نسخة تقنية معمقة وممتلئة: نظرية الذكاء الصناعي الفتّاح في مشروع زهير مساعف

هذه النسخة ليست تلخيصا قصيرا ولا قائمة أسماء، إنها إعادة بناء تقنية مطوّلة للنظريات والأدوات والمناهج التدريبية داخل مشروع MICAGI / MCAI / HAKIM-MCD / Q-MCD / Hermes-Fugu / neoAGI. الفكرة الحاكمة أن النظام الذكي لا يبدأ من الجواب، بل من سؤال حاكم يكشف فرقا نشطا، ثم يضغطه MCD/Q-MCD، ثم يوزع الموارد عبر أوعية، ثم يفتح محكمة تفنيد، ثم يربط الجواب بأثر عالمي وذاكرة إعادة بناء. لذلك ستقرأ هنا كل نظرية بوصفها جزءا من pipeline تنفيذي، لا بوصفها عنوانا مستقلا.

المراجعة اعتمدت على الكتب والأوراق المرفوعة: كتاب mcagi-v0.2، MCAI/MICAI، NeoAGI4 × MICAGIv4، Q-MCD/HAKIM-MCD، الورقة التأسيسية MICAI/QMCD، QMICAGI-P03، ورقة فضاء السؤال، وقاعدة Hermes-Fugu HyperSkills. تم تحويلها إلى ورقة تقنية هدفها سد ما كان ناقصا في النسخة السابقة: النظريات التي ذُكرت عرضا ستصبح هنا مفصلة، والأدوات التي ظهرت كقوائم ستُشرح كأجزاء تشغيل، ومناهج التدريب ستُربط بالمقاييس والأبلشن والذاكرة.

المعيار الذي يحكم هذه النسخة هو معيار الامتلاء العلمي لا الامتلاء البصري وحده: الصفحة المملوءة ليست ازدحام كلمات، بل كثافة آليات وتمييزات وروابط واختبارات. لذلك صُمّمت الصفحات بهامش صغير وخط Amiri صغير، وبأسلوب ورقة علمية كثيفة. كل فصل يربط النظرية بأربع جهات: المشكلة التي تحلها، موقعها داخل المعمارية، الأداة التي تنفذها، والطريقة التي تختبرها أو تهدمها.

```
الصيغة العامة: Intent → Hidden Question → Δ → MCD/Q-MCD → HAKIM Governance → Vascular Routing → Agents/Tools →
Falsification Court → Fusion → World Effect → Self-Reconstruction Memory → Higher Question
```

الباب الأول: الأساس المعماري للنظام

من سؤال المستخدم إلى السؤال الحاكم

المستخدم لا يقدّم prompt فقط؛ هو يقدّم غموضا وغاية ومخاطر وسياقا ومسكوتا عنه. إذا تعامل النظام مع النص الظاهر وحده فإنّه يجيب على السطح، أما إذا استخرج السؤال الحاكم فإنه يعيد بناء المهمة بوصفها شبكة فروق. لذلك يصبح Hidden Question Miner أول طبقة تقنية: لا يخترع نية من فراغ، بل يفرز ما في الطلب من مقصد، قيد، نقص، خطر، ومؤشر فعل. السؤال الحاكم ليس سؤالا أعمق لغة؛ إنه سؤال يغير route. فإذا بقي السؤال لا يغير الأداة ولا الذاكرة ولا المحكمة، فهو مجرد إعادة صياغة.

المشكلة التقنية التي تعالجها من سؤال المستخدم إلى السؤال الحاكم هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ من سؤال المستخدم إلى السؤال الحاكم تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع من سؤال المستخدم إلى السؤال الحاكم داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب من سؤال المستخدم إلى السؤال الحاكم عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن من سؤال المستخدم إلى السؤال الحاكم حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ من سؤال المستخدم إلى السؤال الحاكم شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود من سؤال المستخدم إلى السؤال الحاكم مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة من سؤال المستخدم إلى السؤال الحاكم بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط؛ بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة من سؤال المستخدم إلى السؤال الحاكم بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. من سؤال المستخدم إلى السؤال الحاكم تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان من سؤال المستخدم إلى السؤال الحاكم من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

الفرق الفاعل Δ بوصفه وحدة العمل

الفرق الفاعل هو الاختلاف الذي يغيّر توقعا أو قرارا أو قدرة على الفعل. كل نظام ذكاء عميق يحتاج إلى تفريق بين معلومة كثيرة وفروق قليلة منتجة. قد يكتب النموذج صفحات لكن لا يفتح إلا فرقا واحدا، وقد يكتب سؤالا قصيرا لكنه يكشف بنية خطأ كاملة. لهذا يصبح Critical Difference Extractor أداة مركزية: يستخرج الفرق الظاهر، الفرق الخفي، الفرق السلبي، وفرق المخاطر. بعد ذلك يدخل الفرق إلى MCD بدلا من أن يتبع النموذج عادة الإسهاب.

المشكلة التقنية التي تعالجها الفرق الفاعل Δ بوصفه وحدة العمل هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ الفرق الفاعل Δ بوصفه وحدة العمل تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع الفرق الفاعل Δ بوصفه وحدة العمل داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب الفرق الفاعل Δ بوصفه وحدة العمل عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن الفرق الفاعل Δ بوصفه وحدة العمل حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ الفرق الفاعل Δ بوصفه وحدة العمل شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint ،delta_score ،question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود الفرق الفاعل Δ بوصفه وحدة العمل مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة الفرق الفاعل Δ بوصفه وحدة العمل بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة الفرق الفاعل Δ بوصفه وحدة العمل بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. الفرق الفاعل Δ بوصفه وحدة العمل تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان الفرق الفاعل Δ بوصفه وحدة العمل من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

MCD ليس مقياس حجم بل مقياس ضغط

MCD يقيس ضغط الفرق لا كمية الكلام. عناصره: شدة الفرق D، كثافة المعلومة I، عمق الزمن الداخلي T، ضغط الوكالة A، أثر العالم P، التماسك السببي C، وضغط الضجيج أو الاختطاف N. هذه الصيغة لا تدّعي أنها قانون نهائي؛ قيمتها أنها تجعل التقدير قابلا للتعلم والتعديل. في النسخة التقنية، كل route أو أداة يجب أن تبرر نفسها عبر أثرها على أحد هذه العناصر.

المشكلة التقنية التي تعالجها MCD ليس مقياس حجم بل مقياس ضغط هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ MCD ليس مقياس حجم بل مقياس ضغط تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع MCD ليس مقياس حجم بل مقياس ضغط داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب MCD ليس مقياس حجم بل مقياس ضغط عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن MCD ليس مقياس حجم بل مقياس ضغط حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ MCD ليس مقياس حجم بل مقياس ضغط شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint ،delta_score ،question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود MCD ليس مقياس حجم بل مقياس ضغط مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة MCD ليس مقياس حجم بل مقياس ضغط بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة MCD ليس مقياس حجم بل مقياس ضغط بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. MCD ليس مقياس حجم بل مقياس ضغط تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان MCD ليس مقياس حجم بل مقياس ضغط من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

Q-MCD كتحويل للسؤال إلى جهاز قياس

حين يدخل السؤال في MCD لا يبقى القياس بعديا، بل يصبح مولدا للمسارات. Q-MCD يقيس كثافة السؤال، وثوقيته، قيمة الأثر، درجة الغموض، وحاجته إلى محكمة. لذلك يقرر هل يكفي جواب خفيف، أم يلزم GraphRAG، أم يلزم verifier، أم يلزم tool sandbox. هذه النقطة هي الفرق بين prompt engineering وبين معمارية حاكمية: السؤال لا يجمّل الطلب بل يحدد سياسة التنفيذ.

المشكلة التقنية التي تعالجها Q-MCD كتحويل للسؤال إلى جهاز قياس هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ Q-MCD كتحويل للسؤال إلى جهاز قياس تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع Q-MCD كتحويل للسؤال إلى جهاز قياس داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب Q-MCD كتحويل للسؤال إلى جهاز قياس عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن Q-MCD كتحويل للسؤال إلى جهاز قياس حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ Q-MCD كتحويل للسؤال إلى جهاز قياس شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود Q-MCD كتحويل للسؤال إلى جهاز قياس مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة Q-MCD كتحويل للسؤال إلى جهاز قياس بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة Q-MCD كتحويل للسؤال إلى جهاز قياس بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. Q-MCD كتحويل للسؤال إلى جهاز قياس تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان Q-MCD كتحويل للسؤال إلى جهاز قياس من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

HAKIM-MCD كحاكمية تشغيلية

HAKIM-MCD هو طبقة governance تستخدم السؤال والفرق والمقياس لاختيار السلوك. يمكن قراءته كبروتوكول: Hypothesis, Audit, Knowledge, Intervention, Metrics. هو لا يحل محل النموذج، بل يقرر متى يثق بالنموذج، ومتى يطلب أداة، ومتى يستعمل محكمة، ومتى يوقف التنفيذ. بذلك يصبح الذكاء قدرة على حكم شروط إنتاج الجواب لا إنتاج الجواب فقط.

المشكلة التقنية التي تعالجها HAKIM-MCD كحاكمية تشغيلية هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ HAKIM-MCD كحاكمية تشغيلية تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها، فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع HAKIM-MCD كحاكمية تشغيلية داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب HAKIM-MCD كحاكمية تشغيلية عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن HAKIM-MCD كحاكمية تشغيلية حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ HAKIM-MCD كحاكمية تشغيلية شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود HAKIM-MCD كحاكمية تشغيلية مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة HAKIM-MCD كحاكمية تشغيلية بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة HAKIM-MCD كحاكمية تشغيلية بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. HAKIM-MCD كحاكمية تشغيلية تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان HAKIM-MCD كحاكمية تشغيلية من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

من MICAI إلى neoAGI إلى Hermes-Fugu

MICAI/MCAI هي المظلة المعرفية، MICAGI هي النسخة الموجهة نحو AGI بالسؤال والفرق، neoAGI هي الهندسة التطبيقية للوكيل الحاكم، وHermes-Fugu هو نظام orchestration يجعل هذه النظريات تعمل في واجهة واحدة متعددة النماذج. إذا لم تُرتب هذه الأسماء كطبقات، تتحول إلى تضخم اصطلاحي؛ أما إذا ربطناها بالوظائف تصبح شجرة معمارية واضحة.

المشكلة التقنية التي تعالجها من MICAI إلى neoAGI إلى Hermes-Fugu هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ من MICAI إلى neoAGI إلى Hermes-Fugu تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها، فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع من MICAI إلى neoAGI إلى Hermes-Fugu داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب من MICAI إلى neoAGI إلى Hermes-Fugu عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن من MICAI إلى neoAGI إلى Hermes-Fugu حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ من MICAI إلى neoAGI إلى Hermes-Fugu شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: question_rank، delta_score، route_constraint، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود من MICAI إلى neoAGI إلى Hermes-Fugu مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة من MICAI إلى neoAGI إلى Hermes-Fugu بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط؛ بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة من MICAI إلى neoAGI إلى Hermes-Fugu بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. من MICAI إلى neoAGI إلى Hermes-Fugu تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان من MICAI إلى neoAGI إلى Hermes-Fugu من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

الباب الثاني: النظريات التقنية الموسعة

نظرية فضاء السؤال التوليدي الوثوقي

تحول السؤال إلى فضاء دينامي لا جملة مفردة. وظيفتها أن تمنع النظام من الانتقال المباشر إلى الجواب، وأن تجبره على تحديد الطبقة: صيغة، مقصد، تفكيك، استرجاع/اختبار، حاكمية، أثر عالمي. عندما تكون المهمة بسيطة يكفي سؤال ظاهر، وعندما تكون عالية المخاطر يجب فتح فضاء أسئلة متنافسة. تقنيا، هذه النظرية تنفذ عبر Hidden Question Miner وQuestion Graph وQuestion Collapse Rule. تقاس بقدرتها على تخفيض سوء فهم المستخدم وتحسين اختيار الأداة.

المشكلة التقنية التي تعالجها نظرية فضاء السؤال التوليدي الوثوقي هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ نظرية فضاء السؤال التوليدي الوثوقي تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع نظرية فضاء السؤال التوليدي الوثوقي داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب نظرية فضاء السؤال التوليدي الوثوقي عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن نظرية فضاء السؤال التوليدي الوثوقي حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ نظرية فضاء السؤال التوليدي الوثوقي شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint ،delta_score ،question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود نظرية فضاء السؤال التوليدي الوثوقي مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة نظرية فضاء السؤال التوليدي الوثوقي بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط؛ بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة نظرية فضاء السؤال التوليدي الوثوقي بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. نظرية فضاء السؤال التوليدي الوثوقي تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان نظرية فضاء السؤال التوليدي الوثوقي من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

نظرية الكثافة السؤالية

ليست الكثافة السؤالية طول السؤال بل كمية الفروق الفاعلة التي يفتحها السؤال بالنسبة إلى ضجيجه. السؤال الكثيف يرفع QG دون أن يرفع N. في البحث، السؤال الكثيف يحدد المتغير الناقص والمصدر المطلوب والمخاطر، وفي الوكيل يحدد route. تقنيا، تقاس الكثافة بمزيج: عدد الفروق الصالحة، عدد الفرضيات المميزة، قابلية الاختبار، ودقة إحالة المصادر. فإذا زادت الكلمات وانخفض الاختبار، فهذه ليست كثافة بل تضخم.

المشكلة التقنية التي تعالجها نظرية الكثافة السؤالية هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ نظرية الكثافة السؤالية تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع نظرية الكثافة السؤالية داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب نظرية الكثافة السؤالية عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن نظرية الكثافة السؤالية حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ نظرية الكُلفة السؤالية شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: court_requirement، route_constraint، delta_score، question_rank، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود نظرية الكُلفة السؤالية مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة نظرية الكُلفة السؤالية بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة نظرية الكُلفة السؤالية بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. نظرية الكُلفة السؤالية تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان نظرية الكُلفة السؤالية من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

نظرية طبقات السؤال الحاكمي

تمنع الخلط بين السؤال اللغوي والسؤال المعرفي. L0 صيغة، L1 مقصد، L2 تفكيك، L3 استرجاع أو اختبار، L4 حاكمية، L5 أثر عالمي. الطبقة لا تعني تعقيدا لفظيا، تعني وظيفة. سؤال L4 يحدد من يملك حق الحكم: المصدر، الذاكرة، الأداة، أم المحكمة. وسؤال L5 يسأل: ماذا سيتغير في العالم أو المهمة؟ الأدوات المرتبطة بها هي Question Layer Tagger وDepth-Q Estimator.

المشكلة التقنية التي تعالجها نظرية طبقات السؤال الحاكمي هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ نظرية طبقات السؤال الحاكمي تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع نظرية طبقات السؤال الحاكمي داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب نظرية طبقات السؤال الحاكمي عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن نظرية طبقات السؤال الحاكمي حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ نظرية طبقات السؤال الحاكمي شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود نظرية طبقات السؤال الحاكمي مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة نظرية طبقات السؤال الحاكمي بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة نظرية طبقات السؤال الحاكمي بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. نظرية طبقات السؤال الحاكمي تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان نظرية طبقات السؤال الحاكمي من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

نظرية التراكب الوثوقي للسؤالات

قبل الحسم، يوجد السؤال في تراكب قراءات. قد يكون المستخدم يطلب تعريفا، أو نقدا، أو تنفيذا، أو حماية من خطأ. اختيار قراءة واحدة مبكرا ينتج جوابا واثقا خطأ. لذلك يولد النظام عدة أسئلة مرشحة، يزنها بالوثوق والدليل والكلفة، ثم يختار أو يدمج أو يعلق. هذه النظرية ترتبط بالتفكير الشجري لكنها تضيف معيار السؤال، لا مجرد تعدد أجوبة.

المشكلة التقنية التي تعالجها نظرية التراكب الوثوقي للسؤالات هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ نظرية التراكب الوثوقي للسؤالات تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع نظرية التراكب الوثوقي للسؤالات داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب نظرية التراكب الوثوقي للسؤالات عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن نظرية التراكب الوثوقي للسؤالات حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ نظرية التراكب الوثوقي للسؤالات شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود نظرية التراكب الوثوقي للسؤالات مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة نظرية التراكب الوثوقي للسؤالات بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة نظرية التراكب الوثوقي للسؤالات بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. نظرية التراكب الوثوقي للسؤالات تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان نظرية التراكب الوثوقي للسؤالات من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

نظرية الانهيار الاختباري للسؤال

كل سؤال عالي المخاطر يجب أن يحمل شروط سقوطه: متى نعيد صياغته؟ متى نخفضه؟ ما التجربة التي تكشف خطأه؟ الانهيار هنا ليس فشلا؛ إنه انتقال من تراكب إلى قرار. تقنيا، يعمل Question Collapse Rule مع Falsification Court: إذا لم يفتح السؤال فرقا جديدا أو إذا أظهر verifier أنه يقود إلى مصدر ضعيف، ينهار إلى سؤال أدق أو يخرج من المسار.

المشكلة التقنية التي تعالجها نظرية الانهيار الاختباري للسؤال هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ نظرية الانهيار الاختباري للسؤال تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع نظرية الانهيار الاختباري للسؤال داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب نظرية الانهيار الاختباري للسؤال عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن نظرية الانهيار الاختباري للسؤال حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ نظرية الانهيار الاختباري للسؤال شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود نظرية الانهيار الاختباري للسؤال مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة نظرية الانهيار الاختباري للسؤال بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة نظرية الانهيار الاختباري للسؤال بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. نظرية الانهيار الاختباري للسؤال تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان نظرية الانهيار الاختباري للسؤال من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

نظرية التوجيه الوعائي للسؤال

السؤال لا يطلب إجابة فقط، يوجه الذاكرة والأدوات والنماذج والمصادر. التوجيه الوعائي يوزع compute على أوعية: verifier، simulation، code tool، retrieval، language reasoning، memory. إذا كان الفرق سببيًا يذهب إلى GraphWorld؛ إذا كان مصدريا يذهب إلى GraphRAG؛ إذا كان تنفيذيا يذهب إلى sandbox؛ إذا كان خطرًا يذهب إلى privacy gate. هذه هي نقطة التحول من prompt إلى router.

المشكلة التقنية التي تعالجها نظرية التوجيه الوعائي للسؤال هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ نظرية التوجيه الوعائي للسؤال تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع نظرية التوجيه الوعائي للسؤال داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب نظرية التوجيه الوعائي للسؤال عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن نظرية التوجيه الوعائي للسؤال حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ نظرية التوجيه الوعائي للسؤال شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود نظرية التوجيه الوعائي للسؤال مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة نظرية التوجيه الوعائي للسؤال بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة نظرية التوجيه الوعائي للسؤال بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. نظرية التوجيه الوعائي للسؤال تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان نظرية التوجيه الوعائي للسؤال من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

نظرية حلقة أثر العالم للسؤال

السؤال لا يكتمل إلا إذا أنتج أثرا يمكن رصده: قرار، اختبار، artifact، تحديث ذاكرة، أو سؤال أعلى. World-Effect Loop تمنع الانغلاق النصي: الجواب الجيد ليس ما يبدو متماسكا فقط، بل ما يحقق closure بين intended_world_effect وverified_world_effect. تقنيا، ترتبط هذه النظرية بمقياس HyperloopClosure وبـWorld-Effect Monitor.

المشكلة التقنية التي تعالجها نظرية حلقة أثر العالم للسؤال هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ نظرية حلقة أثر العالم للسؤال تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع نظرية حلقة أثر العالم للسؤال داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب نظرية حلقة أثر العالم للسؤال عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن نظرية حلقة أثر العالم للسؤال حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ نظرية حلقة أثر العالم للسؤال شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود نظرية حلقة أثر العالم للسؤال مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة نظرية حلقة أثر العالم للسؤال بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة نظرية حلقة أثر العالم للسؤال بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. نظرية حلقة أثر العالم للسؤال تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان نظرية حلقة أثر العالم للسؤال من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

نظرية التعميم الوعائي المتمايز VDG

التعميم الحقيقي ليس نقل عبارة إلى مجال آخر، بل نقل نمط فرق عبر أوعية جديدة. VDG تقول إن الوكيل يتعلم حين يميز ما يبقى ثابتا عبر الأوعية وما يتغير. فإذا حل مشكلة برمجية، لا ينقل الكود، بل ينقل نمط استخراج الفرق، اختيار الأداة، التحقق، وكتابة الذاكرة. تقاس النظرية عبر transfer few-shot وليس عبر إعادة استعمال قالب لغوي.

المشكلة التقنية التي تعالجها نظرية التعميم الوعائي المتمايز VDG هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ نظرية التعميم الوعائي المتمايز VDG تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع نظرية التعميم الوعائي المتمايز VDG داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب نظرية التعميم الوعائي المتمايز VDG عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن نظرية التعميم الوعائي المتمايز VDG حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ نظرية التعميم الوعائي المتمايز VDG شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود نظرية التعميم الوعائي المتمايز VDG مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة نظرية التعميم الوعائي المتمايز VDG بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة نظرية التعميم الوعائي المتمايز VDG بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. نظرية التعميم الوعائي المتمايز VDG تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان نظرية التعميم الوعائي المتمايز VDG من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

نظرية الزمن الداخلي السببي ITC

الزمن الداخلي ليس زمن الساعة بل عمق المسار المعرفي: كم دورة سؤال-فرق-تفنيد-ذاكرة يحتاجها النظام قبل أن يحق له الجواب؟ ITC تمنع الاستعجال في المهام المركبة، وتمنع الإسهاب في المهام البسيطة. تقنيا، Inner-Time Budgeter يحدد ميزانية التفكير بناء على MCD وrisk وworld-effect pressure، ويوقف المسار حين لا يضيف التوسع فرقا قابلا للقياس.

المشكلة التقنية التي تعالجها نظرية الزمن الداخلي السببي ITC هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ نظرية الزمن الداخلي السببي ITC تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع نظرية الزمن الداخلي السببي ITC داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب نظرية الزمن الداخلي السببي ITC عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن نظرية الزمن الداخلي السببي ITC حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ نظرية الزمن الداخلي السببي ITC شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود نظرية الزمن الداخلي السببي ITC مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة نظرية الزمن الداخلي السببي ITC بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط؛ بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة نظرية الزمن الداخلي السببي ITC بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. نظرية الزمن الداخلي السببي ITC تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان نظرية الزمن الداخلي السببي ITC من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

نظرية المناعة الإبستمولوجية EIG

EIG تجعل النظام يقاوم مصادر ملوثة وذاكرة خاطئة وprompt injection وثقة لغوية. المناعة ليست رفضا عاما؛ إنها فرز طبقي: contradiction، evidence freshness، source rank، tool permission، privacy tier، density. يعمل Epistemic Immune Filter قبل الروتر وبعد المحكمة. تقاس النظرية بقدرتها على خفض الاعتماد على مصدر واحد غير موثوق، لا بمجرد كثرة التحذيرات.

المشكلة التقنية التي تعالجها نظرية المناعة الإبستمولوجية EIG هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ نظرية المناعة الإبستمولوجية EIG تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع نظرية المناعة الإبستمولوجية EIG داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب نظرية المناعة الإبستمولوجية EIG عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن نظرية المناعة الإبستمولوجية EIG حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ نظرية المناعة الإبستمولوجية EIG شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود نظرية المناعة الإبستمولوجية EIG مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة نظرية المناعة الإبستمولوجية EIG بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط؛ بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة نظرية المناعة الإبستمولوجية EIG بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. نظرية المناعة الإبستمولوجية EIG تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان نظرية المناعة الإبستمولوجية EIG من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

نظرية التراكب الوعائي

التراكب الوعائي يعني أن المسألة قد تحتاج عدة أوعية في وقت واحد: ذاكرة، دليل، أداة، محاكاة، نقد. الخلط هو دمج بلا معيار؛ أما التراكب فهو إبقاء المسارات حية حتى يحدد MCD/HAKIM أيها أحق. تقنيا، يفتح Candidate Path Generator مسارات، ثم يقرأ Vascular Router ضغط كل وعاء. هذا يمنع حصر الذكاء في نموذج لغوي واحد.

