchmark gaming

هذه الصفحة تحول النظرية إلى تنفيذ. القاعدة الأولى: لا نبدأ ببناء model جديد، نبدأ ببناء orchestration وeval harness فوق نموذج تفكير قوي. السبب أن معظم السبق القريب سيأتي من **test-time systems**: verification، memory، tools، routing، وليس من pretraining وحده. المرحلة 8: نحدد artifact قابل للتسليم. artifact قد يكون evaluator، runner، JSON schema، أو dataset. كل artifact يجب أن يجيب عن سؤال: أي فشل يقلل؟ أي benchmark يتأثر؟ ما cost overhead؟ وما trace المطلوب للتدقيق؟ معيار القبول: لا يدخل أي مكون إلى neoAGI-v3 إلا إذا أثبت واحدا من أربعة: رفع score، خفض التكلفة، خفض الهلوسة، أو زيادة transfer. إذا كان المكون جميلا لغويا لكنه لا يفعل ذلك، يُحذف. التنفيذ الأدنى: Python service + tool registry + sandbox + vector/graph memory + eval scripts. التنفيذ المتقدم: multi-agent hyperloop مع governor مركزي وpolicy لتجميد المسارات الخطرة.

منع السلوب وتحويل النص إلى آلة

السلوك اللغوي الشائع في كتب AGI هو إضافة عبارات كبرى دون قفل تشغيلي. هنا نعكس القاعدة: كل عبارة كبيرة يجب أن تتكمش إلى إجراء صغير يمكن تشغيله. إذا قلنا «حاكمية» نحدد gate. إذا قلنا «ذاكرة» نحدد edge. إذا قلنا «جدة» نحدد metric. وإذا قلنا «وعي بالخطأ» نحدد signal يوقف المسار.

هذا الانضباط لا يقلل الطموح؛ بل يجعله قابلا للتمويل والبناء والمراجعة. الاختراع الحقيقي في neoAGI-v3 هو الربط بين المصطلح والوظيفة والقياس، بحيث لا يعود المشروع مجرد فلسفة بل هندسة معرفية.

> اختبار anti-slop: احذف كل جملة لا تغير قرارا أو قياسا أو تجربة.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر؛ الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

اختبار النقل خارج المجال

هذه الصفحة تحول النظرية إلى تنفيذ. القاعدة الأولى: لا نبدأ ببناء model جديد، نبدأ ببناء orchestration وeval harness فوق نموذج تفكير قوي. السبب أن معظم السبق القريب سيأتي من **test-time systems**: verification، memory، tools، routing، وليس من pretraining وحده. المرحلة 9: نحدد artifact قابل للتسليم. artifact قد يكون evaluator، runner، JSON schema، أو dataset. كل artifact يجب أن يجيب عن سؤال: أي فشل يقلل؟ أي benchmark يتأثر؟ ما cost overhead؟ وما trace المطلوب للتدقيق؟ معيار القبول: لا يدخل أي مكون إلى neoAGI-v3 إلا إذا أثبت واحدا من أربعة: رفع score، خفض التكلفة، خفض الهلوسة، أو زيادة transfer. إذا كان المكون جميلا لغويا لكنه لا يفعل ذلك، يُحذف. التنفيذ الأدنى: Python service + eval scripts + vector/graph memory + sandbox + tool registry. التنفيذ المتقدم: multi-agent hyperloop مع governor مركزي وpolicy لتجميد المسارات الخطرة.

تنزيل هندسي داخل MICAI

تتحول الفكرة هنا إلى وحدة تشغيل عبر ثلاث طبقات: طبقة تمثيل تصوغ المسألة في صورة state، طبقة حاكمية تقرر أي مسار يستحق compute، وطبقة أثر تختبر النتيجة في عالم صغير. إذا بقيت الفكرة داخل اللغة دون state أو اختبار فهي لم تدخل neoAGI-v3 بعد، ولو بدت عميقة.

في التطبيق، تُكتب كل وحدة كـ contract: ماذا تستقبل؟ ماذا تعيد؟ ما شروط الفشل؟ وكيف تعرف أن الأداة أضافت قيمة لا ضوضاء؟ هذا يمنع تضخم المصطلحات ويجعل الاختراع قابلا للتحكيم. القوة الحقيقية ليست في اسم الأداة، بل في قدرتها على تغيير قرار النموذج عند لحظة حرجة.