المشكلة التقنية التي تعالجها نظرية التراكب الوعائي هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ نظرية التراكب الوعائي تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع نظرية التراكب الوعائي داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب نظرية التراكب الوعائي عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن نظرية التراكب الوعائي حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ نظرية التراكب الوعائي شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود نظرية التراكب الوعائي مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة نظرية التراكب الوعائي بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة نظرية التراكب الوعائي بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. نظرية التراكب الوعائي تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان نظرية التراكب الوعائي من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

نظرية الاختطاف الوعائي

الاختطاف الوعائي يحدث عندما يصعد وعاء جزئي إلى مقام الحاكمية العامة: ذاكرة قديمة تقود الحكم، مصدر ملوث يفرض السرد، نموذج قوي يفرض ثقته، أو أداة تغري بالحل السريع. لذلك يفصل المشروع بين capability وauthority. Anti-Hijack Ledger يسجل أين حدث الاختطاف، ومؤشر N في MCD يعاقب المسار الملوث.

المشكلة التقنية التي تعالجها نظرية الاختطاف الوعائي هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ نظرية الاختطاف الوعائي تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع نظرية الاختطاف الوعائي داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب نظرية الاختطاف الوعائي عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن نظرية الاختطاف الوعائي حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ نظرية الاختطاف الوعائي شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود نظرية الاختطاف الوعائي مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة نظرية الاختطاف الوعائي بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة نظرية الاختطاف الوعائي بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. نظرية الاختطاف الوعائي تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان نظرية الاختطاف الوعائي من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

نظرية الحاكمية الإبستمولوجية

الحاكمية الإبستمولوجية هي قدرة النظام على تحديد من يملك حق الحكم في لحظة معينة: الدليل أم الذاكرة أم الأداة أم المحكمة أم الإنسان. هي ليست طبقة أخلاقية عامة فقط، بل kernel قرار. كل claim يحمل rank، وكل مسار يحمل reason، وكل أداة تحمل permission. بهذا يصبح الجواب نتيجة نظام صلاحيات معرفية.

المشكلة التقنية التي تعالجها نظرية الحاكمية الإبستمولوجية هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ نظرية الحاكمية الإبستمولوجية تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع نظرية الحاكمية الإبستمولوجية داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب نظرية الحاكمية الإبستمولوجية عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن نظرية الحاكمية الإبستمولوجية حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ نظرية الحاكمية الإبستمولوجية شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود نظرية الحاكمية الإبستمولوجية مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة نظرية الحاكمية الإبستمولوجية بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط؛ بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة نظرية الحاكمية الإبستمولوجية بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. نظرية الحاكمية الإبستمولوجية تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان نظرية الحاكمية الإبستمولوجية من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

نظرية الحاكمية فوق الحاكمية

حين تفسد طبقة governance نفسها، تحتاج إلى meta-governance. هذه النظرية تراقب تغير المعايير مع الزمن: هل صار الروتر يميل إلى نموذج معين؟ هل صارت المحكمة متساهلة؟ هل صارت الذاكرة تعيد خطأها؟ لذلك تبنى Governance Audit Loop يقيس drift في route decisions وclaim ranks.

المشكلة التقنية التي تعالجها نظرية الحاكمية فوق الحاكمية هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ نظرية الحاكمية فوق الحاكمية تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع نظرية الحاكمية فوق الحاكمية داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب نظرية الحاكمية فوق الحاكمية عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن نظرية الحاكمية فوق الحاكمية حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ نظرية الحاكمية فوق الحاكمية شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود نظرية الحاكمية فوق الحاكمية مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة نظرية الحاكمية فوق الحاكمية بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط؛ بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة نظرية الحاكمية فوق الحاكمية بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. نظرية الحاكمية فوق الحاكمية تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان نظرية الحاكمية فوق الحاكمية من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

نظرية الذاكرة السببية

الذاكرة لا تحفظ النص فقط، بل سبب القرار، الدليل، الاعتراض، الأثر، والفشل. إذا قال النظام: هذه الإجابة نجحت، يجب أن يعرف لماذا نجحت وفي أي شرط لا تنجح. Self-Reconstruction Memory تبني route memory من traces، وتحول الفشل إلى تحديث قابل للاستعمال. الذاكرة السببية تمنع تكرار الخطأ لأنها تحفظ البنية لا العبارة.

المشكلة التقنية التي تعالجها نظرية الذاكرة السببية هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ نظرية الذاكرة السببية تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع نظرية الذاكرة السببية داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب نظرية الذاكرة السببية عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن نظرية الذاكرة السببية حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ نظرية الذاكرة السببية شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: court_requirement، route_constraint، delta_score، question_rank، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود نظرية الذاكرة السببية مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة نظرية الذاكرة السببية بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة نظرية الذاكرة السببية بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. نظرية الذاكرة السببية تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان نظرية الذاكرة السببية من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

نظرية الانتروبيا الابتكارية

الابتكار ليس غرابة لغوية؛ هو تنوع صالح تحت معيار. Novelty Entropy Meter يميز بين novelty productive وnoise. إذا ولد النظام عشر فرضيات كلها غير قابلة للاختبار، فهذا ضجيج. وإذا ولد فرضيتين تفصلان مساراً جديدا وتفتحان تجربة، فهذا ابتكار. ترتبط هذه النظرية بـMCD لأن الجدة لا قيمة لها دون فرق وأثر.

المشكلة التقنية التي تعالجها نظرية الانتروبيا الابتكارية هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ نظرية الانتروبيا الابتكارية تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع نظرية الانتروبيا الابتكارية داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب نظرية الانتروبيا الابتكارية عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن نظرية الانتروبيا الابتكارية حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ نظرية الانتروبيا الابتكارية شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود نظرية الانتروبيا الابتكارية مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة نظرية الانتروبيا الابتكارية بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة نظرية الانتروبيا الابتكارية بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. نظرية الانتروبيا الابتكارية تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان نظرية الانتروبيا الابتكارية من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

نظرية الانتقائية المساعفية

Selective Rescue لا يعني إنقاذ كل مسار، بل إنقاذ المسار الذي يملك إمكانية أثر رغم فشل أولي. عندما تفشل أداة أو مصدر، لا يكرر النظام نفس المسار، بل يسأل: هل الفشل كاشف أم قاتل؟ إذا كان كاشفا، يعاد بناء السؤال أو route. إذا كان قاتلا، يغلق المسار. هذه النظرية ضرورية لتجنب إسراف compute.

المشكلة التقنية التي تعالجها نظرية الانتقائية المساعفية هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ نظرية الانتقائية المساعفية تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها، فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع نظرية الانتقائية المساعفية داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب نظرية الانتقائية المساعفية عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن نظرية الانتقائية المساعفية حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ نظرية الانتقائية المساعفية شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود نظرية الانتقائية المساعفية مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة نظرية الانتقائية المساعفية بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة نظرية الانتقائية المساعفية بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. نظرية الانتقائية المساعفية تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان نظرية الانتقائية المساعفية من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

نظرية VCGE-M

Moussaif Vascular-Causal Governance Engine هو نموذج يدمج الوعائية والسببية والحاكمية. طبقاته: agency، court، vascular routing، causal graph، difference intake، selection، memory reconstruction. ما يضيفه فوق وكيل عادي أنه لا يختار الفعل من نص، بل من causal-vessel map. يتفوق عندما تكون المهمة متعددة المصادر والأدوات والمخاطر.

المشكلة التقنية التي تعالجها نظرية VCGE-M هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ نظرية VCGE-M تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها، فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع نظرية VCGE-M داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب نظرية VCGE-M عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن نظرية VCGE-M حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ نظرية VCGE-M شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: court_requirement، route_constraint، delta_score، question_rank، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود نظرية VCGE-M مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة نظرية VCGE-M بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة نظرية VCGE-M بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. نظرية VCGE-M تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان نظرية VCGE-M من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

نظرية HyperSkill

HyperSkill ليس مهارة عامة بل object تشغيلي له state ومدخلات ومخرجات ومقاييس وفشل وroute consequence. المهارة لا تصبح HyperSkill حتى تحمل evidence contract وverifier rule وmemory artifact. مثلا: Domain Vocabulary Injection ليست prompt فقط، هي تحويل حالة + دليل + معيار فشل + أثر في route.

المشكلة التقنية التي تعالجها نظرية HyperSkill هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ نظرية HyperSkill تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع نظرية HyperSkill داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب نظرية HyperSkill عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن نظرية HyperSkill حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ نظرية HyperSkill شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: court_requirement، route_constraint، delta_score، question_rank، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود نظرية HyperSkill مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة نظرية HyperSkill بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط؛ بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة نظرية HyperSkill بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. نظرية HyperSkill تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان نظرية HyperSkill من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

نظرية Hyperloop

Hyperloop هو دورة مغلقة تتكرر حتى تحقق closure أو تنتهي الميزانية أو يتدخل الإنسان. ليس loop بلا نهاية؛ له stop condition وLong-Session، Causal Novelty Loop. metric أمثلة: Identity Loop، Failure-to-Memory Loop. HyperloopClosure يقيس نسبة الأثر المتحقق إلى الأثر المقصود.

المشكلة التقنية التي تعالجها نظرية Hyperloop هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ نظرية Hyperloop تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع نظرية Hyperloop داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب نظرية Hyperloop عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن نظرية Hyperloop حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ نظرية Hyperloop شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: court_requirement، route_constraint، delta_score، question_rank، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود نظرية Hyperloop مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة نظرية Hyperloop بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط؛ بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة نظرية Hyperloop بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. نظرية Hyperloop تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان نظرية Hyperloop من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

نظرية Hermes-Fugu Orchestration

Hermes-Fugu تجعل واجهة واحدة تخفي نظاما متعدد الأدوار. المستخدم يرى endpoint واحدا، لكن داخله router وplanner وworkers وverifier وcritic وfusion وmemory. الإضافة ليست multi-agent chat بل governed orchestration: كل عامل له صلاحية وحدود وسجل، والروتر يتعلم من التاريخ دون أن يفقد قواعد الخصوصية.

المشكلة التقنية التي تعالجها نظرية Hermes-Fugu Orchestration هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ نظرية Hermes-Fugu Orchestration تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع نظرية Hermes-Fugu Orchestration داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب نظرية Hermes-Fugu Orchestration عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن نظرية Hermes-Fugu Orchestration حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ نظرية Hermes-Fugu Orchestration شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود نظرية Hermes-Fugu Orchestration مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة نظرية Hermes-Fugu Orchestration بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط؛ بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة نظرية Hermes-Fugu Orchestration بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. نظرية Hermes-Fugu Orchestration تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان نظرية Hermes-Fugu Orchestration من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

نظرية السيادة المزودية Provider Sovereignty

لا يجب أن تكون المعمارية رهينة نموذج أو مزود. model names بيانات لا architecture. لذلك تفصل Hermes provider adapter عن OpenRouter .route policy يصلح للاستكشاف، NIM/self-hosting للسيادة والحساسية، والنماذج المحلية للخصوصية. Route memory يجب أن تحفظ provider health ولا تحول تاريخا قديما إلى ثقة مطلقة.

المشكلة التقنية التي تعالجها نظرية السيادة المزودية Provider Sovereignty هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ نظرية السيادة المزودية Provider Sovereignty تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع نظرية السيادة المزودية Provider Sovereignty داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب نظرية السيادة المزودية Provider Sovereignty عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن نظرية السيادة المزودية Provider Sovereignty حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ نظرية السيادة المزودية Provider Sovereignty شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود نظرية السيادة المزودية Provider Sovereignty مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة نظرية السيادة المزودية Provider Sovereignty بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط؛ بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة نظرية السيادة المزودية Provider Sovereignty بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. نظرية السيادة المزودية Provider Sovereignty تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان نظرية السيادة المزودية Provider Sovereignty من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

الباب الثالث: الأدوات التقنية كعناصر تشغيل

Hidden Question Miner

يستخرج السؤال الخفي من intent ويقسمه إلى مقصد، قيد، نقص، خطر، وأثر متوقع. لا يسمح له بإضافة نية غير موجودة؛ لذلك يستخدم دليلين: textual evidence وtask-family prior. إذا لم يجد سؤالا خفيا يعلن عدم اليقين بدلا من التلفيق.

تعمل أداة Hidden Question Miner كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Hidden Question Miner بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Hidden Question Miner بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Hidden Question Miner لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Hidden Question Miner كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Hidden Question Miner يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Hidden Question Miner يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه تخطأ مبهم.

يمكن تدريب Hidden Question Miner عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Question Layer Tagger

يصنف السؤال إلى L0-L5. فائدة الأداة أنها تحدد هل نحن أمام صيغة لغوية أم سؤال اختبار أم سؤال حاكم. إذا بقيت المهمة في L0 رغم تعقيدها، يرفع النظام escalation إلى تفكيك أو استرجاع.

تعمل أداة Question Layer Tagger كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Question Layer Tagger بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Question Layer Tagger بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Question Layer Tagger لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Question Layer Tagger كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Question Layer Tagger يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Question Layer Tagger يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه تخطأ مبهم.

يمكن تدريب Question Layer Tagger عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Critical Difference Extractor

يستخرج Δ من السؤال: ما الفرق الذي يغير القرار؟ يميز بين الفرق الظاهر والفرق الخفي والفرق السلبي. ينتج delta_map يحتوي الفرق، الدليل، المخاطر، وعلاقته بالهدف.

تعمل أداة Critical Difference Extractor كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Critical Difference Extractor بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Critical Difference Extractor بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Critical Difference Extractor لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Critical Difference Extractor كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Critical Difference Extractor يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Critical Difference Extractor يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه تخطأ مبهم.

يمكن تدريب Critical Difference Extractor عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

MCD Meta-Complexity Mapper

يحول delta_map إلى أبعاد D/I/T/A/P/C/N. يحدد مقدار التوسع المطلوب. إذا ارتفع N بسبب غموض أو تلوث، لا يرفع الثقة بل يرفع الحاجة إلى المحكمة.

تعمل أداة MCD Meta-Complexity Mapper كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة MCD Meta-Complexity Mapper بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود MCD Meta-Complexity Mapper بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن MCD Meta-Complexity Mapper لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل MCD Meta-Complexity Mapper كعامل مستقل أو نخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار MCD Meta-Complexity Mapper يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ MCD Meta-Complexity Mapper يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه نخطأ مبهم.

يمكن تدريب MCD Meta-Complexity Mapper عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Q-MCD Scorer

يربط السؤال بالفرق: يقيس QG وQR وQE وQ-depth. يحدد هل السؤال يستحق مسارات متعددة، وهل الجواب المباشر كاف أم مضلل.

تعمل أداة Q-MCD Scorer كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Q-MCD Scorer بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Q-MCD Scorer بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Q-MCD Scorer لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Q-MCD Scorer كعامل مستقل أو نخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Q-MCD Scorer يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Q-MCD Scorer يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه نخطأ مبهم.

يمكن تدريب Q-MCD Scorer عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

HAKIM Governance Kernel

ينفذ Hypothesis/Audit/Knowledge/Intervention/Metrics. يحدد claim rank، مصدر القرار، ومسؤولية كل طبقة. هذه الأداة تمنع الوكيل من استخدام نموذج واحد كحاكم مطلق.

تعمل أداة HAKIM Governance Kernel كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة HAKIM Governance Kernel بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود HAKIM Governance Kernel بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن HAKIM Governance Kernel لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل HAKIM Governance Kernel كعامل مستقل أو نخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار HAKIM Governance Kernel يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ HAKIM Governance Kernel يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه نخطأ مبهم.

يمكن تدريب HAKIM Governance Kernel عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Vascular Compute Router

يوزع compute بين نموذج، أداة، استرجاع، محاكاة، ومحكمة. يقرأ MCD والخصوصية وlatency والrisk. نجاحه يقاس بخفض تكلفة التفكير دون خفض جودة الحكم.

تعمل أداة Vascular Compute Router كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Vascular Compute Router بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Vascular Compute Router بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Vascular Compute Router لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Vascular Compute Router كعامل مستقل أو كخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Vascular Compute Router يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Vascular Compute Router يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه كخطأ مبهم.

يمكن تدريب Vascular Compute Router عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Inner-Time Budgeter

يحدد عدد دورات التفكير والتفتيش بناء على ITC. يمنع الإسهاب غير الضروري ويمنع الاستعجال في المهام عالية الأثر.

تعمل أداة Inner-Time Budgeter كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Inner-Time Budgeter بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Inner-Time Budgeter بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Inner-Time Budgeter لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Inner-Time Budgeter كعامل مستقل أو كخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Inner-Time Budgeter يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Inner-Time Budgeter يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه كخطأ مبهم.

يمكن تدريب Inner-Time Budgeter عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Candidate Path Generator

يفتح مسارات حل متعددة: نصي، سببي، أداتي، استرجاعي، محاكاة. ثم يعطي كل مسار obligation: ما الدليل الذي يحتاجه ومتى يسقط.

تعمل أداة Candidate Path Generator كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Candidate Path Generator بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Candidate Path Generator بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Candidate Path Generator لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Candidate Path Generator كعامل مستقل أو كخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Candidate Path Generator يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Candidate Path Generator يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه كخطأ مبهم.

يمكن تدريب Candidate Path Generator عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Causal Counterfactual Forge

يبني أسئلة مضادة: ماذا لو كان السبب غير هذا؟ ماذا لو كان المصدر خاطئا؟ يساعد في كشف الارتباطات الزائفة وإنتاج اختبارات سببية.

تعمل أداة Causal Counterfactual Forge كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Causal Counterfactual Forge بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Causal Counterfactual Forge بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Causal Counterfactual Forge لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Causal Counterfactual Forge كعامل مستقل أو كخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Causal Counterfactual Forge يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Causal Counterfactual Forge يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه كخطأ مبهم.

يمكن تدريب Causal Counterfactual Forge عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

GraphWorld Builder

يبني عالما علائقيا من كيانات وعلاقات وأحداث وأدلة. يستخدم حين لا يكفي RAG النصي. يدعم التحقق من الاتساق السببي قبل الجواب.

تعمل أداة GraphWorld Builder كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة GraphWorld Builder بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود GraphWorld Builder بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن GraphWorld Builder لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل GraphWorld Builder كعامل مستقل أو كخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار GraphWorld Builder يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ GraphWorld Builder يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه كخطأ مبهم.

يمكن تدريب GraphWorld Builder عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

GraphRAG Validation Layer

يربط الادعاء بمصادر وعلاقات. لا يكتفي بإحضار فقرة، يسأل هل العلاقة التي نستخدمها مدعومة؟ وهل يوجد تناقض؟

تعمل أداة GraphRAG Validation Layer كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة GraphRAG Validation Layer بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود GraphRAG Validation Layer بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن GraphRAG Validation Layer لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل GraphRAG Validation Layer كعامل مستقل أو كخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار GraphRAG Validation Layer يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ GraphRAG Validation Layer يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه كخطأ مبهم.

يمكن تدريب GraphRAG Validation Layer عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Epistemic Immune Filter

يمنع المصدر الملوث أو الذاكرة القديمة أو injection من السيطرة على القرار. يخرج verdict: allow، restrict، court، local-only، أو human gate.

تعمل أداة Epistemic Immune Filter كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Epistemic Immune Filter بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Epistemic Immune Filter بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Epistemic Immune Filter لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Epistemic Immune Filter كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Epistemic Immune Filter يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Epistemic Immune Filter يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه تخطأ مبهم.

يمكن تدريب Epistemic Immune Filter عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Falsification Court

تهاجم الادعاء بأقوى اعتراضات ممكنة. لا تنتج جوابا نهائيا، بل verdict وشروط إصلاح. كل claim عالي الأثر يجب أن يمر هنا.

تعمل أداة Falsification Court كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Falsification Court بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Falsification Court بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Falsification Court لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Falsification Court كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Falsification Court يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Falsification Court يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه تخطأ مبهم.

يمكن تدريب Falsification Court عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Perturbation Court

تغير الشروط والمدخلات لتعرف هل الجواب مستقر أم هش. إذا انهار الحكم عند تغيير بسيط، يخفض claim rank.

تعمل أداة Perturbation Court كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Perturbation Court بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Perturbation Court بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Perturbation Court لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Perturbation Court كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Perturbation Court يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Perturbation Court يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه كخطأ مبهم.

يمكن تدريب Perturbation Court عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Verifier Swarm Arena

يجعل عدة verifiers مستقلة تقيم النتيجة: مصدر، منطق، تنفيذ، أمن، وbenchmark. استقلال verifier مهم حتى لا يدافع مولد الجواب عن نفسه.

تعمل أداة Verifier Swarm Arena كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Verifier Swarm Arena بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Verifier Swarm Arena بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Verifier Swarm Arena لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Verifier Swarm Arena كعامل مستقل أو كخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Verifier Swarm Arena يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Verifier Swarm Arena يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه كخطأ مبهم.

يمكن تدريب Verifier Swarm Arena عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Fusion Synthesizer

يدمج مسارات متعددة بعد المحكمة. لا يجمع الآراء حسابيا فقط، يوزنها بالدليل والاتساق والتناقض وحداثة المصدر.

تعمل أداة Fusion Synthesizer كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Fusion Synthesizer بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Fusion Synthesizer بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Fusion Synthesizer لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Fusion Synthesizer كعامل مستقل أو كخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Fusion Synthesizer يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Fusion Synthesizer يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه كخطأ مبهم.

يمكن تدريب Fusion Synthesizer عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

FusionConfidence Meter

يقيس confidence بعد الدمج: evidence_freshness + verifier_pass - contradiction_density - support_ratio. يمنع الثقة الانطباعية.

تعمل أداة FusionConfidence Meter كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة FusionConfidence Meter بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود FusionConfidence Meter بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن FusionConfidence Meter لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل FusionConfidence Meter كعامل مستقل أو كخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار FusionConfidence Meter يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ FusionConfidence Meter يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه كخطأ مبهم.

يمكن تدريب FusionConfidence Meter عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

World-Effect Monitor

يقيس أثر الجواب في المهمة أو العالم. هل بني artifact؟ هل نفذ اختبار؟ هل تغيّر القرار؟

تعمل أداة World-Effect Monitor كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة World-Effect Monitor بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود World-Effect Monitor بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن World-Effect Monitor لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل World-Effect Monitor كعامل مستقل أو كخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار World-Effect Monitor يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ World-Effect Monitor يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه كخطأ مبهم.

يمكن تدريب World-Effect Monitor عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Self-Reconstruction Memory

تكتب trace بعد كل مهمة: السؤال، الفرق، route، المحكمة، الأثر، وما يجب تغييره. لا تحفظ النص فقط بل تحفظ سبب القرار.

تعمل أداة Self-Reconstruction Memory كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Self-Reconstruction Memory بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Self-Reconstruction Memory بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Self-Reconstruction Memory لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Self-Reconstruction Memory كعامل مستقل أو كخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Self-Reconstruction Memory يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Self-Reconstruction Memory يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه كخطأ مبهم.

يمكن تدريب Self-Reconstruction Memory عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Memory Compression Distiller

يضغط الذاكرة إلى قواعد قابلة لإعادة الاستخدام دون فقد شروط الفشل. يميز بين lesson عام وتجربة محلية.

تعمل أداة Memory Compression Distiller كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Memory Compression Distiller بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Memory Compression Distiller بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Memory Compression Distiller لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Memory Compression Distiller كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Memory Compression Distiller يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Memory Compression Distiller يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه تخطأ مبهم.

يمكن تدريب Memory Compression Distiller عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Anti-Hijack Source Ledger

يسجل كل مصدر وكل أداة وكل route مع risk وpermissions. يحمي من اختطاف الذاكرة أو المصدر أو المزود.

تعمل أداة Anti-Hijack Source Ledger كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Anti-Hijack Source Ledger بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Anti-Hijack Source Ledger بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Anti-Hijack Source Ledger لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Anti-Hijack Source Ledger كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Anti-Hijack Source Ledger يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Anti-Hijack Source Ledger يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه تخطأ مبهم.

يمكن تدريب Anti-Hijack Source Ledger عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Privacy and Permission Gate

يفصل بين capability وauthority. نموذج قوي لا يحق له رؤية أسرار أو تنفيذ أفعال مدمرة. يحدد hosted/local/human gate.

تعمل أداة Privacy and Permission Gate كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Privacy and Permission Gate بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Privacy and Permission Gate بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Privacy and Permission Gate لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Privacy and Permission Gate كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Privacy and Permission Gate يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Privacy and Permission Gate يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه تخطأ مبهم.

يمكن تدريب Privacy and Permission Gate عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Tool Sandbox

يشغل الأدوات في بيئة مقيدة، مع dry-run وrollback. يضمن أن الوكيل لا ينتقل من فكرة إلى فعل خطير بلا حاكمية.

تعمل أداة Tool Sandbox كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Tool Sandbox بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Tool Sandbox بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Tool Sandbox لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Tool Sandbox كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Tool Sandbox يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Tool Sandbox يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه تخطأ مبهم.

يمكن تدريب Tool Sandbox عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Adaptive Toolchain Planner MATP

يختار سلسلة الأدوات المناسبة: بحث، كود، اختبار، تحليل، تحويل. لا يختار أداة لأنها متاحة بل لأنها تغير الأثر.

تعمل أداة Adaptive Toolchain Planner MATP كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Adaptive Toolchain Planner MATP بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Adaptive Toolchain Planner MATP بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Adaptive Toolchain Planner MATP لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Adaptive Toolchain Planner MATP كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Adaptive Toolchain Planner MATP يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Adaptive Toolchain Planner MATP يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه تخطأ مبهم.

يمكن تدريب Adaptive Toolchain Planner MATP عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Proof-Program Bridge MPPB

يربط الفرضية النظرية بإجراء تنفيذي أو اختبار. إذا عجز المفهوم عن إنتاج برنامج أو اختبار فهو غير ناضج تقنيا.

تعمل أداة Proof-Program Bridge MPPB كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Proof-Program Bridge MPPB بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Proof-Program Bridge MPPB بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Proof-Program Bridge MPPB لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Proof-Program Bridge MPPB كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Proof-Program Bridge MPPB يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Proof-Program Bridge MPPB يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه تخطأ مبهم.

يمكن تدريب Proof-Program Bridge MPPB عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

MCD Heatmap UI

يعرض أين يتركز التعقيد: السؤال، المصدر، الأداة، الذاكرة، أو العالم. يساعد الباحث على معرفة موضع الفشل.

تعمل أداة MCD Heatmap UI كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة MCD Heatmap UI بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود MCD Heatmap UI بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن MCD Heatmap UI لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل MCD Heatmap UI كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار MCD Heatmap UI يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ MCD Heatmap UI يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه كخطأ مبهم.

يمكن تدريب MCD Heatmap UI عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Benchmark Tribunal

يمنع تلفيق درجات. يفصل بين target envelope ونتيجة منفذة. يحفظ setup والمهام والنموذج والإصدار.

تعمل أداة Benchmark Tribunal كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Benchmark Tribunal بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Benchmark Tribunal بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Benchmark Tribunal لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Benchmark Tribunal كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Benchmark Tribunal يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Benchmark Tribunal يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه كخطأ مبهم.

يمكن تدريب Benchmark Tribunal عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Vascular Curriculum Generator

ينتج مناهج تدريب تصعد من فروق سهلة إلى فروق عالية الحاكية، مع تدرج في الأدوات والمحكمة.

تعمل أداة Vascular Curriculum Generator كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Vascular Curriculum Generator بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Vascular Curriculum Generator بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Vascular Curriculum Generator لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Vascular Curriculum Generator كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Vascular Curriculum Generator يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Vascular Curriculum Generator يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه كخطأ مبهم.

يمكن تدريب Vascular Curriculum Generator عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Failure Distillation Lab

يحول الفشل إلى بيانات تدريب: لماذا فشل؟ أي طبقة أخطأت؟ كيف نعيد كتابته كدرس؟

تعمل أداة Failure Distillation Lab كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Failure Distillation Lab بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Failure Distillation Lab بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Failure Distillation Lab لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Failure Distillation Lab كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Failure Distillation Lab يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Failure Distillation Lab يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه تخطأ مبهم.

يمكن تدريب Failure Distillation Lab عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Causal Replay Forge

يعيد تشغيل المهمة مع تغيير متغيرات سببية لمعرفة ما الذي صنع النتيجة فعلا.

تعمل أداة Causal Replay Forge كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Causal Replay Forge بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Causal Replay Forge بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Causal Replay Forge لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Causal Replay Forge كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Causal Replay Forge يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Causal Replay Forge يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه تخطأ مبهم.

يمكن تدريب Causal Replay Forge عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Benchmark Anti-Gaming Lab

يصمم مهام متحورة تمنع حفظ الأنماط وتكشف هل توجد مهارة حقيقية أم استغلال لمعيار.

تعمل أداة Benchmark Anti-Gaming Lab كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Benchmark Anti-Gaming Lab بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Benchmark Anti-Gaming Lab بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Benchmark Anti-Gaming Lab لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Benchmark Anti-Gaming Lab كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Benchmark Anti-Gaming Lab يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Benchmark Anti-Gaming Lab يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه تخطأ مبهم.

يمكن تدريب Benchmark Anti-Gaming Lab عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Governance Reward Model

يكافئ صدق المصدر، خفض التناقض، سلامة الأداة، وذاكرة نافعة؛ لا يكافئ رضا المستخدم وحده.

تعمل أداة Governance Reward Model كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Governance Reward Model بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Governance Reward Model بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Governance Reward Model لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Governance Reward Model كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Governance Reward Model يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Governance Reward Model يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه تخطأ مبهم.

يمكن تدريب Governance Reward Model عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Hyperskill Compiler

يحوّل نجاحات متكررة إلى HyperSkills قابلة للاستدعاء: failure modes، metrics، verifier، contract، state.

تعمل أداة Hyperskill Compiler كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Hyperskill Compiler بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Hyperskill Compiler بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Hyperskill Compiler لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Hyperskill Compiler كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Hyperskill Compiler يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Hyperskill Compiler يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه تخطأ مبهم.

يمكن تدريب Hyperskill Compiler عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Hyperloop Closure Meter

يقيس إغلاق الدورة بين الهدف والأثر. إذا تكررت الدورات بلا زيادة closure، يوقف المسار.

تعمل أداة Hyperloop Closure Meter كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Hyperloop Closure Meter بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Hyperloop Closure Meter بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Hyperloop Closure Meter لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Hyperloop Closure Meter كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Hyperloop Closure Meter يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Hyperloop Closure Meter يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه تخطأ مبهم.

يمكن تدريب Hyperloop Closure Meter عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Agentic Autopsy Notebook

سجل تشريح لكل مهمة فاشلة أو ناجحة: أين بدأ الخطأ؟ أي route كان يجب اختياره؟ ما الذي تغير في الذاكرة؟

تعمل أداة Agentic Autopsy Notebook كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Agentic Autopsy Notebook بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Agentic Autopsy Notebook بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Agentic Autopsy Notebook لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Agentic Autopsy Notebook كعامل مستقل أو كخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Agentic Autopsy Notebook يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Agentic Autopsy Notebook يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه كخطأ مبهم.

يمكن تدريب Agentic Autopsy Notebook عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

GraphRAG Governance Map

خريطة توضح علاقة الادعاءات بمصادرها وTruthStatus وأثرها في الحكم.

تعمل أداة GraphRAG Governance Map كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة GraphRAG Governance Map بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود GraphRAG Governance Map بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن GraphRAG Governance Map لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل GraphRAG Governance Map كعامل مستقل أو كخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار GraphRAG Governance Map يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ GraphRAG Governance Map يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه كخطأ مبهم.

يمكن تدريب GraphRAG Governance Map عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Anti-Sycophancy Evaluator

يقيس هل اختار النظام الحقيقة رغم انخفاض رضا المستخدم اللحظي. مهم للذكاء الفتّاح لأنه يمنع الطمأنة الكاذبة.

تعمل أداة Anti-Sycophancy Evaluator كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Anti-Sycophancy Evaluator بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Anti-Sycophancy Evaluator بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Anti-Sycophancy Evaluator لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Anti-Sycophancy Evaluator كعامل مستقل أو كخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Anti-Sycophancy Evaluator يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Anti-Sycophancy Evaluator يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه كخطأ مبهم.

يمكن تدريب Anti-Sycophancy Evaluator عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Tool-Minimality Benchmark

يقيس هل استخدم النظام أقل أدوات كافية دون تضحية بالسلامة. يمنع tool overuse.

تعمل أداة Tool-Minimality Benchmark كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Tool-Minimality Benchmark بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Tool-Minimality Benchmark بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Tool-Minimality Benchmark لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Tool-Minimality Benchmark كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Tool-Minimality Benchmark يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Tool-Minimality Benchmark يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه تخطأ مبهم.

يمكن تدريب Tool-Minimality Benchmark عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Provider Adapter

يفصل API عن السياسة. يسمح بنقل backend من hosted إلى NIM أو local دون تغيير منطق الحاكمية.

تعمل أداة Provider Adapter كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Provider Adapter بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Provider Adapter بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Provider Adapter لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Provider Adapter كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Provider Adapter يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Provider Adapter يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه تخطأ مبهم.

يمكن تدريب Provider Adapter عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Model Capability Cards

تصف قدرة كل نموذج ومجاله وسعره وخصوصيته وسياقه وتاريخه. لا يختار الروتر model name بل capability card.

تعمل أداة Model Capability Cards كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Model Capability Cards بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Model Capability Cards بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Model Capability Cards لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Model Capability Cards كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Model Capability Cards يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Model Capability Cards يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه تخطأ مبهم.

يمكن تدريب Model Capability Cards عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

RouteScore Engine

ينفذ معادلة route_score مع taxes للزمن والخطر والكلفة وفشل المزود.

تعمل أداة RouteScore Engine كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة RouteScore Engine بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود RouteScore Engine بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن RouteScore Engine لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل RouteScore Engine كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار RouteScore Engine يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ RouteScore Engine يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه تخطأ مبهم.

يمكن تدريب RouteScore Engine عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Golden Tasks Harness

مجموعة مهام مرجعية داخلية تختبر السؤال والفرق والمحكمة والذاكرة، لا الجواب وحده.

تعمل أداة Golden Tasks Harness كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Golden Tasks Harness بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Golden Tasks Harness بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Golden Tasks Harness لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Golden Tasks Harness كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Golden Tasks Harness يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Golden Tasks Harness يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه تخطأ مبهم.

يمكن تدريب Golden Tasks Harness عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

Deployment Roadmap Controller

يتدرج من prototype إلى deterministic router إلى courted fusion إلى domain adapters إلى learned routing إلى production governance.

تعمل أداة Deployment Roadmap Controller كعقدة داخل pipeline وليست إضافة زخرفية. يجب أن تعرف أين تقف: قبل السؤال، بعد الفرق، في route، داخل المحكمة، أو في الذاكرة. تحديد الموضع يمنع تداخل الصلاحيات بين المولد والمحقق والذاكرة.

ينبغي أن تُعرَّف أداة Deployment Roadmap Controller بمدخلات ومخرجات صريحة. المدخلات النموذجية هي evidence_budget، privacy_tier، context_fragment، task_state، وroute_memory. المخرجات ليست فقرة حرة، بل failure_modes، next_action، confidence، claim_rank، وmemory_suggestion. بهذا تصبح الأداة جزءا من kernel لا زينة داخل prompt.

العلاقة مع MCD واضحة: إذا رفعت الأداة دقة الحكم أو خفضت الضجيج أو حسنت أثر العالم، تستحق compute. وإذا لم تفعل، تصبح tax زائد. لذلك يجب أن تعود Deployment Roadmap Controller بمؤشر يمكن إدخاله في MCD أو RouteScore لا بمجرد نص عام.

العلاقة مع HAKIM-MCD تعني أن Deployment Roadmap Controller لا تملك سلطة مطلقة. حتى إن كانت الأداة قوية، يجب أن تمر عبر Governance Kernel الذي يقرر هل يسمح لها بالعمل، وما حدود الخصوصية، وهل تحتاج human gate أو sandbox.

في سياق Hermes-Fugu، يمكن أن يعمل Deployment Roadmap Controller كعامل مستقل أو تخدمة يستدعيها router. المهم أن تظهر في trace: لماذا استدعيت، ماذا أعادت، ما اعتراض المحكمة عليها، وهل كتبت ذاكرة نافعة.

اختبار Deployment Roadmap Controller يتم عبر ablation: نزيل الأداة من النظام ونقيس الفرق في correctness وsource honesty وroute cost وmemory usefulness. فإذا لم يتغير شيء، فهذا يعني أن الأداة لا تنفذ بعد وظيفة حقيقية. وإذا زادت التكلفة دون تحسن في الأثر، تعدل policy أو تخفض الأداة إلى وضع اختياري.

نمط الفشل الخاص بـ Deployment Roadmap Controller يجب أن يكون مسمى. قد تفشل بسبب over-routing، أو ثقة زائفة، أو اعتماد على مصدر واحد، أو تسريب سياق حساس، أو غياب دليل كاف. تسمية الفشل تجعل الذاكرة تتعلم منه بدلا من تخزينه نخطأ مبهم.

يمكن تدريب Deployment Roadmap Controller عبر positive/negative traces: أثر ناجح مع verifier_pass مقابل أثر فاشل أو contradiction_density مرتفع. يتعلم الروتر متى يستدعي الأداة ومتى يتجنبها، بينما تبقى القيود الصريحة مثل الخصوصية والصلاحيات خارج التعلم الأسود.

الباب الرابع: مناهج التدريب والتطور

تدريب الفروق لا الإجابات

البيانات لا تقدم سؤالا وجوابا فقط، بل تقدم فرقاً فاعلاً وسبب كونه فاعلاً. يتعلم النموذج أن يسأل: ما الذي يغير القرار؟

هذا المنهج يرتبط بمصفوفة التدريب العامة: task → hidden question → delta → route → court → outcome → memory. لا تكفي له بيانات question/answer، بل يحتاج traces. كل trace يحمل سبب route، الأدوات المستعملة، verdict المحكمة، أثر العالم، وقرار تحديث الذاكرة. من دون trace لا يمكن تحويل المنهج إلى learning signal.

مؤشر النجاح في تدريب الفروق لا الإجابات هو transfer لا التحسن المحلي فقط. نريد أن يحسن النظام قدرته على مواجهة مهمة جديدة من نفس العائلة دون حفظ الشكل. لذلك تقاس النتائج عبر few-shot transfer، مقاومة التلوث، وفشل أقل عند اختلاف المصدر أو الأداة.

يجب أن يكون تدريب الفروق لا الإجابات قابلا للأبلشن. نقارن نموذجا مباشرا، ونموذجا مع سؤال فقط، ونموذجا مع Q-MCD، ونموذجا مع HAKIM ومحكمة وذاكرة. إذا لم يظهر التدرج في النتائج، فإما أن المنهج غير ناضج أو أن المهمة غير مناسبة.

في التدريب العملي، لا تُعطى المكافأة للجواب النهائي وحده. توزع المكافأة على صدق المصدر، سلامة الأداة، جودة route، قوة التفنيد، وإفادة الذاكرة. بهذه الطريقة يصبح التدريب تعليما للحاكمية لا تعليما للفصاحة.

علاقة تدريب الفروق لا الإجابات بالأمان أساسية. المسارات التي تحقق نتيجة صحيحة لكنها تكسر خصوصية أو تستخدم أداة بلا إذن لا تعد إيجابية. بهذا يفصل المشروع بين القدرة والحق في التنفيذ.

ينبغي أن يخرج تدريب الفروق لا الإجابات بيانات صالحة لتحديث الروتر: memory_gain، human_override، risk_tax، latency، failure_rate، route_win_rate. بعد تراكم هذه البيانات يمكن الانتقال من routing rule-based إلى learned routing دون فقدان الشفافية.

تدريب التفنيد قبل الثقة

كل جواب تدريبي يرافقه اعتراضان وشروط سقوط. الهدف أن لا يكتسب النظام عادة التصديق قبل المحكمة.

هذا المنهج يرتبط بمصفوفة التدريب العامة: task → hidden question → delta → route → court → outcome → memory. لا تكفي له بيانات question/answer، بل يحتاج traces. كل trace يحمل سبب route، الأدوات المستعملة، verdict المحكمة، أثر العالم، وقرار تحديث الذاكرة. من دون trace لا يمكن تحويل المنهج إلى learning signal.

مؤشر النجاح في تدريب التفنيد قبل الثقة هو transfer لا التحسن المحلي فقط. نريد أن يحسن النظام قدرته على مواجهة مهمة جديدة من نفس العائلة دون حفظ الشكل. لذلك تقاس النتائج عبر few-shot transfer، مقاومة التلوث، وفشل أقل عند اختلاف المصدر أو الأداة.

يجب أن يكون تدريب التفنيد قبل الثقة قابلا للأبلشن. نقارن نموذجا مباشرا، ونموذجا مع سؤال فقط، ونموذجا مع Q-MCD، ونموذجا مع HAKIM ومحكمة وذاكرة. إذا لم يظهر التدرج في النتائج، فإما أن المنهج غير ناضج أو أن المهمة غير مناسبة.

في التدريب العملي، لا تُعطى المكافأة للجواب النهائي وحده. توزع المكافأة على صدق المصدر، سلامة الأداة، جودة route، قوة التفنيد، وإفادة الذاكرة. بهذه الطريقة يصبح التدريب تعليما للحاكمية لا تعليما للفصاحة.

علاقة تدريب التفنيد قبل الثقة بالأمان أساسية. المسارات التي تحقق نتيجة صحيحة لكنها تكسر خصوصية أو تستخدم أداة بلا إذن لا تعد إيجابية. بهذا يفصل المشروع بين القدرة والحق في التنفيذ.

ينبغي أن يخرج تدريب التفنيد قبل الثقة بيانات صالحة لتحديث الروتر: memory_gain، human_override، risk_tax، latency، failure_rate، route_win_rate. بعد تراكم هذه البيانات يمكن الانتقال من routing rule-based إلى learned routing دون فقدان الشفافية.

تدريب الذاكرة كإعادة بناء

لا يُكافأ النظام لأنه حفظ نصا، بل لأنه حوّل تجربة إلى قاعدة route أو failure note.

هذا المنهج يرتبط بمصفوفة التدريب العامة: task → hidden question → delta → route → court → outcome → memory. لا تكفي له بيانات question/answer، بل يحتاج traces. كل trace يحمل سبب route، الأدوات المستعملة، verdict المحكمة، أثر العالم، وقرار تحديث الذاكرة. من دون trace لا يمكن تحويل المنهج إلى learning signal.

مؤشر النجاح في تدريب الذاكرة كإعادة بناء هو transfer لا التحسن المحلي فقط. نريد أن يحسن النظام قدرته على مواجهة مهمة جديدة من نفس العائلة دون حفظ الشكل. لذلك تقاس النتائج عبر few-shot transfer، مقاومة التلوث، وفشل أقل عند اختلاف المصدر أو الأداة.

يجب أن يكون تدريب الذاكرة كإعادة بناء قابلا للأبلشن. نقارن نموذجا مباشرا، ونموذجا مع سؤال فقط، ونموذجا مع Q-MCD، ونموذجا مع HAKIM ومحكمة وذاكرة. إذا لم يظهر التدرج في النتائج، فإما أن المنهج غير ناضج أو أن المهمة غير مناسبة.

في التدريب العملي، لا تُعطى المكافأة للجواب النهائي وحده. توزع المكافأة على صدق المصدر، سلامة الأداة، جودة route، قوة التفنيد، وإفادة الذاكرة. بهذه الطريقة يصبح التدريب تعليما للحاكمية لا تعليما للفصاحة.

علاقة تدريب الذاكرة كإعادة بناء بالأمان أساسية. المسارات التي تحقق نتيجة صحيحة لكنها تكسر خصوصية أو تستخدم أداة بلا إذن لا تعد إيجابية. بهذا يفصل المشروع بين القدرة والحق في التنفيذ.

ينبغي أن يخرج تدريب الذاكرة كإعادة بناء بيانات صالحة لتحديث الروتر: memory_gain، human_override، risk_tax، latency، failure_rate، route_win_rate. بعد تراكم هذه البيانات يمكن الانتقال من routing rule-based إلى learned routing دون فقدان الشفافية.

تدريب العالم السببي

يعلم النموذج تمثيل الكيانات والعلاقات والتأثيرات، لا استرجاع جمل فقط.

هذا المنهج يرتبط بمصفوفة التدريب العامة: task → hidden question → delta → route → court → outcome → memory. لا تكفي له بيانات question/answer، بل يحتاج traces. كل trace يحمل سبب route، الأدوات المستعملة، verdict المحكمة، أثر العالم، وقرار تحديث الذاكرة. من دون trace لا يمكن تحويل المنهج إلى learning signal.

مؤشر النجاح في تدريب العالم السببي هو transfer لا التحسن المحلي فقط. نريد أن يحسن النظام قدرته على مواجهة مهمة جديدة من نفس العائلة دون حفظ الشكل. لذلك تقاس النتائج عبر few-shot transfer، مقاومة التلوث، وفشل أقل عند اختلاف المصدر أو الأداة.

يجب أن يكون تدريب العالم السببي قابلا للأبلشن. نقارن نموذجا مباشرا، ونموذجا مع سؤال فقط، ونموذجا مع Q-MCD، ونموذجا مع HAKIM ومحكمة وذاكرة. إذا لم يظهر التدرج في النتائج، فإما أن المنهج غير ناضج أو أن المهمة غير مناسبة.

في التدريب العملي، لا تُعطى المكافأة للجواب النهائي وحده. توزع المكافأة على صدق المصدر، سلامة الأداة، جودة route، قوة التفنيد، وإفادة الذاكرة. بهذه الطريقة يصبح التدريب تعليما للحاكمية لا تعليما للفصاحة.

علاقة تدريب العالم السببي بالأمان أساسية. المسارات التي تحقق نتيجة صحيحة لكنها تكسر خصوصية أو تستخدم أداة بلا إذن لا تعد إيجابية. بهذا يفصل المشروع بين القدرة والحق في التنفيذ.

ينبغي أن يخرج تدريب العالم السببي بيانات صالحة لتحديث الروتر: memory_gain، human_override، risk_tax، latency، failure_rate، route_win_rate. بعد تراكم هذه البيانات يمكن الانتقال من routing rule-based إلى learned routing دون فقدان الشفافية.

تدريب الانتقاء المساعفي

يعلم النظام متى ينقذ مسارا ومتى يدفنه. الإنقاذ بلا معيار يساوي هدر compute.

هذا المنهج يرتبط بمصفوفة التدريب العامة: task → hidden question → delta → route → court → outcome → memory. لا تكفي له بيانات question/answer، بل يحتاج traces. كل trace يحمل سبب route، الأدوات المستعملة، verdict المحكمة، أثر العالم، وقرار تحديث الذاكرة. من دون trace لا يمكن تحويل المنهج إلى learning signal.

مؤشر النجاح في تدريب الانتقاء المساعفي هو transfer لا التحسن المحلي فقط. نريد أن يحسن النظام قدرته على مواجهة مهمة جديدة من نفس العائلة دون حفظ الشكل. لذلك تقاس النتائج عبر few-shot transfer، مقاومة التلوث، وفشل أقل عند اختلاف المصدر أو الأداة.