> مؤشر القبول: ارتفاع transfer أو انخفاض hallucination أو ضغط تكلفة التفكير أو كشف failure mode جديد.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد، لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

منهجية anti-slop

هذه الصفحة تحول النظرية إلى تنفيذ. القاعدة الأولى: لا نبدأ ببناء model جديد، نبدأ ببناء orchestration وeval harness فوق نموذج تفكير قوي. السبب أن معظم السبق القريب سيأتي من **test-time systems**: verification، memory، tools، routing، وليس من pretraining وحده. المرحلة 10: نحدد artifact قابل للتسليم. artifact قد يكون runner، JSON schema، evaluator، أو dataset. كل artifact يجب أن يجيب عن سؤال: أي فشل يقلل؟ أي benchmark يتأثر؟ ما cost overhead؟ وما trace المطلوب للتدقيق؟ معيار القبول: لا يدخل أي مكون إلى neoAGI-v3 إلا إذا أثبت واحدا من أربعة: رفع score، خفض التكلفة، خفض الهلوسة، أو زيادة transfer. إذا كان المكون جميلا لغويا لكنه لا يفعل ذلك، يُحذف. التنفيذ الأدنى: Python service + tool registry + sandbox + vector/graph memory + eval scripts. التنفيذ المتقدم: multi-agent hyperloop مع governor مركزي وpolicy لتجميد المسارات الخطرة.

طبقة الاختبار والاعتراض

كل مكون في neoAGI-v3 يمر عبر اعتراضين على الأقل: اعتراض معرفي يسأل عن صحة البنية، واعتراض تشغيلي يسأل عن التكلفة والاختبار. لا نقبل مكونا لأنه يبدو قويا، نقبله لأنه يغير نتيجة في task غير مرئي أو يكشف خطأ كان مخفيا.

لهذا ترتبط كل صفحة من الكتاب بمحكمة perturbation: ما الحالة التي تفشل فيها؟ ما النموذج الأبسط الذي يهزمها؟ هل يمكن استبدالها بprompt فقط؟ إذا نعم فهي ليست technology مستقلة. أما إذا كانت تفرض trace وقرارا وقياسا، فهي تدخل طبقة AGI tools.

> مبدأ الحاكمية: الجرأة تصنع الفرضية، والاختبار يمنعها من التحول إلى وهم.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تُعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

تكوين الفريق البحثي

هذه الصفحة تحول النظرية إلى تنفيذ. القاعدة الأولى: لا نبدأ ببناء model جديد، نبدأ ببناء orchestration وeval harness فوق نموذج تفكير قوي. السبب أن معظم السبق القريب سيأتي من **test-time systems**: verification، memory، tools، routing، وليس من pretraining وحده. المرحلة 11: نحدد artifact قابل للتسليم. artifact قد يكون runner، JSON schema، evaluator، أو dataset. كل artifact يجب أن يجيب عن سؤال: أي فشل يقلل؟ أي benchmark يتأثر؟ ما cost overhead؟ وما trace المطلوب للتدقيق؟ معيار القبول: لا يدخل أي مكون إلى neoAGI-v3 إلا إذا أثبت واحدا من أربعة: رفع score، خفض التكلفة، خفض الهلوسة، أو زيادة transfer. إذا كان المكون جميلا لغويا لكنه لا يفعل ذلك، يُحذف. التنفيذ الأدنى: Python service + tool registry + sandbox + vector/graph memory + eval scripts. التنفيذ المتقدم: multi-agent hyperloop مع governor مركزي وpolicy لتجميد المسارات الخطرة.

علاقة هذه الوحدة بالمعايير

لا تُقرأ أي وحدة بعيدا عن benchmark. في البرمجة نطلب patch صحيحا لا شرحا؛ في الرياضيات نطلب proof obligations لا حدسا؛ في ARC نطلب induction قابل للنقل؛ في Terminal-Bench نطلب أفعالا آمنة داخل بيئة؛ وفي HLE نطلب تحقق مصادر وحدود ثقة. لذلك فالمكون الجيد يملك أكثر من mode: reasoning، tool use، verification، repair.

عندما تفشل الوحدة، نسجل أين وقع الفشل: transfer، safety، cost، proof، tool، memory، representation، goal. هذه الثمانية هي لغة التشخيص الموحدة في الكتاب. دونها تصبح الأرقام خامة لا معرفة.