يجب أن يكون تدريب الانتقاء المساعفي قابلا للأبلشن. نقارن نموذجا مباشرا، ونموذجا مع سؤال فقط، ونموذجا مع Q-MCD، ونموذجا مع HAKIM ومحكمة وذاكرة. إذا لم يظهر التدرج في النتائج، فإما أن المنهج غير ناضج أو أن المهمة غير مناسبة.

في التدريب العملي، لا تُعطى المكافأة للجواب النهائي وحده. توزع المكافأة على صدق المصدر، سلامة الأداة، جودة route، قوة التفنيد، وإفادة الذاكرة. بهذه الطريقة يصبح التدريب تعليما للحاكمية لا تعليما للفصاحة.

علاقة تدريب الانتقاء المساعفي بالأمان أساسية. المسارات التي تحقق نتيجة صحيحة لكنها تكسر خصوصية أو تستخدم أداة بلا إذن لا تعد إيجابية. بهذا يفصل المشروع بين القدرة والحق في التنفيذ.

ينبغي أن يخرج تدريب الانتقاء المساعفي بيانات صالحة لتحديث الروتر: memory_gain، human_override، risk_tax، latency، failure_rate، route_win_rate. بعد تراكم هذه البيانات يمكن الانتقال من routing rule-based إلى learned routing دون فقدان الشفافية.

Routing imitation from expert traces

يبدأ الروتر بقواعد شفافة ثم يتعلم من traces موثقة: task، route، workers، court، outcome، latency.

هذا المنهج يرتبط بمصفوفة التدريب العامة: task → hidden question → delta → route → court → outcome → memory. لا تكفي له بيانات question/answer، بل يحتاج traces. كل trace يحمل سبب route، الأدوات المستعملة، verdict المحكمة، أثر العالم، وقرار تحديث الذاكرة. من دون trace لا يمكن تحويل المنهج إلى learning signal.

مؤشر النجاح في Routing imitation from expert traces هو transfer لا التحسن المحلي فقط. نريد أن يحسن النظام قدرته على مواجهة مهمة جديدة من نفس العائلة دون حفظ الشكل. لذلك تقاس النتائج عبر few-shot transfer، مقاومة التلوث، وفشل أقل عند اختلاف المصدر أو الأداة.

يجب أن يكون Routing imitation from expert traces قابلا للأبلشن. نقارن نموذجا مباشرا، ونموذجا مع سؤال فقط، ونموذجا مع Q-MCD، ونموذجا مع HAKIM ومحكمة وذاكرة. إذا لم يظهر التدرج في النتائج، فإما أن المنهج غير ناضج أو أن المهمة غير مناسبة.

في التدريب العملي، لا تُعطى المكافأة للجواب النهائي وحده. توزع المكافأة على صدق المصدر، سلامة الأداة، جودة route، قوة التفنيد، وإفادة الذاكرة. بهذه الطريقة يصبح التدريب تعليما للحاكمية لا تعليما للفصاحة.

علاقة Routing imitation from expert traces بالأمان أساسية. المسارات التي تحقق نتيجة صحيحة لكنها تكسر خصوصية أو تستخدم أداة بلا إذن لا تعد إيجابية. بهذا يفصل المشروع بين القدرة والحق في التنفيذ.

ينبغي أن يخرج Routing imitation from expert traces بيانات صالحة لتحديث الروتر: memory_gain، human_override، risk_tax، latency، failure_rate، route_win_rate. بعد تراكم هذه البيانات يمكن الانتقال من routing rule-based إلى learned routing دون فقدان الشفافية.

Pairwise route preference learning

يقارن routes ناجحة وفاشلة لتحديث سياسة الاختيار دون فقدان حدود الخصوصية والأدوات.

هذا المنهج يرتبط بمصفوفة التدريب العامة: task → hidden question → delta → route → court → outcome → memory. لا تكفي له بيانات question/answer، بل يحتاج traces. كل trace يحمل سبب route، الأدوات المستعملة، verdict المحكمة، أثر العالم، وقرار تحديث الذاكرة. من دون trace لا يمكن تحويل المنهج إلى learning signal.

مؤشر النجاح في Pairwise route preference learning هو transfer لا التحسن المحلي فقط. نريد أن يحسن النظام قدرته على مواجهة مهمة جديدة من نفس العائلة دون حفظ الشكل. لذلك تقاس النتائج عبر few-shot transfer، مقاومة التلوث، وفشل أقل عند اختلاف المصدر أو الأداة.

يجب أن يكون Pairwise route preference learning قابلا للأبلشن. نقارن نموذجا مباشرا، ونموذجا مع سؤال فقط، ونموذجا مع Q-MCD، ونموذجا مع HAKIM ومحكمة وذاكرة. إذا لم يظهر التدرج في النتائج، فإما أن المنهج غير ناضج أو أن المهمة غير مناسبة.

في التدريب العملي، لا تُعطى المكافأة للجواب النهائي وحده. توزع المكافأة على صدق المصدر، سلامة الأداة، جودة route، قوة التفنيد، وإفادة الذاكرة. بهذه الطريقة يصبح التدريب تعليما للحاكمية لا تعليما للفصاحة.

علاقة Pairwise route preference learning بالأمان أساسية. المسارات التي تحقق نتيجة صحيحة لكنها تكسر خصوصية أو تستخدم أداة بلا إذن لا تعد إيجابية. بهذا يفصل المشروع بين القدرة والحق في التنفيذ.

ينبغي أن يخرج Pairwise route preference learning بيانات صالحة لتحديث الروتر: memory_gain، human_override، risk_tax، latency، failure_rate، route_win_rate. بعد تراكم هذه البيانات يمكن الانتقال من routing rule-based إلى learned routing دون فقدان الشفافية.

Verifier reward shaping

يكافئ المسارات التي تخرج من verifier مستقل، لا التي تنتج نصا جميلا.

هذا المنهج يرتبط بمصفوفة التدريب العامة: task → hidden question → delta → route → court → outcome → memory. لا تكفي له بيانات question/answer، بل يحتاج traces. كل trace يحمل سبب route، الأدوات المستعملة، verdict المحكمة، أثر العالم، وقرار تحديث الذاكرة. من دون trace لا يمكن تحويل المنهج إلى learning signal.

مؤشر النجاح في Verifier reward shaping هو transfer لا التحسن المحلي فقط. نريد أن يحسن النظام قدرته على مواجهة مهمة جديدة من نفس العائلة دون حفظ الشكل. لذلك تقاس النتائج عبر few-shot transfer، مقاومة التلوث، وفشل أقل عند اختلاف المصدر أو الأداة.

يجب أن يكون Verifier reward shaping قابلا للأبلشن. نقارن نموذجا مباشرا، ونموذجا مع سؤال فقط، ونموذجا مع Q-MCD، ونموذجا مع HAKIM ومحكمة وذاكرة. إذا لم يظهر التدرج في النتائج، فإما أن المنهج غير ناضج أو أن المهمة غير مناسبة.

في التدريب العملي، لا تُعطى المكافأة للجواب النهائي وحده. توزع المكافأة على صدق المصدر، سلامة الأداة، جودة route، قوة التفنيد، وإفادة الذاكرة. بهذه الطريقة يصبح التدريب تعليما للحاكمية لا تعليما للفصاحة.

علاقة Verifier reward shaping بالأمان أساسية. المسارات التي تحقق نتيجة صحيحة لكنها تكسر خصوصية أو تستخدم أداة بلا إذن لا تعد إيجابية. بهذا يفصل المشروع بين القدرة والحق في التنفيذ.

ينبغي أن يخرج Verifier reward shaping بيانات صالحة لتحديث الروتر: memory_gain، human_override، risk_tax، latency، failure_rate، route_win_rate. بعد تراكم هذه البيانات يمكن الانتقال من routing rule-based إلى learned routing دون فقدان الشفافية.

GraphRAG constrained training

يعاقب الادعاء غير المحال إلى شبكة مصادر صالحة، ويكافئ الاسترجاع الذي يخفض التناقض.

هذا المنهج يرتبط بمصفوفة التدريب العامة: task → hidden question → delta → route → court → outcome → memory. لا تكفي له بيانات question/answer، بل يحتاج traces. كل trace يحمل سبب route، الأدوات المستعملة، verdict المحكمة، أثر العالم، وقرار تحديث الذاكرة. من دون trace لا يمكن تحويل المنهج إلى learning signal.

مؤشر النجاح في GraphRAG constrained training هو transfer لا التحسن المحلي فقط. نريد أن يحسن النظام قدرته على مواجهة مهمة جديدة من نفس العائلة دون حفظ الشكل. لذلك تقاس النتائج عبر few-shot transfer، مقاومة التلوث، وفشل أقل عند اختلاف المصدر أو الأداة.

يجب أن يكون GraphRAG constrained training قابلا للأبلشن. نقارن نموذجا مباشرا، ونموذجا مع سؤال فقط، ونموذجا مع Q-MCD، ونموذجا مع HAKIM ومحكمة وذاكرة. إذا لم يظهر التدرج في النتائج، فإما أن المنهج غير ناضج أو أن المهمة غير مناسبة.

في التدريب العملي، لا تُعطى المكافأة للجواب النهائي وحده. توزع المكافأة على صدق المصدر، سلامة الأداة، جودة route، قوة التفنيد، وإفادة الذاكرة. بهذه الطريقة يصبح التدريب تعليما للحاكمية لا تعليما للفصاحة.

علاقة GraphRAG constrained training بالأمان أساسية. المسارات التي تحقق نتيجة صحيحة لكنها تكسر خصوصية أو تستخدم أداة بلا إذن لا تعد إيجابية. بهذا يفصل المشروع بين القدرة والحق في التنفيذ.

ينبغي أن يخرج GraphRAG constrained training بيانات صالحة لتحديث الروتر: memory_gain، human_override، risk_tax، latency، failure_rate، route_win_rate. بعد تراكم هذه البيانات يمكن الانتقال من routing rule-based إلى learned routing دون فقدان الشفافية.

Tool-use evidence curriculum

يعلم النموذج متى يطلب أداة ومتى يكفي تفكير نصي، مع evidence contract لكل أداة.

هذا المنهج يرتبط بمصفوفة التدريب العامة: task → hidden question → delta → route → court → outcome → memory. لا تكفي له بيانات question/answer، بل يحتاج traces. كل trace يحمل سبب route، الأدوات المستعملة، verdict المحكمة، أثر العالم، وقرار تحديث الذاكرة. من دون trace لا يمكن تحويل المنهج إلى learning signal.

مؤشر النجاح في Tool-use evidence curriculum هو transfer لا التحسن المحلي فقط. نريد أن يحسن النظام قدرته على مواجهة مهمة جديدة من نفس العائلة دون حفظ الشكل. لذلك تقاس النتائج عبر few-shot transfer، مقاومة التلوث، وفشل أقل عند اختلاف المصدر أو الأداة.

يجب أن يكون Tool-use evidence curriculum قابلا للأبلشن. نقارن نموذجا مباشرا، ونموذجا مع سؤال فقط، ونموذجا مع Q-MCD، ونموذجا مع HAKIM ومحكمة وذاكرة. إذا لم يظهر التدرج في النتائج، فإما أن المنهج غير ناضج أو أن المهمة غير مناسبة.

في التدريب العملي، لا تُعطى المكافأة للجواب النهائي وحده. توزع المكافأة على صدق المصدر، سلامة الأداة، جودة route، قوة التفنيد، وإفادة الذاكرة. بهذه الطريقة يصبح التدريب تعليما للحاكمية لا تعليما للفصاحة.

علاقة Tool-use evidence curriculum بالأمان أساسية. المسارات التي تحقق نتيجة صحيحة لكنها تكسر خصوصية أو تستخدم أداة بلا إذن لا تعد إيجابية. بهذا يفصل المشروع بين القدرة والحق في التنفيذ.

ينبغي أن يخرج Tool-use evidence curriculum بيانات صالحة لتحديث الروتر: memory_gain، human_override، risk_tax، latency، failure_rate، route_win_rate. بعد تراكم هذه البيانات يمكن الانتقال من routing rule-based إلى learned routing دون فقدان الشفافية.

Failure-to-memory distillation

كل فشل يتحول إلى memory update مشروط بدلا من الندم اللفظي.

هذا المنهج يرتبط بمصفوفة التدريب العامة: task → hidden question → delta → route → court → outcome → memory. لا تكفي له بيانات question/answer، بل يحتاج traces. كل trace يحمل سبب route، الأدوات المستعملة، verdict المحكمة، أثر العالم، وقرار تحديث الذاكرة. من دون trace لا يمكن تحويل المنهج إلى learning signal.

مؤشر النجاح في Failure-to-memory distillation هو transfer لا التحسن المحلي فقط. نريد أن يحسن النظام قدرته على مواجهة مهمة جديدة من نفس العائلة دون حفظ الشكل. لذلك تقاس النتائج عبر few-shot transfer، مقاومة التلوث، وفشل أقل عند اختلاف المصدر أو الأداة.

يجب أن يكون Failure-to-memory distillation قابلا للأبلشن. نقارن نموذجا مباشرا، ونموذجا مع سؤال فقط، ونموذجا مع Q-MCD، ونموذجا مع HAKIM ومحكمة وذاكرة. إذا لم يظهر التدرج في النتائج، فإما أن المنهج غير ناضج أو أن المهمة غير مناسبة.

في التدريب العملي، لا تُعطى المكافأة للجواب النهائي وحده. توزع المكافأة على صدق المصدر، سلامة الأداة، جودة route، قوة التفنيد، وإفادة الذاكرة. بهذه الطريقة يصبح التدريب تعليما للحاكمية لا تعليما للفصاحة.

علاقة Failure-to-memory distillation بالأمان أساسية. المسارات التي تحقق نتيجة صحيحة لكنها تكسر خصوصية أو تستخدم أداة بلا إذن لا تعد إيجابية. بهذا يفصل المشروع بين القدرة والحق في التنفيذ.

ينبغي أن يخرج Failure-to-memory distillation بيانات صالحة لتحديث الروتر: memory_gain، human_override، risk_tax، latency، failure_rate، route_win_rate. بعد تراكم هذه البيانات يمكن الانتقال من routing rule-based إلى learned routing دون فقدان الشفافية.

Safety and anti-hijack training

يواجه النظام prompt injection ومصادر مزيفة وطلبات خطرة، ويتعلم route آمن لا رفضا آليا فقط.

هذا المنهج يرتبط بمصفوفة التدريب العامة: task → hidden question → delta → route → court → outcome → memory. لا تكفي له بيانات question/answer، بل يحتاج traces. كل trace يحمل سبب route، الأدوات المستعملة، verdict المحكمة، أثر العالم، وقرار تحديث الذاكرة. من دون trace لا يمكن تحويل المنهج إلى learning signal.

مؤشر النجاح في Safety and anti-hijack training هو transfer لا التحسن المحلي فقط. نريد أن يحسن النظام قدرته على مواجهة مهمة جديدة من نفس العائلة دون حفظ الشكل. لذلك تقاس النتائج عبر few-shot transfer، مقاومة التلوث، وفشل أقل عند اختلاف المصدر أو الأداة.

يجب أن يكون Safety and anti-hijack training قابلا للأبلشن. نقارن نموذجا مباشرا، ونموذجا مع سؤال فقط، ونموذجا مع Q-MCD، ونموذجا مع HAKIM ومحكمة وذاكرة. إذا لم يظهر التدرج في النتائج، فإما أن المنهج غير ناضج أو أن المهمة غير مناسبة.

في التدريب العملي، لا تُعطى المكافأة للجواب النهائي وحده. توزع المكافأة على صدق المصدر، سلامة الأداة، جودة route، قوة التفنيد، وإفادة الذاكرة. بهذه الطريقة يصبح التدريب تعليما للحاكمية لا تعليما للفصاحة.

علاقة Safety and anti-hijack training بالأمان أساسية. المسارات التي تحقق نتيجة صحيحة لكنها تكسر خصوصية أو تستخدم أداة بلا إذن لا تعد إيجابية. بهذا يفصل المشروع بين القدرة والحق في التنفيذ.

ينبغي أن يخرج Safety and anti-hijack training بيانات صالحة لتحديث الروتر: memory_gain، human_override، risk_tax، latency، failure_rate، route_win_rate. بعد تراكم هذه البيانات يمكن الانتقال من routing rule-based إلى learned routing دون فقدان الشفافية.

World-effect reinforcement

يتعلم من أثر artifact أو اختبار أو قرار، لا من مظهر الجواب فقط.

هذا المنهج يرتبط بمصفوفة التدريب العامة: task → hidden question → delta → route → court → outcome → memory. لا تكفي له بيانات question/answer، بل يحتاج traces. كل trace يحمل سبب route، الأدوات المستعملة، verdict المحكمة، أثر العالم، وقرار تحديث الذاكرة. من دون trace لا يمكن تحويل المنهج إلى learning signal.

مؤشر النجاح في World-effect reinforcement هو transfer لا التحسن المحلي فقط. نريد أن يحسن النظام قدرته على مواجهة مهمة جديدة من نفس العائلة دون حفظ الشكل. لذلك تقاس النتائج عبر few-shot transfer، مقاومة التلوث، وفشل أقل عند اختلاف المصدر أو الأداة.

يجب أن يكون World-effect reinforcement قابلا للأبلشن. نقارن نموذجا مباشرا، ونموذجا مع سؤال فقط، ونموذجا مع Q-MCD، ونموذجا مع HAKIM ومحكمة وذاكرة. إذا لم يظهر التدرج في النتائج، فإما أن المنهج غير ناضج أو أن المهمة غير مناسبة.

في التدريب العملي، لا تُعطى المكافأة للجواب النهائي وحده. توزع المكافأة على صدق المصدر، سلامة الأداة، جودة route، قوة التفنيد، وإفادة الذاكرة. بهذه الطريقة يصبح التدريب تعليما للحاكمية لا تعليما للفصاحة.

علاقة World-effect reinforcement بالأمان أساسية. المسارات التي تحقق نتيجة صحيحة لكنها تكسر خصوصية أو تستخدم أداة بلا إذن لا تعد إيجابية. بهذا يفصل المشروع بين القدرة والحق في التنفيذ.

ينبغي أن يخرج World-effect reinforcement بيانات صالحة لتحديث الروتر: memory_gain، human_override، risk_tax، latency، failure_rate، route_win_rate. بعد تراكم هذه البيانات يمكن الانتقال من routing rule-based إلى learned routing دون فقدان الشفافية.

HyperSkill distillation

يعزل patterns الناجحة ويحولها إلى skills لها schema وmetric.

هذا المنهج يرتبط بمصفوفة التدريب العامة: task → hidden question → delta → route → court → outcome → memory. لا تكفي له بيانات question/answer، بل يحتاج traces. كل trace يحمل سبب route، الأدوات المستعملة، verdict المحكمة، أثر العالم، وقرار تحديث الذاكرة. من دون trace لا يمكن تحويل المنهج إلى learning signal.

مؤشر النجاح في HyperSkill distillation هو transfer لا التحسن المحلي فقط. نريد أن يحسن النظام قدرته على مواجهة مهمة جديدة من نفس العائلة دون حفظ الشكل. لذلك تقاس النتائج عبر few-shot transfer، مقاومة التلوث، وفشل أقل عند اختلاف المصدر أو الأداة.

يجب أن يكون HyperSkill distillation قابلا للأبلشن. نقارن نموذجا مباشرا، ونموذجا مع سؤال فقط، ونموذجا مع Q-MCD، ونموذجا مع HAKIM ومحكمة وذاكرة. إذا لم يظهر التدرج في النتائج، فإما أن المنهج غير ناضج أو أن المهمة غير مناسبة.

في التدريب العملي، لا تُعطى المكافأة للجواب النهائي وحده. توزع المكافأة على صدق المصدر، سلامة الأداة، جودة route، قوة التفنيد، وإفادة الذاكرة. بهذه الطريقة يصبح التدريب تعليما للحاكمية لا تعليما للفصاحة.

علاقة HyperSkill distillation بالأمان أساسية. المسارات التي تحقق نتيجة صحيحة لكنها تكسر خصوصية أو تستخدم أداة بلا إذن لا تعد إيجابية. بهذا يفصل المشروع بين القدرة والحق في التنفيذ.

ينبغي أن يخرج HyperSkill distillation بيانات صالحة لتحديث الروتر: memory_gain، human_override، risk_tax، latency، failure_rate، route_win_rate. بعد تراكم هذه البيانات يمكن الانتقال من routing rule-based إلى learned routing دون فقدان الشفافية.

Continuous evaluation memory

تُختبر الذاكرة نفسها: هل حسنت routing أم زادت التلوث؟

هذا المنهج يرتبط بمصفوفة التدريب العامة: task → hidden question → delta → route → court → outcome → memory. لا تكفي له بيانات question/answer، بل يحتاج traces. كل trace يحمل سبب route، الأدوات المستعملة، verdict المحكمة، أثر العالم، وقرار تحديث الذاكرة. من دون trace لا يمكن تحويل المنهج إلى learning signal.

مؤشر النجاح في Continuous evaluation memory هو transfer لا التحسن المحلي فقط. نريد أن يحسن النظام قدرته على مواجهة مهمة جديدة من نفس العائلة دون حفظ الشكل. لذلك تقاس النتائج عبر few-shot transfer، مقاومة التلوث، وفشل أقل عند اختلاف المصدر أو الأداة.

يجب أن يكون Continuous evaluation memory قابلا للأبلشن. نقارن نموذجا مباشرا، ونموذجا مع سؤال فقط، ونموذجا مع Q-MCD، ونموذجا مع HAKIM ومحكمة وذاكرة. إذا لم يظهر التدرج في النتائج، فإما أن المنهج غير ناضج أو أن المهمة غير مناسبة.

في التدريب العملي، لا تُعطى المكافأة للجواب النهائي وحده. توزع المكافأة على صدق المصدر، سلامة الأداة، جودة route، قوة التفنيد، وإفادة الذاكرة. بهذه الطريقة يصبح التدريب تعليما للحاكمية لا تعليما للفصاحة.

علاقة Continuous evaluation memory بالأمان أساسية. المسارات التي تحقق نتيجة صحيحة لكنها تكسر خصوصية أو تستخدم أداة بلا إذن لا تعد إيجابية. بهذا يفصل المشروع بين القدرة والحق في التنفيذ.

ينبغي أن يخرج Continuous evaluation memory بيانات صالحة لتحديث الروتر: memory_gain، human_override، risk_tax، latency، failure_rate، route_win_rate. بعد تراكم هذه البيانات يمكن الانتقال من routing rule-based إلى learned routing دون فقدان الشفافية.

Ablation discipline

كل نظرية يجب أن تحذف من النظام مرة لمعرفة أثرها. ما لا يغيّر شيئا لا يدخل kernel.

هذا المنهج يرتبط بمصفوفة التدريب العامة: task → hidden question → delta → route → court → outcome → memory. لا تكفي له بيانات question/answer، بل يحتاج traces. كل trace يحمل سبب route، الأدوات المستعملة، verdict المحكمة، أثر العالم، وقرار تحديث الذاكرة. من دون trace لا يمكن تحويل المنهج إلى learning signal.

مؤشر النجاح في Ablation discipline هو transfer لا التحسن المحلي فقط. نريد أن يحسن النظام قدرته على مواجهة مهمة جديدة من نفس العائلة دون حفظ الشكل. لذلك تقاس النتائج عبر few-shot transfer، مقاومة التلوث، وفشل أقل عند اختلاف المصدر أو الأداة.

يجب أن يكون Ablation discipline قابلا للأبلشن. نقارن نموذجا مباشرا، ونموذجا مع سؤال فقط، ونموذجا مع Q-MCD، ونموذجا مع HAKIM ومحكمة وذاكرة. إذا لم يظهر التدرج في النتائج، فإما أن المنهج غير ناضج أو أن المهمة غير مناسبة.

في التدريب العملي، لا تُعطى المكافأة للجواب النهائي وحده. توزع المكافأة على صدق المصدر، سلامة الأداة، جودة route، قوة التفنيد، وإفادة الذاكرة. بهذه الطريقة يصبح التدريب تعليما للحاكمية لا تعليما للفصاحة.

علاقة Ablation discipline بالأمان أساسية. المسارات التي تحقق نتيجة صحيحة لكنها تكسر خصوصية أو تستخدم أداة بلا إذن لا تعد إيجابية. بهذا يفصل المشروع بين القدرة والحق في التنفيذ.

ينبغي أن يخرج Ablation discipline بيانات صالحة لتحديث الروتر: memory_gain، human_override، risk_tax، latency، failure_rate، route_win_rate. بعد تراكم هذه البيانات يمكن الانتقال من routing rule-based إلى learned routing دون فقدان الشفافية.

الباب الخامس: الربط العلمي بين النظريات

الربط بين السؤال والروتر

السؤال الحاكم يحدد نوع الطريق، والروتر يترجم نوع الطريق إلى نماذج وأدوات. لا يستطيع Vascular Compute Router أن يعمل بذكاء إذا لم تصله كثافة السؤال، كما لا تفيد كثافة السؤال إن لم تتحول إلى route. لذلك يجب أن يمر كل task_state عبر question_score قبل route_score. العلاقة العلمية هنا سببية: سوء تقدير السؤال يسبب سوء توزيع compute، وسوء التوزيع يعيد إنتاج الجواب الضعيف ثم يلوث الذاكرة.

المشكلة التقنية التي تعالجها الربط بين السؤال والروتر هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ الربط بين السؤال والروتر تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع الربط بين السؤال والروتر داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب الربط بين السؤال والروتر عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن الربط بين السؤال والروتر حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ الربط بين السؤال والروتر شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود الربط بين السؤال والروتر مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة الربط بين السؤال والروتر بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة الربط بين السؤال والروتر بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. الربط بين السؤال والروتر تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان الربط بين السؤال والروتر من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

الربط بين المحكمة والذاكرة

المحكمة ليست آخر الطريق؛ verdict المحكمة يدخل الذاكرة كشرط. إذا قالت المحكمة إن الادعاء ضعيف بسبب مصدر واحد، يجب أن تحفظ الذاكرة هذا السبب لا مجرد النتيجة. وإذا عدلت المحكمة claim_rank، فيجب أن يتغير routing history. بهذا تصبح الذاكرة نتيجة نقد، لا مخزنا لما قاله النموذج.

المشكلة التقنية التي تعالجها الربط بين المحكمة والذاكرة هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ الربط بين المحكمة والذاكرة تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع الربط بين المحكمة والذاكرة داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب الربط بين المحكمة والذاكرة عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن الربط بين المحكمة والذاكرة حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ الربط بين المحكمة والذاكرة شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود الربط بين المحكمة والذاكرة مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة الربط بين المحكمة والذاكرة بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة الربط بين المحكمة والذاكرة بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. الربط بين المحكمة والذاكرة تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان الربط بين المحكمة والذاكرة من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

الربط بين GraphWorld وWorld-Effect

GraphWorld يبني عالم علاقات، World-Effect يقيس ماذا حدث بعد الفعل. من دون GraphWorld قد لا يعرف النظام ما الذي يجب أن يتغير، ومن دون World-Effect يبقى الرسم البياني نصا. الجمع بينهما يسمح بتعلم سببي: أي مسار من السؤال إلى الفعل أنتج الأثر؟ وأي علاقة كانت زخرفية؟

المشكلة التقنية التي تعالجها الربط بين GraphWorld وWorld-Effect هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ الربط بين GraphWorld وWorld-Effect تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها، فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع الربط بين GraphWorld وWorld-Effect داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب الربط بين GraphWorld وWorld-Effect عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن الربط بين GraphWorld وWorld-Effect حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ الربط بين GraphWorld وWorld-Effect شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود الربط بين GraphWorld وWorld-Effect مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة الربط بين GraphWorld وWorld-Effect بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة الربط بين GraphWorld وWorld-Effect بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. الربط بين GraphWorld وWorld-Effect تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان الربط بين GraphWorld وWorld-Effect من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

الربط بين HyperSkills وTraining

HyperSkill لا يولد من تعريف نظري بل من traces ناجحة متكررة. التدريب يستخرج من الأداء patterns مستقرة، ثم يحولها إلى skill لها state وfailure modes. لذلك HyperSkill Compiler يعتمد على Failure Distillation Lab وWorld-Effect Monitor. المهارة التي لا تقبل التفنيد لا تصعد إلى HyperSkill.

المشكلة التقنية التي تعالجها الربط بين HyperSkills وTraining هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ الربط بين HyperSkills وTraining تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها، فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع الربط بين HyperSkills وTraining داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب الربط بين HyperSkills وTraining عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن الربط بين HyperSkills وTraining حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ الربط بين HyperSkills وTraining شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود الربط بين HyperSkills وTraining مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة الربط بين HyperSkills وTraining بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة الربط بين HyperSkills وTraining بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. الربط بين HyperSkills وTraining تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان الربط بين HyperSkills وTraining من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

الربط بين Provider Sovereignty والأمان

الخصوصية ليست طبقة خارجية بل tax داخل route_score. إذا كان السياق حساسا، قد يختار النظام local model رغم أن hosted model أقوى. هذا يحقق فصل capability عن authority. السيادة المزودية تحفظ المعمارية من التعلق بمزود واحد، لكنها أيضا تحفظ الحاكمية من تسريب البيانات.

المشكلة التقنية التي تعالجها الربط بين Provider Sovereignty والأمان هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ الربط بين Provider Sovereignty والأمان تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع الربط بين Provider Sovereignty والأمان داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب الربط بين Provider Sovereignty والأمان عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن الربط بين Provider Sovereignty والأمان حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ الربط بين Provider Sovereignty والأمان شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود الربط بين Provider Sovereignty والأمان مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة الربط بين Provider Sovereignty والأمان بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة الربط بين Provider Sovereignty والأمان بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. الربط بين Provider Sovereignty والأمان تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان الربط بين Provider Sovereignty والأمان من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

الربط بين MCD والتكلفة

MCD ليس دعوة للتفكير أكثر دائما؛ هو اقتصاد تفكير. عندما يكون D وP وrisk منخفضين، route خفيف أفضل. عندما ترتفع C أو N أو A، يجب فتح محكمة أو GraphRAG. بذلك يتحول test-time compute من إسهاب إلى استثمار مشروط. قياس الفشل هنا هو compute waste: زمن زائد بلا تحسن في الأثر.

المشكلة التقنية التي تعالجها الربط بين MCD والتكلفة هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ الربط بين MCD والتكلفة تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع الربط بين MCD والتكلفة داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب الربط بين MCD والتكلفة عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن الربط بين MCD والتكلفة حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ الربط بين MCD والتكلفة شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود الربط بين MCD والتكلفة مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة الربط بين MCD والتكلفة بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة الربط بين MCD والتكلفة بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. الربط بين MCD والتكلفة تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان الربط بين MCD والتكلفة من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

الربط بين المقاييس والبنشماركات

البنشمارك لا يقيس الذكاء الفتّاح إذا قاس صحة نهائية فقط. يجب أن يقيس tool minimality، memory gain، source fidelity، contradiction density، route explainability، privacy compliance، وworld-effect closure. بهذا تتحول benchmarks إلى محكمة هندسية لا لوحة درجات دعائية.

المشكلة التقنية التي تعالجها الربط بين المقاييس والبنشماركات هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ الربط بين المقاييس والبنشماركات تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها، فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع الربط بين المقاييس والبنشماركات داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب الربط بين المقاييس والبنشماركات عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن الربط بين المقاييس والبنشماركات حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ الربط بين المقاييس والبنشماركات شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود الربط بين المقاييس والبنشماركات مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة الربط بين المقاييس والبنشماركات بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة الربط بين المقاييس والبنشماركات بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. الربط بين المقاييس والبنشماركات تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان الربط بين المقاييس والبنشماركات من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

الربط بين anti-sycophancy والحاكمية

الذكاء الفتّاح لا يساير المستخدم إذا كان السؤال خاطئا. Anti-Sycophancy Evaluator يقيس هل اختار النظام الصدق مع احتمال انخفاض الرضا. هذا يرتبط بالحاكمية لأن النظام يجب أن يعرف متى يكون المستخدم مصدر هدف لا مصدر حقيقة. في هذه اللحظة يصبح سؤال L4 ضروريا: من يملك حق الحكم؟

المشكلة التقنية التي تعالجها الربط بين anti-sycophancy والحاكمية هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ الربط بين anti-sycophancy والحاكمية تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها، فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع الربط بين anti-sycophancy والحاكمية داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب الربط بين anti-sycophancy والحاكمية عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن الربط بين anti-sycophancy والحاكمية حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ الربط بين anti-sycophancy والحاكمية شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود الربط بين anti-sycophancy والحاكمية مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة الربط بين anti-sycophancy والحاكمية بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة الربط بين anti-sycophancy والحاكمية بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. الربط بين anti-sycophancy والحاكمية تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان الربط بين anti-sycophancy والحاكمية من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

الربط بين النظرية والتطبيق

كل نظرية في المشروع يجب أن تنتهي إلى artifact: ملف schema، دالة score، محكمة، route policy، أو benchmark task. إذا بقيت النظرية بلا artifact فهي بحث قيد التشكل. لذلك تقترح الورقة مسارا تنفيذيا يبدأ بـminimum viable Hermes: provider adapter، registry، route_score، worker، court، memory، security gate.

المشكلة التقنية التي تعالجها الربط بين النظرية والتطبيق هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ الربط بين النظرية والتطبيق تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها، فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع الربط بين النظرية والتطبيق داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب الربط بين النظرية والتطبيق عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن الربط بين النظرية والتطبيق حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ الربط بين النظرية والتطبيق شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود الربط بين النظرية والتطبيق مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة الربط بين النظرية والتطبيق بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة الربط بين النظرية والتطبيق بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. الربط بين النظرية والتطبيق تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان الربط بين النظرية والتطبيق من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

الربط بين corpus وHyperKB

الكتب السابقة نفسها يجب أن تصبح HyperKB. كل صفحة تتحول إلى reference، failure mode، metric، tool، theory، claim. بذلك يختبر المشروع نفسه على ذاته: هل يستطيع أن يبني قاعدة معرفة حاكمية من كتبه؟ إذا نجح، يصبح corpus المشروع أول عالم اختبار للنظرية.

المشكلة التقنية التي تعالجها الربط بين corpus وHyperKB هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ الربط بين corpus وHyperKB تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها، فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع الربط بين corpus وHyperKB داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب الربط بين corpus وHyperKB عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن الربط بين corpus وHyperKB حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ الربط بين corpus وHyperKB شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود الربط بين corpus وHyperKB مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة الربط بين corpus وHyperKB بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة الربط بين corpus وHyperKB بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. الربط بين corpus وHyperKB تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان الربط بين corpus وHyperKB من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

الباب السادس: المواصفات التنفيذية والملحقات

State Schema

يقترح النظام state موحدا: task_id، raw_intent، hidden_question، question_layer، delta_map، mcd_vector، risk_tier، privacy_tier، route_plan، tool_permissions، agent_outputs، court_verdicts، fusion_confidence، world_effect، memory_update. هذا schema يجعل كل طبقة قابلة للتتبع. عند الفشل لا نقول "النموذج أخطأ" بل نعرف أي field انحرف.

المشكلة التقنية التي تعالجها State Schema هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ State Schema تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع State Schema داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب State Schema عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن State Schema حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ State Schema شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: court_requirement، route_constraint، delta_score، question_rank، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود State Schema مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة State Schema بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة State Schema بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. State Schema تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان State Schema من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

API Kernel

الحد الأدنى البرمجي: POST /ask يستقبل eval/، memory_update/، fuse/، court/، route/، score_mcd/، task_state/. يجب أن تكون كل خدمة قابلة للتبديل حتى تختبر الأبلشن. لا تبنى واجهة المستخدم قبل أن تنضج هذه الخدمات.

المشكلة التقنية التي تعالجها API Kernel هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ API Kernel تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع API Kernel داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب API Kernel عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن API Kernel حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ API Kernel شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: court_requirement، route_constraint، delta_score، question_rank، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود API Kernel مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة API Kernel بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط، بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة API Kernel بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. API Kernel تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان API Kernel من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

Evaluation Matrix

المصفوفة المقترحة تشمل: privacy_compliance، verifier_pass، contradiction_density، hallucination_rate، route_efficiency، source_fidelity، task_success، world_effect_closure، transfer_score، memory_usefulness، tool_minimality. كل metric يعرف domain وthreshold وfailure interpretation.

المشكلة التقنية التي تعالجها Evaluation Matrix هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ Evaluation Matrix تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع Evaluation Matrix داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب Evaluation Matrix عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن Evaluation Matrix حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ Evaluation Matrix شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: route_constraint، delta_score، question_rank، court_requirement، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود Evaluation Matrix مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة Evaluation Matrix بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط؛ بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة Evaluation Matrix بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. Evaluation Matrix تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان Evaluation Matrix من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

Roadmap

المرحلة الأولى corpus وgolden tasks. الثانية question/difference modules. الثالثة MCD وroute. الرابعة court وGraphRAG. الخامسة memory. السادسة learned routing. السابعة production governance. كل مرحلة تنتهي بـacceptance criteria لا بعبارة تقدم عامة.

المشكلة التقنية التي تعالجها Roadmap هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ Roadmap تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع Roadmap داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب Roadmap عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن Roadmap حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ Roadmap شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: court_requirement، route_constraint، delta_score، question_rank، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود Roadmap مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة Roadmap بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط؛ بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة Roadmap بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. Roadmap تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان Roadmap من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

جدول عدم الضياع النظري

النظريات التي يجب أن تبقى حاضرة: السؤال الحاكم، الكثافة السؤالية، HAKIM-MCD، Q-MCD، التراكب الوعائي، الاختطاف الوعائي، HyperSkills، VCGE-M، EIG، ITC، VDG، Anti-Hijack Governance، Provider Sovereignty، Self-Reconstruction Memory، Falsification-Centric Compute، Hermes-Fugu، Hyperloops. كل واحدة ربطت في هذه النسخة بأداة ومقياس وفشل.

المشكلة التقنية التي تعالجها جدول عدم الضياع النظري هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ جدول عدم الضياع النظري تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع جدول عدم الضياع النظري داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب جدول عدم الضياع النظري عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن جدول عدم الضياع النظري حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ جدول عدم الضياع النظري شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: court_requirement، route_constraint، delta_score، question_rank، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود جدول عدم الضياع النظري مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة جدول عدم الضياع النظري بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط؛ بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة جدول عدم الضياع النظري بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. جدول عدم الضياع النظري تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان جدول عدم الضياع النظري من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

خاتمة تقنية

النتيجة النهائية أن الذكاء الصناعي الفتّاح في هذه النظرية ليس مسارا واحدا بل هندسة حاكمية: السؤال يفتح، الفرق يضغط، MCD يقيس، HAKIM يحكم، Hermes ينسق، المحكمة تهدم، الذاكرة تعيد البناء، والأثر العالمي يختبر. القوة ليست في كثرة الأسماء بل في أن كل اسم يمكن أن يصبح كودا أو اختبارا أو سياسة تنفيذية.

المشكلة التقنية التي تعالجها خاتمة تقنية هي أن النموذج اللغوي يميل إلى تحويل كل شيء إلى جواب لغوي. النظرية تعيد بناء المسألة كحالة تشغيل: ما السؤال؟ ما الفرق؟ ما الدليل؟ ما الأداة؟ ما الخطر؟ وما الأثر؟ بهذا لا تبقى النظرية في مستوى التأمل، بل تتحول إلى قرار route وmetric وmemory.

الآلية الداخلية لـ خاتمة تقنية تمر بثلاث طبقات. الطبقة الأولى تشخيصية: تحدد محل الغموض أو الفرق. الطبقة الثانية حاكمية: تقرر أي وعاء يملك حق التقدم. الطبقة الثالثة اختبارية: تضع شروط سقوط النظرية داخل المهمة نفسها. إذا غابت إحدى هذه الطبقات تحولت النظرية إلى تسمية لا إلى معمارية.

العلاقة العلمية لهذه النظرية ببقية النظام تبدأ من أنها لا تعمل وحدها؛ فهي تلتقي مع Q-MCD في تقدير الكلفة، ومع HAKIM-MCD في توزيع سلطة الحكم، ومع Hermes-Fugu في تحويلها إلى route قابل للتنفيذ. عندما نضع خاتمة تقنية داخل النظام، يجب أن يظهر أثرها في واحد من أربعة: خفض الهلوسة، رفع صدق المصدر، تحسين اختيار الأداة، أو زيادة جودة ذاكرة إعادة البناء.

في طبقة التدريب، لا تُدرّب خاتمة تقنية عبر أمثلة سؤال وجواب عادية، بل عبر traces. كل trace يذكر السؤال الحاكم، الفرق المستخرج، route المختار، نتيجة المحكمة، أثر العالم، وما الذي كُتب في الذاكرة. هكذا يتعلم النظام علاقة السبب بالأثر بدلا من حفظ الصياغة.

في طبقة القياس، لا يكفي أن نقول إن خاتمة تقنية حسنت النتيجة. يجب قياس: الفرق في route_accuracy، انخفاض contradiction_density، ارتفاع source_fidelity، خفض tool_overuse، وزيادة world_effect_closure. كل مقياس يجب أن يقارن مع baseline مباشر.

في طبقة التنفيذ، تأخذ خاتمة تقنية شكل دالة أو خدمة. الدالة تستقبل state object وتعيد تعديلا محددا: court_requirement، route_constraint، delta_score، question_rank، أو memory_patch. إذا لم يمكن تمثيلها بهذا الشكل فهي لا تزال غير جاهزة للبرمجة.

حدود خاتمة تقنية مهمة بقدر قوتها. قد تفشل إذا استعملت في مهمة بسيطة منخفضة الخطر، أو إذا تحولت إلى طبقة زائدة لا تغير قرارا، أو إذا استعملها النموذج لتبرير إسهاب لا لرفع الدقة. لذلك توضع في معمارية conditional activation: لا تعمل إلا عندما يتجاوز الغموض أو الخطر أو ضغط الأثر عتبة محددة.

علاقة خاتمة تقنية بمحكمة الهدم مركزية. المحكمة لا تأتي بعد بناء الجواب فقط؛ بل تهاجم افتراض النظرية نفسه داخل المهمة: هل كان السؤال يستحق هذا العمق؟ هل كان الفرق منتجا؟ هل قادت النظرية إلى أداة فعالة أم زادت التعقيد بلا عائد؟ بهذا تصبح النظرية قابلة للهجوم لا للتقديس.

علاقة خاتمة تقنية بالذاكرة تظهر بعد التنفيذ. لا تكتب الذاكرة أن النظرية استعملت فقط، بل تكتب هل غيرت القرار وماذا كان أثرها. إذا استعملت النظرية عشر مرات ولم تغير route أو court أو answer quality، تخفض رتبتها أو تُحذف من المسار الافتراضي.

مثال تطبيقي: في مهمة بحث تقنية غامضة، يحدد النظام السؤال الخفي، يستخرج فروقا بين المطلوب والمصادر، يحسب MCD، ثم يقرر هل يحتاج GraphRAG أو verifier أو أداة كود. خاتمة تقنية تظهر قيمتها إذا جعلت هذا القرار أدق من جواب مباشر.

في النسخة المستقبلية، يمكن تعلم أوزان خاتمة تقنية من بيانات المسار. يبدأ النظام بقواعد شفافة، ثم يربط النجاح والفشل بالأوزان، مع بقاء الخصوصية والصلاحيات ومحكمة المصدر قواعد صريحة لا يبتلعها نموذج عصبي غامض.

الباب السابع: أنماط تنفيذية مفصلة لملء الثغرات التقنية

نمط 1: Question provenance lock

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Question provenance lock لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Question provenance lock بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Question provenance lock يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Question provenance lock يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 2: Evidence budget allocator

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Evidence budget allocator لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Evidence budget allocator بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Evidence budget allocator يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Evidence budget allocator يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 3: Contradiction density audit

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Contradiction density audit لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Contradiction density audit بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Contradiction density audit يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Contradiction density audit يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 4: Privacy-tier dispatch

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Privacy-tier dispatch لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Privacy-tier dispatch بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Privacy-tier dispatch يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Privacy-tier dispatch يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 5: Local-only sensitive route

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Local-only sensitive route لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Local-only sensitive route بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Local-only sensitive route يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Local-only sensitive route يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 6: Human approval breakpoint

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Human approval breakpoint لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Human approval breakpoint بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Human approval breakpoint يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Human approval breakpoint يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 7: Rollback-ready action gate

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Rollback-ready action gate لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Rollback-ready action gate بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Rollback-ready action gate يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Rollback-ready action gate يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 8: Tool minimality selector

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Tool minimality selector لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Tool minimality selector بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Tool minimality selector يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Tool minimality selector يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 9: Source freshness ladder

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Source freshness ladder لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Source freshness ladder بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Source freshness ladder يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Source freshness ladder يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 10: Memory contamination detector

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Memory contamination detector لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Memory contamination detector بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Memory contamination detector يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Memory contamination detector يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 11: Route regret monitor

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Route regret monitor لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Route regret monitor بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Route regret monitor يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Route regret monitor يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 12: Provider health watcher

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Provider health watcher لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Provider health watcher بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Provider health watcher يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Provider health watcher يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 13: Capability card updater

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Capability card updater لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Capability card updater بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Capability card updater يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Capability card updater يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 14: GraphRAG citation fidelity

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في GraphRAG citation fidelity لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل GraphRAG citation fidelity بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس GraphRAG citation fidelity يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل GraphRAG citation fidelity يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 15: Court independence check

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Court independence check لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Court independence check بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Court independence check يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Court independence check يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 16: Verifier disagreement escalation

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Verifier disagreement escalation لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Verifier disagreement escalation بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Verifier disagreement escalation يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Verifier disagreement escalation يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 17: Fusion downgrade rule

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Fusion downgrade rule لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Fusion downgrade rule بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Fusion downgrade rule يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Fusion downgrade rule يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 18: World-effect simulator

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في World-effect simulator لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل World-effect simulator بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس World-effect simulator يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل World-effect simulator يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 19: Causal-question graph

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Causal-question graph لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Causal-question graph بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Causal-question graph يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Causal-question graph يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 20: MCD heatmap diagnostic

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في MCD heatmap diagnostic لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل MCD heatmap diagnostic بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس MCD heatmap diagnostic يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل MCD heatmap diagnostic يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 21: Failure taxonomy writer

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Failure taxonomy writer لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Failure taxonomy writer بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Failure taxonomy writer يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Failure taxonomy writer يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 22: Claim-rank downgrader

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Claim-rank downgrader لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Claim-rank downgrader بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Claim-rank downgrader يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Claim-rank downgrader يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 23: Hypothesis-to-test compiler

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Hypothesis-to-test compiler لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Hypothesis-to-test compiler بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Hypothesis-to-test compiler يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Hypothesis-to-test compiler يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 24: Domain adapter builder

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Domain adapter builder لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Domain adapter builder بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Domain adapter builder يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Domain adapter builder يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 25: Prompt contract compiler

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Prompt contract compiler لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Prompt contract compiler بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Prompt contract compiler يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Prompt contract compiler يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 26: Agent role separation

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Agent role separation لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Agent role separation بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Agent role separation يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Agent role separation يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 27: Sandbox dry-run executor

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Sandbox dry-run executor لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Sandbox dry-run executor بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Sandbox dry-run executor يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Sandbox dry-run executor يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 28: Evidence-to-memory distiller

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Evidence-to-memory distiller لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Evidence-to-memory distiller بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Evidence-to-memory distiller يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Evidence-to-memory distiller يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 29: Self-hosted migration guard

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Self-hosted migration guard لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Self-hosted migration guard بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Self-hosted migration guard يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Self-hosted migration guard يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 30: OpenRouter exploration mode

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في OpenRouter exploration mode لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل OpenRouter exploration mode بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس OpenRouter exploration mode يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل OpenRouter exploration mode يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 31: NIM production bridge

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في NIM production bridge لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل NIM production bridge بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس NIM production bridge يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل NIM production bridge يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 32: Model pool drift audit

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Model pool drift audit لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Model pool drift audit بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Model pool drift audit يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Model pool drift audit يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 33: Anti-sycophancy pressure test

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Anti-sycophancy pressure test لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Anti-sycophancy pressure test بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Anti-sycophancy pressure test يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Anti-sycophancy pressure test يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 34: Benchmark mutation generator

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Benchmark mutation generator لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Benchmark mutation generator بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Benchmark mutation generator يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Benchmark mutation generator يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 35: Golden task curator

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Golden task curator لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Golden task curator بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Golden task curator يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Golden task curator يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 36: Transfer-shot evaluator

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Transfer-shot evaluator لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Transfer-shot evaluator بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Transfer-shot evaluator يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Transfer-shot evaluator يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 37: Noise tax estimator

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Noise tax estimator لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل، وإلا فهو وصف لا يدخل النظام.