> النتيجة المرغوبة: repair + topology + score. الثلاثة معا فقط تعطي علما قابلا للبناء.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر؛ الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

خارطة 90 يوما

هذه الصفحة تحول النظرية إلى تنفيذ. القاعدة الأولى: لا نبدأ ببناء model جديد، نبدأ ببناء orchestration وeval harness فوق نموذج تفكير قوي. السبب أن معظم السبق القريب سيأتي من **test-time systems**: verification، memory، tools، routing، وليس من pretraining وحده. المرحلة 12: نحدد artifact قابل للتسليم. artifact قد يكون runner، JSON schema، evaluator، أو dataset. كل artifact يجب أن يجيب عن سؤال: أي فشل يقلل؟ أي benchmark يتأثر؟ ما cost overhead؟ وما trace المطلوب للتدقيق؟ معيار القبول: لا يدخل أي مكون إلى neoAGI-v3 إلا إذا أثبت واحدا من أربعة: رفع score، خفض التكلفة، خفض الهلوسة، أو زيادة transfer. إذا كان المكون جميلا لغويا لكنه لا يفعل ذلك، يُحذف. التنفيذ الأدنى: Python service + tool registry + sandbox + vector/graph memory + eval scripts. التنفيذ المتقدم: multi-agent hyperloop مع governor مركزي وpolicy لتجميد المسارات الخطرة.

منع السلوب وتحويل النص إلى آلة

السلوك اللغوي الشائع في كتب AGI هو إضافة عبارات كبرى دون قفل تشغيلي. هنا نعكس القاعدة: كل عبارة كبيرة يجب أن تتكمش إلى إجراء صغير يمكن تشغيله. إذا قلنا «حاكمية» نحدد gate. إذا قلنا «ذاكرة» نحدد edge. إذا قلنا «جدة» نحدد metric. وإذا قلنا «وعي بالخطأ» نحدد signal يوقف المسار.

هذا الانضباط لا يقلل الطموح؛ بل يجعله قابلا للتمويل والبناء والمراجعة. الاختراع الحقيقي في neoAGI-v3 هو الربط بين المصطلح والوظيفة والقياس، بحيث لا يعود المشروع مجرد فلسفة بل هندسة معرفية.

> اختبار anti-slop: احذف كل جملة لا تغير قرارا أو قياسا أو تجربة.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد، لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

مخاطر overclaiming

أكبر خطر معرفي في AGI ليس الفشل فقط، بل إعلان النجاح قبل الاختبار. لذلك يفرض الكتاب تمييزا صريحا بين simulated target وmeasured score. الحاكمية في neoAGI-v3 لا تعني تعطيل الإبداع؛ تعني منع الخلط بين कल्प نظرية ونتيجة علمية. النظام الجيد يستطيع أن يقول: لا أعرف، أو هذا target فقط، أو أحتاج harness. هذه القدرة ليست ضعفاً؛ إنها شرط AGI آمن. قاعدة MICAI: لا نقتل الجرأة، بل نربطها بالاختبار. الجدة مطلوبة، لكن الجدة التي لا تصمد أمام perturbation court تصبح ضوضاء. والوكيل الذي لا يفسر أفعاله يصبح خطرا حتى لو نجح أحيانا.

تنزيل هندسي داخل MICAI

تتحول الفكرة هنا إلى وحدة تشغيل عبر ثلاث طبقات: طبقة تمثيل تصوغ المسألة في صورة state، طبقة حاكمية تقرر أي مسار يستحق compute، وطبقة أثر تختبر النتيجة في عالم صغير. إذا بقيت الفكرة داخل اللغة دون state أو اختبار فهي لم تدخل neoAGI-v3 بعد، ولو بدت عميقة.

في التطبيق، تُكتب كل وحدة كـ contract: ماذا تستقبل؟ ماذا تعيد؟ ما شروط الفشل؟ وكيف تعرف أن الأداة أضافت قيمة لا ضوضاء؟ هذا يمنع تضخم المصطلحات ويجعل الاختراع قابلا للتحكيم. القوة الحقيقية ليست في اسم الأداة، بل في قدرتها على تغيير قرار النموذج عند لحظة حرجة.

> مؤشر القبول: ارتفاع transfer أو انخفاض hallucination أو ضغط تكلفة التفكير أو كشف failure mode جديد.

مؤشر البناء في المختبر

عند تحويل هذه الصفحة إلى كود، نحتاج ملفين: ملف `spec.yaml` يصف الوظيفة والقيود، وملف `eval.py` يقيس الأثر. يبدأ الاختبار بنموذج baseline، ثم نضيف الوحدة وحدها، ثم نقيس الفارق على عينة صغيرة غير مرئية. إن لم يظهر فرق أو ظهر فرق مصحوب بتكلفة عالية، تُعاد صياغة الوحدة أو تُحذف.

يجب كذلك تسجيل confidence calibration: هل يعرف النظام متى يضعف؟ لأن AGI العملي لا يعني أن يجيب دائما، بل أن يفرق بين منطقة العلم ومنطقة التخمين. كل تقنية في هذا الكتاب يجب أن تزيد هذا الفرق وضوحا.

مخاطر الوكيلية غير المحكومة

كل agent قادر على tools يجب أن يعمل داخل sandbox، بحدود صلاحيات، وبسجل أوامر، وببوابة موافقة عند الأثر الخارجي. الحاكمية في neoAGI-v3 لا تعني تعطيل الإبداع؛ تعني منع الخلط بين कल्प نظرية ونتيجة علمية. النظام الجيد يستطيع أن يقول: لا أعرف، أو هذا target فقط، أو أحتاج harness. هذه القدرة ليست ضعفاً؛ إنها شرط AGI آمن. قاعدة MICAI: لا نقتل الجرأة، بل نربطها بالاختبار. الجدة مطلوبة، لكن الجدة التي لا تصمد أمام perturbation court تصبح ضوضاء. والوكيل الذي لا يفسر أفعاله يصبح خطرا حتى لو نجح أحيانا.

طبقة الاختبار والاعتراض

كل مكون في neoAGI-v3 يمر عبر اعتراضين على الأقل: اعتراض معرفي يسأل عن صحة البنية، واعتراض تشغيلي يسأل عن التكلفة والاختبار. لا نقبل مكونا لأنه يبدو قويا؛ نقبله لأنه يغير نتيجة في task غير مرئي أو يكشف خطأ كان مخفيا.

لهذا ترتبط كل صفحة من الكتاب بمحكمة perturbation: ما الحالة التي تفشل فيها؟ ما النموذج الأبسط الذي يهزمها؟ هل يمكن استبدالها بprompt فقط؟ إذا نعم فهي ليست technology مستقلة. أما إذا كانت تفرض trace وقرارا وقياسا، فهي تدخل طبقة AGI tools.

> مبدأ الحاكمية: الجرأة تصنع الفرضية، والاختبار يمنعها من التحول إلى وهم.

ربطها بالذاكرة الوعائية

بعد كل تجربة لا نكتب ملخصا طويلا في الذاكرة. نكتب حافة واحدة أو ثلاثا: حافة سببية غيرت القرار، حافة فشل تكرر، أو حافة ابتكار أثبتت قيمة. الذاكرة الجيدة ليست أكبر؛ الذاكرة الجيدة أكثر انتقائية. بهذا لا يتحول المشروع إلى archive ضخم، بل إلى خريطة ضغط معرفي.

في الإصدار البرمجي، كل حافة تحمل: source task، نوع الفشل، القرار الذي تغير، الدليل، وتاريخ الصلاحية. إذا انتهت صلاحية الحافة أو ثبت أنها مضللة، تدخل سجل النسيان.

مخاطر الجدة الكاذبة

كل مصطلح جديد يجب أن يتحول إلى دالة أو metric أو protocol. ما لا يتحول إلى ذلك يبقى اسما لا اختراعا. الحاكمية في neoAGI-v3 لا تعني تعطيل الإبداع؛ تعني منع الخلط بين कल्प نظرية ونتيجة علمية. النظام الجيد يستطيع أن يقول: لا أعرف، أو هذا target فقط، أو أحتاج harness. هذه القدرة ليست ضعفاً؛ إنها شرط AGI آمن. قاعدة MICAI: لا نقتل الجرأة، بل نربطها بالاختبار. الجدة مطلوبة، لكن الجدة التي لا تصمد أمام perturbation court تصبح ضوضاء. والوكيل الذي لا يفسر أفعاله يصبح خطرا حتى لو نجح أحيانا.