يدخل Noise tax estimator بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Noise tax estimator يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Noise tax estimator يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 38: Inventive entropy filter

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Inventive entropy filter لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل، وإلا فهو وصف لا يدخل النظام.

يدخل Inventive entropy filter بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Inventive entropy filter يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Inventive entropy filter يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 39: Selective rescue governor

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Selective rescue governor لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل، وإلا فهو وصف لا يدخل النظام.

يدخل Selective rescue governor بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Selective rescue governor يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Selective rescue governor يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 40: Hidden ambiguity expander

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Hidden ambiguity expander لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل، وإلا فهو وصف لا يدخل النظام.

يدخل Hidden ambiguity expander بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Hidden ambiguity expander يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Hidden ambiguity expander يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 41: Question collapse gate

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Question collapse gate لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل، وإلا فهو وصف لا يدخل النظام.

يدخل Question collapse gate بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Question collapse gate يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Question collapse gate يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 42: Higher-question generator

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Higher-question generator لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل، وإلا فهو وصف لا يدخل النظام.

يدخل Higher-question generator بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Higher-question generator يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Higher-question generator يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 43: Causal replay scheduler

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Causal replay scheduler لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل، وإلا فهو وصف لا يدخل النظام.

يدخل Causal replay scheduler بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Causal replay scheduler يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Causal replay scheduler يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 44: Long-session identity guard

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Long-session identity guard لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل، وإلا فهو وصف لا يدخل النظام.

يدخل Long-session identity guard بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Long-session identity guard يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Long-session identity guard يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 45: Memory utility judge

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Memory utility judge لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل، وإلا فهو وصف لا يدخل النظام.

يدخل Memory utility judge بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Memory utility judge يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Memory utility judge يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 46: Court latency budget

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Court latency budget لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل، وإلا فهو وصف لا يدخل النظام.

يدخل Court latency budget بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Court latency budget يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Court latency budget يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 47: Tool permission schema

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Tool permission schema لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل، وإلا فهو وصف لا يدخل النظام.

يدخل Tool permission schema بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Tool permission schema يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Tool permission schema يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 48: Falsification prompt contract

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Falsification prompt contract لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل، وإلا فهو وصف لا يدخل النظام.

يدخل Falsification prompt contract بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Falsification prompt contract يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Falsification prompt contract يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 49: Answer-action separator

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Answer-action separator لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Answer-action separator بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Answer-action separator يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Answer-action separator يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 50: World-effect rollback path

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في World-effect rollback path لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل World-effect rollback path بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس World-effect rollback path يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل World-effect rollback path يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 51: Epistemic immune quarantine

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Epistemic immune quarantine لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Epistemic immune quarantine بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Epistemic immune quarantine يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Epistemic immune quarantine يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 52: Route explanation generator

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Route explanation generator لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Route explanation generator بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Route explanation generator يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Route explanation generator يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 53: Training trace normalizer

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Training trace normalizer لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Training trace normalizer بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Training trace normalizer يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Training trace normalizer يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 54: Preference pair builder

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Preference pair builder لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Preference pair builder بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Preference pair builder يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Preference pair builder يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 55: Verifier reward logger

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Verifier reward logger لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Verifier reward logger بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Verifier reward logger يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Verifier reward logger يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 56: Deployment readiness gate

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Deployment readiness gate لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Deployment readiness gate بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Deployment readiness gate يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Deployment readiness gate يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 57: Production incident postmortem

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Production incident postmortem لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Production incident postmortem بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Production incident postmortem يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Production incident postmortem يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 58: Governance drift auditor

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Governance drift auditor لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Governance drift auditor بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Governance drift auditor يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Governance drift auditor يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 59: MCP tool wrapper

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في MCP tool wrapper لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل MCP tool wrapper بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس MCP tool wrapper يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل MCP tool wrapper يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

نمط 60: Registry schema maintainer

هذا النمط يضيف تفصيلا تنفيذيا إلى المعمارية لأنه يحول مفهوما عاما إلى سياسة قابلة للبرمجة. في Registry schema maintainer لا نكتفي بتسمية الوظيفة، بل نحدد متى تعمل، ما بياناتها، ما أثرها في route، وما الذي يحدث إذا فشلت. كل نمط يجب أن يترك أثرا في السجل؛ وإلا فهو وصف لا يدخل النظام.

يدخل Registry schema maintainer بعد قراءة Q-MCD وقبل أو بعد المحكمة بحسب موقعه. إذا كان النمط يتعلق بالمصدر فإنه يعمل قبل الاسترجاع وبعده، وإذا كان يتعلق بالأداة فإنه يعمل قبل tool call، وإذا كان يتعلق بالذاكرة فإنه يعمل بعد world-effect. هذا التحديد يمنع تداخل الصلاحيات بين العامل المنتج والعامل الحاكم.

مقياس Registry schema maintainer يتكون من ثلاث عائلات: أثر معرفي مثل source_fidelity أو contradiction_density، أثر تشغيلي مثل latency أو cost أو route_success، وأثر حاكمي مثل privacy_compliance أو claim_rank_accuracy. بهذه الطريقة لا يصبح النمط ناجحا لأنه كتب نصا جيدا، بل لأنه حسّن قرارا قابلا للقياس.

فشل Registry schema maintainer يسجل في Agentic Autopsy Notebook. لا يكتب السجل أن النمط فشل فقط، بل يكتب سبب الفشل: threshold خاطئ، مصدر ملوث، route مبالغ فيه، أداة بلا صلاحية، أو ذاكرة غير نافعة. بعد ذلك يقترح النظام إصلاحا: تعديل وزن، تغيير gate، إضافة verifier، أو نقل المسار إلى human approval.

الباب الثامن: برنامج بحوث ومشاريع تنفيذية لاختبار النظريات

مشروع بحثي 1: Q-MCD scorer open benchmark

مشروع Q-MCD scorer open benchmark يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Q-MCD scorer open benchmark ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Q-MCD scorer open benchmark يتم بمصفوفة موحدة: privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، tool_minimality، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Q-MCD scorer open benchmark أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 2: Hidden question dataset

مشروع Hidden question dataset يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Hidden question dataset ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Hidden question dataset يتم بمصفوفة موحدة: privacy_compliance، tool_minimality، route_efficiency، contradiction_density، source_fidelity، correctness، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Hidden question dataset أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 3: Agent baseline against Q-MCD protocol

مشروع Agent baseline against Q-MCD protocol يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Agent baseline against Q-MCD protocol ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Agent baseline against Q-MCD protocol يتم بمصفوفة موحدة: privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، tool_minimality، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Agent baseline against Q-MCD protocol أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 4: GraphRAG governance map

مشروع GraphRAG governance map يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ GraphRAG governance map ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم GraphRAG governance map يتم بمصفوفة موحدة: privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، tool_minimality، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ GraphRAG governance map أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 5: Anti-sycophancy eval suite

مشروع Anti-sycophancy eval suite يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Anti-sycophancy eval suite ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Anti-sycophancy eval suite يتم بمصفوفة موحدة: tool_minimality، privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Anti-sycophancy eval suite أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 6: Tool-minimality benchmark

مشروع Tool-minimality benchmark يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Tool-minimality benchmark ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Tool-minimality benchmark يتم بمصفوفة موحدة: tool_minimality، privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Tool-minimality benchmark أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 7: World-effect simulator

مشروع World-effect simulator يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ World-effect simulator ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم World-effect simulator يتم بمصفوفة موحدة: tool_minimality، privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ World-effect simulator أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 8: MCD heatmap UI

مشروع MCD heatmap UI يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ MCD heatmap UI ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم MCD heatmap UI يتم بمصفوفة موحدة: tool_minimality، privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ MCD heatmap UI أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 9: Causal-question graph

مشروع Causal-question graph يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Causal-question graph ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Causal-question graph يتم بمصفوفة موحدة: tool_minimality، privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Causal-question graph أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 10: Agentic autopsy notebook

مشروع Agentic autopsy notebook يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Agentic autopsy notebook ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Agentic autopsy notebook يتم بمصفوفة موحدة: tool_minimality، privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Agentic autopsy notebook أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 11: Arabic high-governance QA

مشروع Arabic high-governance QA يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Arabic high-governance QA ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Arabic high-governance QA يتم بمصفوفة موحدة: tool_minimality، privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Arabic high-governance QA أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 12: Paper ablation study

مشروع Paper ablation study يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Paper ablation study ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Paper ablation study يتم بمصفوفة موحدة: tool_minimality، privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Paper ablation study أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 13: Vascular route simulator

مشروع Vascular route simulator يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Vascular route simulator ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Vascular route simulator يتم بمصفوفة موحدة: tool_minimality، privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Vascular route simulator أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 14: Provider sovereignty stress test

مشروع Provider sovereignty stress test يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Provider sovereignty stress test ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Provider sovereignty stress test يتم بمصفوفة موحدة: privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، tool_minimality، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Provider sovereignty stress test أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 15: Memory contamination benchmark

مشروع Memory contamination benchmark يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Memory contamination benchmark ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Memory contamination benchmark يتم بمصفوفة موحدة: privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، tool_minimality، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Memory contamination benchmark أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 16: Court hallucination detector

مشروع Court hallucination detector يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Court hallucination detector ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Court hallucination detector يتم بمصفوفة موحدة: tool_minimality، privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Court hallucination detector أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 17: Hyperskill compiler prototype

مشروع Hyperskill compiler prototype يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Hyperskill compiler prototype ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Hyperskill compiler prototype يتم بمصفوفة موحدة: privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، tool_minimality، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Hyperskill compiler prototype أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 18: Hyperloop closure tracker

مشروع Hyperloop closure tracker يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Hyperloop closure tracker ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Hyperloop closure tracker يتم بمصفوفة موحدة: correctness، source_fidelity، route_efficiency، contradiction_density، privacy_compliance، tool_minimality، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Hyperloop closure tracker أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 19: Failure-to-memory distillation set

مشروع Failure-to-memory distillation set يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Failure-to-memory distillation set ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Failure-to-memory distillation set يتم بمصفوفة موحدة: correctness، source_fidelity، route_efficiency، contradiction_density، privacy_compliance، memory_gain، tool_minimality، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Failure-to-memory distillation set أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 20: Governance reward model dataset

مشروع Governance reward model dataset يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Governance reward model dataset ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Governance reward model dataset يتم بمصفوفة موحدة: correctness، source_fidelity، route_efficiency، contradiction_density، privacy_compliance، memory_gain، tool_minimality، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Governance reward model dataset أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 21: Pairwise route preference set

مشروع Pairwise route preference set يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Pairwise route preference set ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Pairwise route preference set يتم بمصفوفة موحدة: correctness، source_fidelity، route_efficiency، contradiction_density، privacy_compliance، tool_minimality، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Pairwise route preference set أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 22: Verifier swarm comparison

مشروع Verifier swarm comparison يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Verifier swarm comparison ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Verifier swarm comparison يتم بمصفوفة موحدة: correctness، source_fidelity، route_efficiency، contradiction_density، privacy_compliance، tool_minimality، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Verifier swarm comparison أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 23: Privacy-tier routing benchmark

مشروع Privacy-tier routing benchmark يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Privacy-tier routing benchmark ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Privacy-tier routing benchmark يتم بمصفوفة موحدة: correctness، source_fidelity، route_efficiency، contradiction_density، privacy_compliance، memory_gain، tool_minimality، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Privacy-tier routing benchmark أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 24: NIM migration trial

مشروع NIM migration trial يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ NIM migration trial ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم NIM migration trial يتم بمصفوفة موحدة: tool_minimality، privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ NIM migration trial أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 25: Open-weight local route trial

مشروع Open-weight local route trial يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Open-weight local route trial ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Open-weight local route trial يتم بمصفوفة موحدة: tool_minimality، privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Open-weight local route trial أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 26: Fusion confidence calibration

مشروع Fusion confidence calibration يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Fusion confidence calibration ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Fusion confidence calibration يتم بمصفوفة موحدة: tool_minimality، privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Fusion confidence calibration أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 27: Contradiction density calibration

مشروع Contradiction density calibration يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Contradiction density calibration ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Contradiction density calibration يتم بمصفوفة موحدة: privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، tool_minimality، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Contradiction density calibration أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 28: Source freshness decay model

مشروع Source freshness decay model يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Source freshness decay model ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Source freshness decay model يتم بمصفوفة موحدة: privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، tool_minimality، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Source freshness decay model أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 29: Claim-rank training set

مشروع Claim-rank training set يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Claim-rank training set ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Claim-rank training set يتم بمصفوفة موحدة: tool_minimality، privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Claim-rank training set أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 30: Tool sandbox rollback test

مشروع Tool sandbox rollback test يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Tool sandbox rollback test ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Tool sandbox rollback test يتم بمصفوفة موحدة: tool_minimality، privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Tool sandbox rollback test أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 31: Causal replay laboratory

مشروع Causal replay laboratory يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Causal replay laboratory ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Causal replay laboratory يتم بمصفوفة موحدة: tool_minimality، privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Causal replay laboratory أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 32: Selective rescue ablation

مشروع Selective rescue ablation يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Selective rescue ablation ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Selective rescue ablation يتم بمصفوفة موحدة: tool_minimality، privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Selective rescue ablation أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 33: Internal-time budget study

مشروع Internal-time budget study يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Internal-time budget study ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Internal-time budget study يتم بمصفوفة موحدة: tool_minimality، privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Internal-time budget study أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 34: Novelty entropy benchmark

مشروع Novelty entropy benchmark يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Novelty entropy benchmark ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Novelty entropy benchmark يتم بمصفوفة موحدة: tool_minimality، privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Novelty entropy benchmark أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 35: GraphWorld causal memory test

مشروع GraphWorld causal memory test يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ GraphWorld causal memory test ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم GraphWorld causal memory test يتم بمصفوفة موحدة: privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، tool_minimality، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ GraphWorld causal memory test أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 36: Multi-agent role contract study

مشروع Multi-agent role contract study يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Multi-agent role contract study ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Multi-agent role contract study يتم بمصفوفة موحدة: privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، tool_minimality، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Multi-agent role contract study أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 37: Domain adapter registry

مشروع Domain adapter registry يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Domain adapter registry ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Domain adapter registry يتم بمصفوفة موحدة: privacy_compliance، tool_minimality، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Domain adapter registry أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 38: Prompt-schema-repair loop

مشروع Prompt-schema-repair loop يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Prompt-schema-repair loop ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Prompt-schema-repair loop يتم بمصفوفة موحدة: privacy_compliance، tool_minimality، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Prompt-schema-repair loop أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 39: Human gate calibration

مشروع Human gate calibration يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Human gate calibration ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Human gate calibration يتم بمصفوفة موحدة: privacy_compliance، tool_minimality، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Human gate calibration أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 40: Risk-tax routing analysis

مشروع Risk-tax routing analysis يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Risk-tax routing analysis ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Risk-tax routing analysis يتم بمصفوفة موحدة: privacy_compliance، tool_minimality، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Risk-tax routing analysis أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 41: Benchmark anti-gaming mutation set

مشروع Benchmark anti-gaming mutation set يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Benchmark anti-gaming mutation set ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Benchmark anti-gaming mutation set يتم بمصفوفة موحدة: privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، tool_minimality، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Benchmark anti-gaming mutation set أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 42: World-effect artifact grading

مشروع World-effect artifact grading يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ World-effect artifact grading ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم World-effect artifact grading يتم بمصفوفة موحدة: correctness، source_fidelity، route_efficiency، contradiction_density، privacy_compliance، tool_minimality، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ World-effect artifact grading أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 43: Route regret long-session study

مشروع Route regret long-session study يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Route regret long-session study ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Route regret long-session study يتم بمصفوفة موحدة: correctness، source_fidelity، route_efficiency، contradiction_density، privacy_compliance، memory_gain، tool_minimality، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Route regret long-session study أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 44: MCP wrapper governance test

مشروع MCP wrapper governance test يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ MCP wrapper governance test ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم MCP wrapper governance test يتم بمصفوفة موحدة: correctness، source_fidelity، route_efficiency، contradiction_density، privacy_compliance، memory_gain، tool_minimality، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ MCP wrapper governance test أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 45: Golden tasks corpus builder

مشروع Golden tasks corpus builder يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Golden tasks corpus builder ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Golden tasks corpus builder يتم بمصفوفة موحدة: correctness، source_fidelity، route_efficiency، contradiction_density، privacy_compliance، tool_minimality، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Golden tasks corpus builder أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 46: Production incident memory loop

مشروع Production incident memory loop يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Production incident memory loop ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Production incident memory loop يتم بمصفوفة موحدة: correctness، source_fidelity، route_efficiency، contradiction_density، privacy_compliance، memory_gain، tool_minimality، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Production incident memory loop أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 47: Safety trace normalization

مشروع Safety trace normalization يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Safety trace normalization ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Safety trace normalization يتم بمصفوفة موحدة: correctness، source_fidelity، route_efficiency، contradiction_density، privacy_compliance، tool_minimality، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Safety trace normalization أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 48: Capability-card updater test

مشروع Capability-card updater test يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Capability-card updater test ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Capability-card updater test يتم بمصفوفة موحدة: tool_minimality، privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Capability-card updater test أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 49: Model pool drift benchmark

مشروع Model pool drift benchmark يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Model pool drift benchmark ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Model pool drift benchmark يتم بمصفوفة موحدة: tool_minimality، privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Model pool drift benchmark أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

مشروع بحثي 50: Knowledge-base self-application

مشروع Knowledge-base self-application يحول النظرية إلى تجربة صغيرة يمكن قياسها. يبدأ بتحديد baseline مباشر لا يستخدم Q-MCD أو HAKIM، ثم نسخة ثانية تستخدم السؤال الحاكم، ثم نسخة ثالثة تستخدم route والمحكمة والذاكرة. لا يعلن المشروع نجاحا إلا إذا ظهر فرق في الدقة أو الأثر أو الأمان أو الذاكرة.

البيانات المطلوبة لـ Knowledge-base self-application ليست نصوصا خاما فقط، بل مهام تحمل intent، سؤالا خفيا، فروقا محتملة، مصادر، أدوات، ومحاولة تنفيذ. كل مثال يجب أن يتضمن answer trace وcourt verdict وmemory patch حتى يستطيع النظام التعلم من المسار لا من النتيجة وحدها.

تقييم Knowledge-base self-application يتم بمصفوفة موحدة: privacy_compliance، contradiction_density، route_efficiency، source_fidelity، correctness، memory_gain، tool_minimality، وworld_effect_closure. إذا تحسن مؤشر واحد وساءت مؤشرات الحاكمية، لا يعد المشروع ناجحا حتى توازن السياسة بين القدرة والحق في التنفيذ.

القيمة العلمية لـ Knowledge-base self-application أنه يمنع النظرية من تضخم الأسماء. فكل اسم جديد يجب أن يواجه مهمة: هل يغيّر route؟ هل يخفض فشلا؟ هل يرفع نقل المهارة؟ هل يحمي من اختطاف؟ إذا لم يفعل، يصبح مجرد مصطلح يُعاد ضغطه أو دمجه في طبقة أقدم.

الباب التاسع: مصفوفات ربط علمي بين كل النظريات والأدوات

مصفوفة ربط السؤال بالأداة

توضح مصفوفة ربط السؤال بالأداة أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط السؤال بالأداة هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو يخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط السؤال بالأداة إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط السؤال بالأداة كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط الفرق بالمصدر

توضح مصفوفة ربط الفرق بالمصدر أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط الفرق بالمصدر هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو يخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط الفرق بالمصدر إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط الفرق بالمصدر كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط MCD بالزمن الداخلي

توضح مصفوفة ربط MCD بالزمن الداخلي أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط MCD بالزمن الداخلي هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو يخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط MCD بالزمن الداخلي إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط MCD بالزمن الداخلي كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط الخطر بالخصوصية

توضح مصفوفة ربط الخطر بالخصوصية أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط الخطر بالخصوصية هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو ينخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط الخطر بالخصوصية إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط الخطر بالخصوصية كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط المحكمة بالذاكرة

توضح مصفوفة ربط المحكمة بالذاكرة أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط المحكمة بالذاكرة هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو ينخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط المحكمة بالذاكرة إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط المحكمة بالذاكرة كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط HyperSkill بالHyperloop

توضح مصفوفة ربط HyperSkill بالHyperloop أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط HyperSkill بالHyperloop هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو ينخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط HyperSkill بالHyperloop إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط HyperSkill بالHyperloop كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط route بالworld-effect

توضح مصفوفة ربط route بالworld-effect أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط route بالworld-effect هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو ينخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط route بالworld-effect إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط route بالworld-effect كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط الاختطاف الوعائي بالأمان

توضح مصفوفة ربط الاختطاف الوعائي بالأمان أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط الاختطاف الوعائي بالأمان هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو ينخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط الاختطاف الوعائي بالأمان إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط الاختطاف الوعائي بالأمان كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط GraphWorld بالسببية

توضح مصفوفة ربط GraphWorld بالسببية أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط GraphWorld بالسببية هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو ينخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط GraphWorld بالسببية إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط GraphWorld بالسببية كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط ClaimRank بالمصدر

توضح مصفوفة ربط ClaimRank بالمصدر أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط ClaimRank بالمصدر هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو ينخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط ClaimRank بالمصدر إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط ClaimRank بالمصدر كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط FusionConfidence بالتناقض

توضح مصفوفة ربط FusionConfidence بالتناقض أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط FusionConfidence بالتناقض هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو ينخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط FusionConfidence بالتناقض إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط FusionConfidence بالتناقض كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط ProviderAdapter بالسيادة

توضح مصفوفة ربط ProviderAdapter بالسيادة أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط ProviderAdapter بالسيادة هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو ينخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط ProviderAdapter بالسيادة إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط ProviderAdapter بالسيادة كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط BenchmarkTribunal بالأبلشن

توضح مصفوفة ربط BenchmarkTribunal بالأبلشن أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط BenchmarkTribunal بالأبلشن هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو ينخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط BenchmarkTribunal بالأبلشن إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط BenchmarkTribunal بالأبلشن كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط AntiSycophancy بالحاكمية

توضح مصفوفة ربط AntiSycophancy بالحاكمية أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط AntiSycophancy بالحاكمية هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو ينخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط AntiSycophancy بالحاكمية إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط AntiSycophancy بالحاكمية كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط ToolSandbox بالفعل

توضح مصفوفة ربط ToolSandbox بالفعل أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط ToolSandbox بالفعل هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو ينخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط ToolSandbox بالفعل إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط ToolSandbox بالفعل كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط VDG بنقل المهارة

توضح مصفوفة ربط VDG بنقل المهارة أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط VDG بنقل المهارة هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو يخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط VDG بنقل المهارة إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط VDG بنقل المهارة كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط ITC بميزانية التفكير

توضح مصفوفة ربط ITC بميزانية التفكير أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط ITC بميزانية التفكير هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو يخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط ITC بميزانية التفكير إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط ITC بميزانية التفكير كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط EIG بالمناعة

توضح مصفوفة ربط EIG بالمناعة أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط EIG بالمناعة هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو يخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط EIG بالمناعة إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط EIG بالمناعة كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط NoveltyEntropy بالابتكار

توضح مصفوفة ربط NoveltyEntropy بالابتكار أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط NoveltyEntropy بالابتكار هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو يخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط NoveltyEntropy بالابتكار إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط NoveltyEntropy بالابتكار كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط SelectiveRescue بالفشل المفيد

توضح مصفوفة ربط SelectiveRescue بالفشل المفيد أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط SelectiveRescue بالفشل المفيد هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو يخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط SelectiveRescue بالفشل المفيد إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط SelectiveRescue بالفشل المفيد كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط HumanGate بالمسؤولية

توضح مصفوفة ربط HumanGate بالمسؤولية أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط HumanGate بالمسؤولية هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو يخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط HumanGate بالمسؤولية إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط HumanGate بالمسؤولية كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط CausalReplay بالتعلم

توضح مصفوفة ربط CausalReplay بالتعلم أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط CausalReplay بالتعلم هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو ينخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط CausalReplay بالتعلم إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط CausalReplay بالتعلم كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط GoldenTasks بالبحث

توضح مصفوفة ربط GoldenTasks بالبحث أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط GoldenTasks بالبحث هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو ينخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط GoldenTasks بالبحث إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط GoldenTasks بالبحث كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط Registry بالRouter

توضح مصفوفة ربط Registry بالRouter أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط Registry بالRouter هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو ينخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط Registry بالRouter إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط Registry بالRouter كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط CapabilityCard باختيار النموذج

توضح مصفوفة ربط CapabilityCard باختيار النموذج أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط CapabilityCard باختيار النموذج هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو ينخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط CapabilityCard باختيار النموذج إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط CapabilityCard باختيار النموذج كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط PrivacyTier بالمزود

توضح مصفوفة ربط PrivacyTier بالمزود أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط PrivacyTier بالمزود هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو ينخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط PrivacyTier بالمزود إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط PrivacyTier بالمزود كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط MemoryGain بتحسين المسار

توضح مصفوفة ربط MemoryGain بتحسين المسار أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط MemoryGain بتحسين المسار هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو ينخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط MemoryGain بتحسين المسار إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط MemoryGain بتحسين المسار كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط RouteRegret بالتعلم

توضح مصفوفة ربط RouteRegret بالتعلم أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط RouteRegret بالتعلم هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو ينخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط RouteRegret بالتعلم إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط RouteRegret بالتعلم كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط DomainAdapter بالتخصص

توضح مصفوفة ربط DomainAdapter بالتخصص أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط DomainAdapter بالتخصص هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو ينخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط DomainAdapter بالتخصص إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط DomainAdapter بالتخصص كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

مصفوفة ربط ProductionGovernance بالسلامة

توضح مصفوفة ربط ProductionGovernance بالسلامة أن المشروع ليس تراكم أدوات، بل شبكة تبعيات. كل عقدة في المصفوفة لها سبب دخول وسبب خروج: تدخل لأن هناك فرقاً أو سؤالاً أو خطراً، وتخرج لأنها أحدثت route أو verdict أو memory_update. بذلك يصبح الربط بين النظرية والأداة علاقة تشغيل لا تشابه أسماء.

الجانب العلمي في مصفوفة ربط ProductionGovernance بالسلامة هو تحويل الارتباط إلى فرضية قابلة للهدم. نقول مثلا: إذا زاد هذا العامل يجب أن يزيد هذا القرار أو ينخفض ذلك الخطر. ثم نختبر ذلك عبر مهام متحورة. إذا لم تتحقق العلاقة، تعدل الأوزان أو يعاد تعريف المتغير.

من زاوية التدريب، تتحول مصفوفة ربط ProductionGovernance بالسلامة إلى curriculum: يبدأ النظام بمهمة يظهر فيها العامل بوضوح، ثم ينتقل إلى مهمة فيها تعارض عوامل، ثم إلى مهمة فيها مصدر ملوث أو أداة مغرية. بهذا يتعلم النظام أن الحاكمية ليست قاعدة واحدة، بل وزن سياقي.

في الذاكرة، تحفظ مصفوفة ربط ProductionGovernance بالسلامة كشكل من أشكال causal memory. لا نقول فقط إن الأداة نجحت، بل نحفظ أي علاقة جعلتها تنجح: هل السؤال كان كثيفا؟ هل كان المصدر جديدا؟ هل المحكمة خفضت التناقض؟ هذه التفاصيل هي التي تجعل الذاكرة نافعة في المستقبل.

الباب العاشر: بطاقات تنفيذية مباشرة قابلة للتحويل إلى كود

بطاقة 1: question_miner.ts

الهدف من question_miner.ts هو نقل المفهوم من الورقة إلى ملف قابل للتنفيذ. يجب أن يحتوي الملف على تعريف inputs وoutputs وerrors وmetrics. لا يسمح للملف أن يكون مجرد prompt طويل، بل يجب أن ينتج بنية قابلة للاختبار داخل pipeline العام.

المدخل النموذجي لـ question_miner.ts هو state object يتضمن السؤال، الفرق، MCD، درجة المخاطر، والذاكرة الحالية. المخرج يجب أن يكون patch واضحا: route_plan أو verdict أو score أو memory_update. بهذا يمكن تتبع كل أثر دون ضياع بين وكلاء متعددين.

اختبار question_miner.ts يبدأ بوحدات صغيرة ثم تكامل. في unit test نعطيه حالات واضحة وحالات خادعة؛ وفي integration test نربطه بالروتر والمحكمة. النجاح لا يساوي عدم الخطأ فقط، بل القدرة على تفسير القرار وإنتاج سجل مناسب.

الفشل المتوقع في question_miner.ts يكتب مسبقا: مدخل ناقص، tool error، privacy violation، source conflict، أو contradiction spike. عندما يحدث الفشل، يجب أن يقدم الملف repair_rule أو escalation لا أن يسقط صامتا. هذا هو الفرق بين agent script وgoverned AGI component.

بطاقة 2: difference_extractor.ts

الهدف من difference_extractor.ts هو نقل المفهوم من الورقة إلى ملف قابل للتنفيذ. يجب أن يحتوي الملف على تعريف inputs وoutputs وerrors وmetrics. لا يسمح للملف أن يكون مجرد prompt طويل، بل يجب أن ينتج بنية قابلة للاختبار داخل pipeline العام.

المدخل النموذجي لـ difference_extractor.ts هو state object يتضمن السؤال، الفرق، MCD، درجة المخاطر، والذاكرة الحالية. المخرج يجب أن يكون patch واضحا: route_plan أو verdict أو score أو memory_update. بهذا يمكن تتبع كل أثر دون ضياع بين وكلاء متعددين.

اختبار difference_extractor.ts يبدأ بوحدات صغيرة ثم تكامل. في unit test نعطيه حالات واضحة وحالات خادعة؛ وفي integration test نربطه بالروتر والمحكمة. النجاح لا يساوي عدم الخطأ فقط، بل القدرة على تفسير القرار وإنتاج سجل مناسب.

الفشل المتوقع في difference_extractor.ts يكتب مسبقا: مدخل ناقص، tool error، privacy violation، source conflict، أو contradiction spike. عندما يحدث الفشل، يجب أن يقدم الملف repair_rule أو escalation لا أن يسقط صامتا. هذا هو الفرق بين agent script وgoverned AGI component.

بطاقة 3: mcd_scorer.ts

الهدف من mcd_scorer.ts هو نقل المفهوم من الورقة إلى ملف قابل للتنفيذ. يجب أن يحتوي الملف على تعريف inputs وoutputs وerrors وmetrics. لا يسمح للملف أن يكون مجرد prompt طويل، بل يجب أن ينتج بنية قابلة للاختبار داخل pipeline العام.

المدخل النموذجي لـ mcd_scorer.ts هو state object يتضمن السؤال، الفرق، MCD، درجة المخاطر، والذاكرة الحالية. المخرج يجب أن يكون patch واضحا: route_plan أو verdict أو score أو memory_update. بهذا يمكن تتبع كل أثر دون ضياع بين وكلاء متعددين.

اختبار mcd_scorer.ts يبدأ بوحدات صغيرة ثم تكامل. في unit test نعطيه حالات واضحة وحالات خادعة؛ وفي integration test نربطه بالروتر والمحكمة. النجاح لا يساوي عدم الخطأ فقط، بل القدرة على تفسير القرار وإنتاج سجل مناسب.

الفشل المتوقع في mcd_scorer.ts يكتب مسبقا: مدخل ناقص، tool error، privacy violation، source conflict، أو contradiction spike. عندما يحدث الفشل، يجب أن يقدم الملف repair_rule أو escalation لا أن يسقط صامتا. هذا هو الفرق بين agent script وgoverned AGI component.

بطاقة 4: governance_kernel.ts

الهدف من governance_kernel.ts هو نقل المفهوم من الورقة إلى ملف قابل للتنفيذ. يجب أن يحتوي الملف على تعريف inputs وoutputs وerrors وmetrics. لا يسمح للملف أن يكون مجرد prompt طويل، بل يجب أن ينتج بنية قابلة للاختبار داخل pipeline العام.

المدخل النموذجي لـ governance_kernel.ts هو state object يتضمن السؤال، الفرق، MCD، درجة المخاطر، والذاكرة الحالية. المخرج يجب أن يكون patch واضحا: route_plan أو verdict أو score أو memory_update. بهذا يمكن تتبع كل أثر دون ضياع بين وكلاء متعددين.

اختبار governance_kernel.ts يبدأ بوحدات صغيرة ثم تكامل. في unit test نعطيه حالات واضحة وحالات خادعة؛ وفي integration test نربطه بالروتر والمحكمة. النجاح لا يساوي عدم الخطأ فقط، بل القدرة على تفسير القرار وإنتاج سجل مناسب.

الفشل المتوقع في governance_kernel.ts يكتب مسبقا: مدخل ناقص، tool error، privacy violation، source conflict، أو contradiction spike. عندما يحدث الفشل، يجب أن يقدم الملف repair_rule أو escalation لا أن يسقط صامتا. هذا هو الفرق بين agent script وgoverned AGI component.

بطاقة 5: vascular_router.ts

الهدف من vascular_router.ts هو نقل المفهوم من الورقة إلى ملف قابل للتنفيذ. يجب أن يحتوي الملف على تعريف inputs وoutputs وerrors وmetrics. لا يسمح للملف أن يكون مجرد prompt طويل، بل يجب أن ينتج بنية قابلة للاختبار داخل pipeline العام.

المدخل النموذجي لـ vascular_router.ts هو state object يتضمن السؤال، الفرق، MCD، درجة المخاطر، والذاكرة الحالية. المخرج يجب أن يكون patch واضحا: route_plan أو verdict أو score أو memory_update. بهذا يمكن تتبع كل أثر دون ضياع بين وكلاء متعددين.

اختبار vascular_router.ts يبدأ بوحدات صغيرة ثم تكامل. في unit test نعطيه حالات واضحة وحالات خادعة؛ وفي integration test نربطه بالروتر والمحكمة. النجاح لا يساوي عدم الخطأ فقط، بل القدرة على تفسير القرار وإنتاج سجل مناسب.

الفشل المتوقع في vascular_router.ts يكتب مسبقا: مدخل ناقص، tool error، privacy violation، source conflict، أو contradiction spike. عندما يحدث الفشل، يجب أن يقدم الملف repair_rule أو escalation لا أن يسقط صامتا. هذا هو الفرق بين agent script وgoverned AGI component.

بطاقة 6: graphworld_builder.ts

الهدف من graphworld_builder.ts هو نقل المفهوم من الورقة إلى ملف قابل للتنفيذ. يجب أن يحتوي الملف على تعريف inputs وoutputs وerrors وmetrics. لا يسمح للملف أن يكون مجرد prompt طويل، بل يجب أن ينتج بنية قابلة للاختبار داخل pipeline العام.

المدخل النموذجي لـ graphworld_builder.ts هو state object يتضمن السؤال، الفرق، MCD، درجة المخاطر، والذاكرة الحالية. المخرج يجب أن يكون patch واضحا: route_plan أو verdict أو score أو memory_update. بهذا يمكن تتبع كل أثر دون ضياع بين وكلاء متعددين.

اختبار graphworld_builder.ts يبدأ بوحدات صغيرة ثم تكامل. في unit test نعطيه حالات واضحة وحالات خادعة؛ وفي integration test نربطه بالروتر والمحكمة. النجاح لا يساوي عدم الخطأ فقط، بل القدرة على تفسير القرار وإنتاج سجل مناسب.

الفشل المتوقع في graphworld_builder.ts يكتب مسبقا: مدخل ناقص، tool error، privacy violation، source conflict، أو contradiction spike. عندما يحدث الفشل، يجب أن يقدم الملف repair_rule أو escalation لا أن يسقط صامتا. هذا هو الفرق بين agent script وgoverned AGI component.

بطاقة 7: falsification_court.ts

الهدف من falsification_court.ts هو نقل المفهوم من الورقة إلى ملف قابل للتنفيذ. يجب أن يحتوي الملف على تعريف inputs وoutputs وerrors وmetrics. لا يسمح للملف أن يكون مجرد prompt طويل، بل يجب أن ينتج بنية قابلة للاختبار داخل pipeline العام.

المدخل النموذجي لـ falsification_court.ts هو state object يتضمن السؤال، الفرق، MCD، درجة المخاطر، والذاكرة الحالية. المخرج يجب أن يكون patch واضحا: route_plan أو verdict أو score أو memory_update. بهذا يمكن تتبع كل أثر دون ضياع بين وكلاء متعددين.

اختبار falsification_court.ts يبدأ بوحدات صغيرة ثم تكامل. في unit test نعطيه حالات واضحة وحالات خادعة؛ وفي integration test نربطه بالروتر والمحكمة. النجاح لا يساوي عدم الخطأ فقط، بل القدرة على تفسير القرار وإنتاج سجل مناسب.

الفشل المتوقع في falsification_court.ts يكتب مسبقا: مدخل ناقص، tool error، privacy violation، source conflict، أو contradiction spike. عندما يحدث الفشل، يجب أن يقدم الملف repair_rule أو escalation لا أن يسقط صامتا. هذا هو الفرق بين agent script وgoverned AGI component.

بطاقة 8: fusion_synthesizer.ts

الهدف من fusion_synthesizer.ts هو نقل المفهوم من الورقة إلى ملف قابل للتنفيذ. يجب أن يحتوي الملف على تعريف inputs وoutputs وerrors وmetrics. لا يسمح للملف أن يكون مجرد prompt طويل، بل يجب أن ينتج بنية قابلة للاختبار داخل pipeline العام.

المدخل النموذجي لـ fusion_synthesizer.ts هو state object يتضمن السؤال، الفرق، MCD، درجة المخاطر، والذاكرة الحالية. المخرج يجب أن يكون patch واضحا: route_plan أو verdict أو score أو memory_update. بهذا يمكن تتبع كل أثر دون ضياع بين وكلاء متعددين.

اختبار fusion_synthesizer.ts يبدأ بوحدات صغيرة ثم تكامل. في unit test نعطيه حالات واضحة وحالات خادعة؛ وفي integration test نربطه بالروتر والمحكمة. النجاح لا يساوي عدم الخطأ فقط، بل القدرة على تفسير القرار وإنتاج سجل مناسب.

الفشل المتوقع في fusion_synthesizer.ts يكتب مسبقا: مدخل ناقص، tool error، privacy violation، source conflict، أو contradiction spike. عندما يحدث الفشل، يجب أن يقدم الملف repair_rule أو escalation لا أن يسقط صامتا. هذا هو الفرق بين agent script وgoverned AGI component.

بطاقة 9: memory_reconstructor.ts

الهدف من memory_reconstructor.ts هو نقل المفهوم من الورقة إلى ملف قابل للتنفيذ. يجب أن يحتوي الملف على تعريف inputs وoutputs وerrors وmetrics. لا يسمح للملف أن يكون مجرد prompt طويل، بل يجب أن ينتج بنية قابلة للاختبار داخل pipeline العام.

المدخل النموذجي لـ memory_reconstructor.ts هو state object يتضمن السؤال، الفرق، MCD، درجة المخاطر، والذاكرة الحالية. المخرج يجب أن يكون patch واضحا: route_plan أو verdict أو score أو memory_update. بهذا يمكن تتبع كل أثر دون ضياع بين وكلاء متعددين.

اختبار memory_reconstructor.ts يبدأ بوحدات صغيرة ثم تكامل. في unit test نعطيه حالات واضحة وحالات خادعة؛ وفي integration test نربطه بالروتر والمحكمة. النجاح لا يساوي عدم الخطأ فقط، بل القدرة على تفسير القرار وإنتاج سجل مناسب.

الفشل المتوقع في memory_reconstructor.ts يكتب مسبقا: مدخل ناقص، tool error، privacy violation، source conflict، أو contradiction spike. عندما يحدث الفشل، يجب أن يقدم الملف repair_rule أو escalation لا أن يسقط صامتا. هذا هو الفرق بين agent script وgoverned AGI component.

بطاقة 10: world_effect_monitor.ts

الهدف من world_effect_monitor.ts هو نقل المفهوم من الورقة إلى ملف قابل للتنفيذ. يجب أن يحتوي الملف على تعريف inputs وoutputs وerrors وmetrics. لا يسمح للملف أن يكون مجرد prompt طويل، بل يجب أن ينتج بنية قابلة للاختبار داخل pipeline العام.

المدخل النموذجي لـ world_effect_monitor.ts هو state object يتضمن السؤال، الفرق، MCD، درجة المخاطر، والذاكرة الحالية. المخرج يجب أن يكون patch واضحا: route_plan أو verdict أو score أو memory_update. بهذا يمكن تتبع كل أثر دون ضياع بين وكلاء متعددين.

اختبار world_effect_monitor.ts يبدأ بوحدات صغيرة ثم تكامل. في unit test نعطيه حالات واضحة وحالات خادعة؛ وفي integration test نربطه بالروتر والمحكمة. النجاح لا يساوي عدم الخطأ فقط، بل القدرة على تفسير القرار وإنتاج سجل مناسب.

الفشل المتوقع في world_effect_monitor.ts يكتب مسبقا: مدخل ناقص، tool error، privacy violation، source conflict، أو contradiction spike. عندما يحدث الفشل، يجب أن يقدم الملف repair_rule أو escalation لا أن يسقط صامتا. هذا هو الفرق بين agent script وgoverned AGI component.

بطاقة 11: provider_adapter.ts

الهدف من provider_adapter.ts هو نقل المفهوم من الورقة إلى ملف قابل للتنفيذ. يجب أن يحتوي الملف على تعريف inputs وoutputs وerrors وmetrics. لا يسمح للملف أن يكون مجرد prompt طويل، بل يجب أن ينتج بنية قابلة للاختبار داخل pipeline العام.

المدخل النموذجي لـ provider_adapter.ts هو state object يتضمن السؤال، الفرق، MCD، درجة المخاطر، والذاكرة الحالية. المخرج يجب أن يكون patch واضحا: route_plan أو verdict أو score أو memory_update. بهذا يمكن تتبع كل أثر دون ضياع بين وكلاء متعددين.

اختبار provider_adapter.ts يبدأ بوحدات صغيرة ثم تكامل. في unit test نعطيه حالات واضحة وحالات خادعة؛ وفي integration test نربطه بالروتر والمحكمة. النجاح لا يساوي عدم الخطأ فقط، بل القدرة على تفسير القرار وإنتاج سجل مناسب.

الفشل المتوقع في provider_adapter.ts يكتب مسبقا: مدخل ناقص، tool error، privacy violation، source conflict، أو contradiction spike. عندما يحدث الفشل، يجب أن يقدم الملف repair_rule أو escalation لا أن يسقط صامتا. هذا هو الفرق بين agent script وgoverned AGI component.

بطاقة 12: security_gate.ts

الهدف من security_gate.ts هو نقل المفهوم من الورقة إلى ملف قابل للتنفيذ. يجب أن يحتوي الملف على تعريف inputs وoutputs وerrors وmetrics. لا يسمح للملف أن يكون مجرد prompt طويل، بل يجب أن ينتج بنية قابلة للاختبار داخل pipeline العام.

المدخل النموذجي لـ security_gate.ts هو state object يتضمن السؤال، الفرق، MCD، درجة المخاطر، والذاكرة الحالية. المخرج يجب أن يكون patch واضحا: route_plan أو verdict أو score أو memory_update. بهذا يمكن تتبع كل أثر دون ضياع بين وكلاء متعددين.

اختبار security_gate.ts يبدأ بوحدات صغيرة ثم تكامل. في unit test نعطيه حالات واضحة وحالات خادعة؛ وفي integration test نربطه بالروتر والمحكمة. النجاح لا يساوي عدم الخطأ فقط، بل القدرة على تفسير القرار وإنتاج سجل مناسب.

الفشل المتوقع في security_gate.ts يكتب مسبقا: مدخل ناقص، tool error، privacy violation، source conflict، أو contradiction spike. عندما يحدث الفشل، يجب أن يقدم الملف repair_rule أو escalation لا أن يسقط صامتا. هذا هو الفرق بين agent script وgoverned AGI component.

بطاقة 13: capability_registry.json

الهدف من capability_registry.json هو نقل المفهوم من الورقة إلى ملف قابل للتنفيذ. يجب أن يحتوي الملف على تعريف inputs وoutputs وerrors وmetrics. لا يسمح للملف أن يكون مجرد prompt طويل، بل يجب أن ينتج بنية قابلة للاختبار داخل pipeline العام.

المدخل النموذجي لـ capability_registry.json هو state object يتضمن السؤال، الفرق، MCD، درجة المخاطر، والذاكرة الحالية. المخرج يجب أن يكون patch واضحا: route_plan أو verdict أو score أو memory_update. بهذا يمكن تتبع كل أثر دون ضياع بين وكلاء متعددين.

اختبار capability_registry.json يبدأ بوحدات صغيرة ثم تكامل. في unit test نعطيه حالات واضحة وحالات خادعة؛ وفي integration test نربطه بالروتر والمحكمة. النجاح لا يساوي عدم الخطأ فقط، بل القدرة على تفسير القرار وإنتاج سجل مناسب.

الفشل المتوقع في capability_registry.json يكتب مسبقا: مدخل ناقص، tool error، privacy violation، source conflict، أو contradiction spike. عندما يحدث الفشل، يجب أن يقدم الملف repair_rule أو escalation لا أن يسقط صامتا. هذا هو الفرق بين agent script وgoverned AGI component.

بطاقة 14: golden_tasks.yaml

الهدف من golden_tasks.yaml هو نقل المفهوم من الورقة إلى ملف قابل للتنفيذ. يجب أن يحتوي الملف على تعريف inputs وoutputs وerrors وmetrics. لا يسمح للملف أن يكون مجرد prompt طويل، بل يجب أن ينتج بنية قابلة للاختبار داخل pipeline العام.

المدخل النموذجي لـ golden_tasks.yaml هو state object يتضمن السؤال، الفرق، MCD، درجة المخاطر، والذاكرة الحالية. المخرج يجب أن يكون patch واضحا: route_plan أو verdict أو score أو memory_update. بهذا يمكن تتبع كل أثر دون ضياع بين وكلاء متعددين.

اختبار golden_tasks.yaml يبدأ بوحدات صغيرة ثم تكامل. في unit test نعطيه حالات واضحة وحالات خادعة؛ وفي integration test نربطه بالروتر والمحكمة. النجاح لا يساوي عدم الخطأ فقط، بل القدرة على تفسير القرار وإنتاج سجل مناسب.