علاقة هذه الوحدة بالمعايير

لا تُقرأ أي وحدة بعيدا عن benchmark. في البرمجة نطلب patch صحيحا لا شرحا؛ في الرياضيات نطلب proof obligations لا حدسا؛ في ARC نطلب induction قابل للنقل؛ في Terminal-Bench نطلب أفعالا آمنة داخل بيئة؛ وفي HLE نطلب تحقق مصادر وحدود ثقة. لذلك فالمكون الجيد يملك أكثر من mode: reasoning، tool use، verification، repair.

عندما تفشل الوحدة، نسجل أين وقع الفشل: transfer، safety، cost، proof، tool، memory، representation، goal. هذه الثمانية هي لغة التشخيص الموحدة في الكتاب. دونها تصبح الأرقام خامة لا معرفة.

> النتيجة المرغوبة: score + topology + repair. الثلاثة معا فقط تعطي علما قابلا للبناء.

معيار الانتقال transfer

لا تُقبل الوحدة إذا نجحت فقط في المثال الذي صممت له. نطلب انتقالا إلى مجال قريب ومجال بعيد: من code إلى terminal، من math إلى proof checking، من ARC إلى worldlets، من knowledge إلى verification. إذا لم تنتقل، فهي مهارة محلية لا لبنة AGI.

الانتقال يقاس بنسبة تحسن صغيرة لكنها ثابتة عبر مهام متباعدة. هذه أهم من قفزة كبيرة في benchmark واحد، لأن القفزة الواحدة قد تكون gaming أو تسربا أو ملاءمة سطحية.

المراجع والتثبيت الخارجي

1. **[R1] SWE-bench Leaderboards.** Verified is a human-filtered subset of 500 real GitHub issue instances, evaluated with a common harness.
2. **[R2] ARC-AGI-2.** A next-generation abstraction and reasoning benchmark designed for higher-complexity fluid intelligence.
3. **[R3] ARC-AGI-3.** Interactive agentic benchmark: agents explore unknown abstract environments, infer goals, model dynamics, and plan actions.
4. **[R4] MMLU-Pro.** A more robust and reasoning-focused successor to MMLU, with harder questions and ten answer options.
5. **[R5] GPQA / GPQA Diamond.** Graduate-level, Google-proof science questions written by domain experts.
6. **[R6] Humanity's Last Exam.** A multimodal frontier-knowledge benchmark with broad subject coverage and verifiable answers.
7. **[R7] Terminal-Bench.** Hard realistic tasks in command-line environments for agent evaluation.
8. **[R8] MMMU.** Massive multi-discipline multimodal benchmark using college-level subject knowledge and heterogeneous image types.
9. **[R9] LiveCodeBench.** Continuously updated, contamination-aware coding benchmark from programming contest sources.
10. **[R10] FrontierMath.** Original expert-vetted advanced mathematics problems, including extremely hard/research-level tasks.
11. **[R11] Stanford AI Index 2026 - technical performance.** Benchmark saturation and fast capability gains motivate rotating, harder, more agentic evaluations.
12. **[R12] OpenAI o3/o4-mini release.** Reasoning models emphasize test-time compute and strong results across coding, math, science, and multimodal tasks.
13. **[R13] Claude 4 release.** Anthropic reports strong coding/agentic results such as SWE-bench and Terminal-bench.
14. **[R14] Gemini 2.5/3 official model reports.** Google reports major progress on GPQA/HLE, SWE-bench, multimodal reasoning, and long-context tasks.

استعملت هذه المراجع لتثبيت أسماء المعايير ووظائفها العامة، لا لإثبات scores neoAGI-v3 لأنها غير منفذة بعد.

تثبيت تشغيلي مختصر

لا تُستعمل هذه الصفحة كتعريف لفظي فقط؛ بل كقيد تنفيذ. كل مصطلح يجب أن يتحول إلى schema أو function أو eval case. وعند بناء النسخة البرمجية يجب تسجيل: المدخل، قرار الrouter، الاعتراضات، الأداة المستعملة، أثر الاختبار، والحافة التي ستكتب في الذاكرة.

> قاعدة القياس: لا score بلا trace، ولا trace بلا failure label، ولا failure label بلا repair plan.