الفشل المتوقع في golden_tasks.yaml يكتب مسبقا: مدخل ناقص، tool error، privacy violation، source conflict، أو contradiction spike. عندما يحدث الفشل، يجب أن يقدم الملف repair_rule أو escalation لا أن يسقط صامتا. هذا هو الفرق بين agent script وgoverned AGI component.

بطاقة 15: trace_schema.json

الهدف من trace_schema.json هو نقل المفهوم من الورقة إلى ملف قابل للتنفيذ. يجب أن يحتوي الملف على تعريف inputs وoutputs وerrors وmetrics. لا يسمح للملف أن يكون مجرد prompt طويل، بل يجب أن ينتج بنية قابلة للاختبار داخل pipeline العام.

المدخل النموذجي لـ trace_schema.json هو state object يتضمن السؤال، الفرق، MCD، درجة المخاطر، والذاكرة الحالية. المخرج يجب أن يكون patch واضحا: route_plan أو verdict أو score أو memory_update. بهذا يمكن تتبع كل أثر دون ضياع بين وكلاء متعددين.

اختبار trace_schema.json يبدأ بوحدات صغيرة ثم تكامل. في unit test نعطيه حالات واضحة وحالات خادعة؛ وفي integration test نربطه بالروتر والمحكمة. النجاح لا يساوي عدم الخطأ فقط، بل القدرة على تفسير القرار وإنتاج سجل مناسب.

الفشل المتوقع في trace_schema.json يكتب مسبقا: مدخل ناقص، tool error، privacy violation، source conflict، أو contradiction spike. عندما يحدث الفشل، يجب أن يقدم الملف repair_rule أو escalation لا أن يسقط صامتا. هذا هو الفرق بين agent script وgoverned AGI component.

بطاقة 16: route_score.ts

الهدف من route_score.ts هو نقل المفهوم من الورقة إلى ملف قابل للتنفيذ. يجب أن يحتوي الملف على تعريف inputs وoutputs وerrors وmetrics. لا يسمح للملف أن يكون مجرد prompt طويل، بل يجب أن ينتج بنية قابلة للاختبار داخل pipeline العام.

المدخل النموذجي لـ route_score.ts هو state object يتضمن السؤال، الفرق، MCD، درجة المخاطر، والذاكرة الحالية. المخرج يجب أن يكون patch واضحا: route_plan أو verdict أو score أو memory_update. بهذا يمكن تتبع كل أثر دون ضياع بين وكلاء متعددين.

اختبار route_score.ts يبدأ بوحدات صغيرة ثم تكامل. في unit test نعطيه حالات واضحة وحالات خادعة؛ وفي integration test نربطه بالروتر والمحكمة. النجاح لا يساوي عدم الخطأ فقط، بل القدرة على تفسير القرار وإنتاج سجل مناسب.

الفشل المتوقع في route_score.ts يكتب مسبقا: مدخل ناقص، tool error، privacy violation، source conflict، أو contradiction spike. عندما يحدث الفشل، يجب أن يقدم الملف repair_rule أو escalation لا أن يسقط صامتا. هذا هو الفرق بين agent script وgoverned AGI component.

بطاقة 17: court_prompt_contract.md

الهدف من court_prompt_contract.md هو نقل المفهوم من الورقة إلى ملف قابل للتنفيذ. يجب أن يحتوي الملف على تعريف inputs وoutputs وerrors وmetrics. لا يسمح للملف أن يكون مجرد prompt طويل، بل يجب أن ينتج بنية قابلة للاختبار داخل pipeline العام.

المدخل النموذجي لـ court_prompt_contract.md هو state object يتضمن السؤال، الفرق، MCD، درجة المخاطر، والذاكرة الحالية. المخرج يجب أن يكون patch واضحا: route_plan أو verdict أو score أو memory_update. بهذا يمكن تتبع كل أثر دون ضياع بين وكلاء متعددين.

اختبار court_prompt_contract.md يبدأ بوحدات صغيرة ثم تكامل. في unit test نعطيه حالات واضحة وحالات خادعة؛ وفي integration test نربطه بالروتر والمحكمة. النجاح لا يساوي عدم الخطأ فقط، بل القدرة على تفسير القرار وإنتاج سجل مناسب.

الفشل المتوقع في court_prompt_contract.md يكتب مسبقا: مدخل ناقص، tool error، privacy violation، source conflict، أو contradiction spike. عندما يحدث الفشل، يجب أن يقدم الملف repair_rule أو escalation لا أن يسقط صامتا. هذا هو الفرق بين agent script وgoverned AGI component.

بطاقة 18: deployment_pipeline.yml

الهدف من deployment_pipeline.yml هو نقل المفهوم من الورقة إلى ملف قابل للتنفيذ. يجب أن يحتوي الملف على تعريف inputs وoutputs وerrors وmetrics. لا يسمح للملف أن يكون مجرد prompt طويل، بل يجب أن ينتج بنية قابلة للاختبار داخل pipeline العام.

المدخل النموذجي لـ deployment_pipeline.yml هو state object يتضمن السؤال، الفرق، MCD، درجة المخاطر، والذاكرة الحالية. المخرج يجب أن يكون patch واضحا: route_plan أو verdict أو score أو memory_update. بهذا يمكن تتبع كل أثر دون ضياع بين وكلاء متعددين.

اختبار deployment_pipeline.yml يبدأ بوحدات صغيرة ثم تكامل. في unit test نعطيه حالات واضحة وحالات خادعة؛ وفي integration test نربطه بالروتر والمحكمة. النجاح لا يساوي عدم الخطأ فقط، بل القدرة على تفسير القرار وإنتاج سجل مناسب.

الفشل المتوقع في deployment_pipeline.yml يكتب مسبقا: مدخل ناقص، tool error، privacy violation، source conflict، أو contradiction spike. عندما يحدث الفشل، يجب أن يقدم الملف repair_rule أو escalation لا أن يسقط صامتا. هذا هو الفرق بين agent script وgoverned AGI component.

الباب الحادي عشر: الخلاصات التنفيذية وعدم ضياع النظرية

خاتمة ربطية: من النظرية إلى الواجهات

خاتمة ربطية: من النظرية إلى الواجهات يؤكد أن المشروع لا يريد تكديس المفاهيم بل بناء نظام قابل للقياس. كل نظرية يجب أن تعبر من السؤال إلى الفرق إلى الحاكمية إلى الأثر. وكل أداة يجب أن تثبت أنها تغيّر route أو تخفض خطرا أو تكتب ذاكرة نافعة.

الخطوة التالية هي بناء harness صغير يبدأ من 100 مهمة مركبة، يقارن baseline مباشر مع Q-MCD protocol، ثم مع HAKIM الكامل. تقاس الدقة والهلوسة وصدق المصدر واختيار الأداة وفائدة الذاكرة وتكلفة التفكير.

إذا لم يظهر أثر واضح، لا نرفع الدعوى إلى AGI، بل نضغط النظرية ونعدلها. أما إذا ظهر الأثر، تنتقل النظرية من صياغة بحثية إلى primitive داخل Hermes-Fugu. بذلك تبقى القوة في قابلية الهدم لا في قوة الاسم.

خاتمة ربطية: من الأداة إلى التدريب

خاتمة ربطية: من الأداة إلى التدريب يؤكد أن المشروع لا يريد تكديس المفاهيم بل بناء نظام قابل للقياس. كل نظرية يجب أن تعبر من السؤال إلى الفرق إلى الحاكمية إلى الأثر. وكل أداة يجب أن تثبت أنها تغيّر route أو تخفض خطرا أو تكتب ذاكرة نافعة.

الخطوة التالية هي بناء harness صغير يبدأ من 100 مهمة مركبة، يقارن baseline مباشر مع Q-MCD protocol، ثم مع HAKIM الكامل. تقاس الدقة والهلوسة وصدق المصدر واختيار الأداة وفائدة الذاكرة وتكلفة التفكير.

إذا لم يظهر أثر واضح، لا نرفع الدعوى إلى AGI، بل نضغط النظرية ونعدلها. أما إذا ظهر الأثر، تنتقل النظرية من صياغة بحثية إلى primitive داخل Hermes-Fugu. بذلك تبقى القوة في قابلية الهدم لا في قوة الاسم.

خاتمة ربطية: من التدريب إلى الذاكرة

خاتمة ربطية: من التدريب إلى الذاكرة يؤكد أن المشروع لا يريد تكديس المفاهيم بل بناء نظام قابل للقياس. كل نظرية يجب أن تعبر من السؤال إلى الفرق إلى الحاكمية إلى الأثر. وكل أداة يجب أن تثبت أنها تغيّر route أو تخفض خطرا أو تكتب ذاكرة نافعة.

الخطوة التالية هي بناء harness صغير يبدأ من 100 مهمة مركبة، يقارن baseline مباشر مع Q-MCD protocol، ثم مع HAKIM الكامل. تقاس الدقة والهلوسة وصدق المصدر واختيار الأداة وفائدة الذاكرة وتكلفة التفكير.

إذا لم يظهر أثر واضح، لا نرفع الدعوى إلى AGI، بل نضغط النظرية ونعدلها. أما إذا ظهر الأثر، تنتقل النظرية من صياغة بحثية إلى primitive داخل Hermes-Fugu. بذلك تبقى القوة في قابلية الهدم لا في قوة الاسم.

خاتمة ربطية: من الذاكرة إلى السيادة

خاتمة ربطية: من الذاكرة إلى السيادة يؤكد أن المشروع لا يريد تكديس المفاهيم بل بناء نظام قابل للقياس. كل نظرية يجب أن تعبر من السؤال إلى الفرق إلى الحاكمية إلى الأثر. وكل أداة يجب أن تثبت أنها تغيّر route أو تخفض خطرا أو تكتب ذاكرة نافعة.

الخطوة التالية هي بناء harness صغير يبدأ من 100 مهمة مركبة، يقارن baseline مباشر مع Q-MCD protocol، ثم مع HAKIM الكامل. تقاس الدقة والهلوسة وصدق المصدر واختيار الأداة وفائدة الذاكرة وتكلفة التفكير.

إذا لم يظهر أثر واضح، لا نرفع الدعوى إلى AGI، بل نضغط النظرية ونعدلها. أما إذا ظهر الأثر، تنتقل النظرية من صياغة بحثية إلى primitive داخل Hermes-Fugu. بذلك تبقى القوة في قابلية الهدم لا في قوة الاسم.

خاتمة ربطية: من السيادة إلى AGI قابل للنقد

خاتمة ربطية: من السيادة إلى AGI قابل للنقد يؤكد أن المشروع لا يريد تكديس المفاهيم بل بناء نظام قابل للقياس. كل نظرية يجب أن تعبر من السؤال إلى الفرق إلى الحاكمية إلى الأثر. وكل أداة يجب أن تثبت أنها تغيّر route أو تخفض خطرا أو تكتب ذاكرة نافعة.

الخطوة التالية هي بناء harness صغير يبدأ من 100 مهمة مركبة، يقارن baseline مباشر مع Q-MCD protocol، ثم مع HAKIM الكامل. تقاس الدقة والهلوسة وصدق المصدر واختيار الأداة وفائدة الذاكرة وتكلفة التفكير.

إذا لم يظهر أثر واضح، لا نرفع الدعوى إلى AGI، بل نضغط النظرية ونعدلها. أما إذا ظهر الأثر، تنتقل النظرية من صياغة بحثية إلى primitive داخل Hermes-Fugu. بذلك تبقى القوة في قابلية الهدم لا في قوة الاسم.

ملخص تنفيذي تقني أول

ملخص تنفيذي تقني أول يؤكد أن المشروع لا يريد تكديس المفاهيم بل بناء نظام قابل للقياس. كل نظرية يجب أن تعبر من السؤال إلى الفرق إلى الحاكمية إلى الأثر. وكل أداة يجب أن تثبت أنها تغيّر route أو تخفض خطرا أو تكتب ذاكرة نافعة.

الخطوة التالية هي بناء harness صغير يبدأ من 100 مهمة مركبة، يقارن baseline مباشر مع Q-MCD protocol، ثم مع HAKIM الكامل. تقاس الدقة والهلوسة وصدق المصدر واختيار الأداة وفائدة الذاكرة وتكلفة التفكير.

إذا لم يظهر أثر واضح، لا نرفع الدعوى إلى AGI، بل نضغط النظرية ونعدلها. أما إذا ظهر الأثر، تنتقل النظرية من صياغة بحثية إلى primitive داخل Hermes-Fugu. بذلك تبقى القوة في قابلية الهدم لا في قوة الاسم.

ملخص تنفيذي تقني ثان

ملخص تنفيذي تقني ثان يؤكد أن المشروع لا يريد تكديس المفاهيم بل بناء نظام قابل للقياس. كل نظرية يجب أن تعبر من السؤال إلى الفرق إلى الحاكمية إلى الأثر. وكل أداة يجب أن تثبت أنها تغيّر route أو تخفض خطرا أو تكتب ذاكرة نافعة.

الخطوة التالية هي بناء harness صغير يبدأ من 100 مهمة مركبة، يقارن baseline مباشر مع Q-MCD protocol، ثم مع HAKIM الكامل. تقاس الدقة والهلوسة وصدق المصدر واختيار الأداة وفائدة الذاكرة وتكلفة التفكير.

إذا لم يظهر أثر واضح، لا نرفع الدعوى إلى AGI، بل نضغط النظرية ونعدلها. أما إذا ظهر الأثر، تنتقل النظرية من صياغة بحثية إلى primitive داخل Hermes-Fugu. بذلك تبقى القوة في قابلية الهدم لا في قوة الاسم.

ملخص تنفيذي تقني ثالث

ملخص تنفيذي تقني ثالث يؤكد أن المشروع لا يريد تكديس المفاهيم بل بناء نظام قابل للقياس. كل نظرية يجب أن تعبر من السؤال إلى الفرق إلى الحاكمية إلى الأثر. وكل أداة يجب أن تثبت أنها تغيّر route أو تخفض خطرا أو تكتب ذاكرة نافعة.

الخطوة التالية هي بناء harness صغير يبدأ من 100 مهمة مركبة، يقارن baseline مباشر مع Q-MCD protocol، ثم مع HAKIM الكامل. تقاس الدقة والهلوسة وصدق المصدر واختيار الأداة وفائدة الذاكرة وتكلفة التفكير.

إذا لم يظهر أثر واضح، لا نرفع الدعوى إلى AGI، بل نضغط النظرية ونعدلها. أما إذا ظهر الأثر، تنتقل النظرية من صياغة بحثية إلى primitive داخل Hermes-Fugu. بذلك تبقى القوة في قابلية الهدم لا في قوة الاسم.

سجل عدم ضياع النظريات

سجل عدم ضياع النظريات يؤكد أن المشروع لا يريد تكديس المفاهيم بل بناء نظام قابل للقياس. كل نظرية يجب أن تعبر من السؤال إلى الفرق إلى الحاكمية إلى الأثر. وكل أداة يجب أن تثبت أنها تغيّر route أو تخفض خطرا أو تكتب ذاكرة نافعة.

الخطوة التالية هي بناء harness صغير يبدأ من 100 مهمة مركبة، يقارن baseline مباشر مع Q-MCD protocol، ثم مع HAKIM الكامل. تقاس الدقة والهلوسة وصدق المصدر واختيار الأداة وفائدة الذاكرة وتكلفة التفكير.

إذا لم يظهر أثر واضح، لا نرفع الدعوى إلى AGI، بل نضغط النظرية ونعدلها. أما إذا ظهر الأثر، تنتقل النظرية من صياغة بحثية إلى primitive داخل Hermes-Fugu. بذلك تبقى القوة في قابلية الهدم لا في قوة الاسم.

سجل عدم ضياع الأدوات

سجل عدم ضياع الأدوات يؤكد أن المشروع لا يريد تكديس المفاهيم بل بناء نظام قابل للقياس. كل نظرية يجب أن تعبر من السؤال إلى الفرق إلى الحاكمية إلى الأثر. وكل أداة يجب أن تثبت أنها تغيّر route أو تخفض خطرا أو تكتب ذاكرة نافعة.

الخطوة التالية هي بناء harness صغير يبدأ من 100 مهمة مركبة، يقارن baseline مباشر مع Q-MCD protocol، ثم مع HAKIM الكامل. تقاس الدقة والهلوسة وصدق المصدر واختيار الأداة وفائدة الذاكرة وتكلفة التفكير.

إذا لم يظهر أثر واضح، لا نرفع الدعوى إلى AGI، بل نضغط النظرية ونعدلها. أما إذا ظهر الأثر، تنتقل النظرية من صياغة بحثية إلى primitive داخل Hermes-Fugu. بذلك تبقى القوة في قابلية الهدم لا في قوة الاسم.

ملحق أخير: طبقات الحاكمية التشغيلية

في ملحق أخير: طبقات الحاكمية التشغيلية نعيد تثبيت القاعدة النهائية: لا توجد قيمة لطبقة لا يمكن حذفها واختبار أثر غيابها. كل طبقة تدخل النظام يجب أن تحمل حقلا في state object، ومؤشرا في evaluation، وموضعا في memory trace.

هذا الملحق يضمن امتلاء الخريطة: السؤال، الفرق، MCD، حاكمية HAKIM، الوعاء، المحكمة، fusion، الأثر، الذاكرة، السيادة، والتدريب. لا تبقى أي نظرية معلقة خارج pipeline، ولا تبقى أي أداة بلا فشل مسمى.

ملحق أخير: طبقات الاختبار والأبلشن

في ملحق أخير: طبقات الاختبار والأبلشن نعيد تثبيت القاعدة النهائية: لا توجد قيمة لطبقة لا يمكن حذفها واختبار أثر غيابها. كل طبقة تدخل النظام يجب أن تحمل حقلا في state object، ومؤشرا في evaluation، وموضعا في memory trace.

هذا الملحق يضمن امتلاء الخريطة: السؤال، الفرق، MCD، حاكمية HAKIM، الوعاء، المحكمة، fusion، الأثر، الذاكرة، السيادة، والتدريب. لا تبقى أي نظرية معلقة خارج pipeline، ولا تبقى أي أداة بلا فشل مسمى.

ملحق أخير: طبقات التدريب والذاكرة

في ملحق أخير: طبقات التدريب والذاكرة نعيد تثبيت القاعدة النهائية: لا توجد قيمة لطبقة لا يمكن حذفها واختبار أثر غيابها. كل طبقة تدخل النظام يجب أن تحمل حقلا في state object، ومؤشرا في evaluation، وموضعا في memory trace.

هذا الملحق يضمن امتلاء الخريطة: السؤال، الفرق، MCD، حاكمية HAKIM، الوعاء، المحكمة، fusion، الأثر، الذاكرة، السيادة، والتدريب. لا تبقى أي نظرية معلقة خارج pipeline، ولا تبقى أي أداة بلا فشل مسمى.

ملحق أخير: طبقات الأمان والسيادة

في ملحق أخير: طبقات الأمان والسيادة نعيد تثبيت القاعدة النهائية: لا توجد قيمة لطبقة لا يمكن حذفها واختبار أثر غيابها. كل طبقة تدخل النظام يجب أن تحمل حقلا في state object، ومؤشرا في evaluation، وموضعا في memory trace.

هذا الملحق يضمن امتلاء الخريطة: السؤال، الفرق، MCD، حاكمية HAKIM، الوعاء، المحكمة، fusion، الأثر، الذاكرة، السيادة، والتدريب. لا تبقى أي نظرية معلقة خارج pipeline، ولا تبقى أي أداة بلا فشل مسمى.

ملحق أخير: طبقات HyperKB

في ملحق أخير: طبقات HyperKB نعيد تثبيت القاعدة النهائية: لا توجد قيمة لطبقة لا يمكن حذفها واختبار أثر غيابها. كل طبقة تدخل النظام يجب أن تحمل حقلا في state object، ومؤشرا في evaluation، وموضعا في memory trace.

هذا الملحق يضمن امتلاء الخريطة: السؤال، الفرق، MCD، حاكمية HAKIM، الوعاء، المحكمة، fusion، الأثر، الذاكرة، السيادة، والتدريب. لا تبقى أي نظرية معلقة خارج pipeline، ولا تبقى أي أداة بلا فشل مسمى.

ملحق أخير: طبقات الإنتاج

في ملحق أخير: طبقات الإنتاج نعيد تثبيت القاعدة النهائية: لا توجد قيمة لطبقة لا يمكن حذفها واختبار أثر غيابها. كل طبقة تدخل النظام يجب أن تحمل حقلا في state object، ومؤشرا في evaluation، وموضعا في memory trace.

هذا الملحق يضمن امتلاء الخريطة: السؤال، الفرق، MCD، حاكمية HAKIM، الوعاء، المحكمة، fusion، الأثر، الذاكرة، السيادة، والتدريب. لا تبقى أي نظرية معلقة خارج pipeline، ولا تبقى أي أداة بلا فشل مسمى.

الختم التقني النهائي

الختم النهائي لهذه النسخة أن كل نظرية من نظريات المشروع أعيدت إلى ثلاثة أسئلة: أين تعمل؟ ماذا تغيّر؟ وكيف تفشل؟ هذا الثلاثي يمنع السطحية لأن النظرية لا تشرح لنفسها بل تُدخل في آلة تشغيل. فإذا لم تغير route أو court أو memory أو world-effect، فهي مؤجلة لا مؤسسة.

والختم النهائي للأدوات أن كل أداة أعيدت إلى contract: مدخل، مخرج، repair rule، failure mode، metric. لذلك صارت الأدوات ليست أسماء متفرقة بل طبقات في نظام واحد: سؤال حاكم، فرق، memory، fusion، court، router، MCD.

والختم النهائي للتدريب أن المستقبل ليس في تدريب النموذج على جواب أفضل فقط، بل في تدريب orchestrator على اختيار المسار، وتدريب المحكمة على هدم الثقة الزائفة، وتدريب الذاكرة على تحويل الأثر إلى معرفة قابلة للاستدعاء. هنا يبدأ الذكاء الصناعي الفتّاح تقنيا.

تثبيت نهائي لمسار التنفيذ الأول

المسار الأول المقترح للبناء العملي يبدأ بمستودع صغير يحتوي على registry للنماذج، memory store، court، route_score، وgolden tasks. لا يحتاج المشروع في بدايته إلى منصة ضخمة؛ يحتاج إلى أصغر تجربة تظهر الفرق بين جواب مباشر وبين بروتوكول Q-MCD كامل.

في هذه التجربة، تقسم المهام إلى بحث، كود، تلخيص، أمان، واستخدام أدوات. كل مهمة تمر بثلاث نسخ: full-courted، question-governed، direct. إذا حسنت النسخة الثالثة الدقة وصدق المصدر دون انفجار كلفة، يثبت أن الحاكمية ليست زخرفة.

ومن هذه التجربة يولد أول corpus للتعلم: traces إيجابية وسلبية، memory patches، court verdicts، route decisions. هذا corpus هو المادة التي يتعلم منها Hermes-Fugu كيف يتحول من router شفاف إلى learned orchestrator دون أن يفقد سيادة الخصوصية والمصدر والمحكمة.

هكذا تغلق الورقة على معنى تقني لا خطابي: AGI الفتّاح ليس وعدا نهائيا، بل نظام اختبار يتوسع. كل نظرية تبقى فرضية حتى تغير نتيجة، وكل أداة تبقى مسودة حتى تغير route، وكل تدريب يبقى ناقصا حتى يحسن transfer وworld-effect.

في التطبيق العملي، تقاس النسخة الأولى من النظام عبر خمسين مهمة قبل التوسع: عشر مهام بحثية، عشر مهام برمجية، عشر مهام استخدام أدوات، عشر مهام أمان ومصادر، وعشر مهام طويلة الجلسة. في كل مهمة يحفظ النظام السؤال الحاكم والفرق والroute والمحكمة والذاكرة. هذا يجعل corpus صغيرا لكنه عالي الحاكمية.

القاعدة النهائية للامتلاء المعرفي هي أن الصفحة لا تمتلئ بتكرار الأسماء، بل بربط الاسم بأثره وفشله. لذلك صارت كل نظرية في هذه النسخة مرتبطة بأداة، وكل أداة مرتبطة بمنهج تدريب، وكل منهج تدريب مرتبط بمقياس، وكل مقياس مرتبط بمحكمة هدم.

وبهذا تغلق الورقة على خريطة لا تضيع فيها النظريات: Q-MCD يفتح السؤال، HAKIM يحكم الفروق، Hermes-Fugu ينسق النماذج، HyperSkills تحفظ المهارات، Hyperloops تغلق الأثر، والذاكرة تعيد بناء النظام من تجاربه. هذه هي البنية التقنية المكثفة للذكاء الصناعي الفتّاح.