الورقة التقنية التركيبية العميقة لمشروع MICAI والذكاء الفتّاح

إعادة بناء نظاميّة لكل نظريات MCD/QMCD وMCAI/MICAGI/neoAGI وHAKIM-MCD والسؤالية وHyperKnowledge OS بوصفها معمارية ذكاء وكالي قابلة للتنفيذ والهدم والقياس

Zouhair Moussaif / زهير مساعف — إنجاز وتركيب زهير مساعف

تنبيه منهجي: المادة الأصلية الكاملة لكل ملفات المشروع السابقة ليست كلها ظاهرة داخل هذه الرسالة؛ لذلك تبني هذه النسخة على سياق المشروع المتاح هنا وعلى الأسماء والنظريات التي ثبت حضورها في الذاكرة الحالية، مع تمييز واضح بين النظرية المقترحة، والأداة التقديرية، والفرضية التي تحتاج تجربة. لا توجد مراجع مخترعة في هذه النسخة؛ وما يحتاج توثيقا خارجيا يحال إلى البرنامج البحثي اللاحق.

1. المدخل المنهجي: من الورقة السطحية إلى الورقة التركيبية

المشكلة في النسخة الخفيفة ليست نقص الأسماء، بل نقص تحويل الأسماء إلى علاقات تشغيلية مترابطة. لذلك تعيد هذه الورقة بناء المشروع من زاوية تقنية صرفة، حيث لا تقبل أي نظرية إلا إذا ظهرت داخلها كطبقة نظام، وكائن بيانات، وخوارزمية قرار، ومؤشر تقييم، ومحكمة هدم، ومسار أثر قابل للرصد.

المنهج المستخدم هو الضغط التركيبي المعرفي: تفكيك كل نواة نظرية إلى فرق حاكم، ثم ضغط الفروق في معمارية واحدة، ثم إعادة توليد الأدوات والمقاييس والفرضيات منها. لا يتم التعامل مع MCD أو QMCD أو MCAI كشعارات، بل كقواعد routing وتقييم وتنفيذ داخل نظام يعمل على نصوص وملفات وأدوات ونماذج وذاكرة.

الوحدات الدنيا التي تنظم الورقة هي Knowledge Atom وDifference Object وQuestion Object وAction Object وWorld Effect وMemory Patch وObject. كل كائن منها يحمل حقولا محددة تسمح بتحويل النظرية إلى runtime، لا إلى لغة تفسيرية عامة.

النسخة النهائية تبقى فرضية معمارية قابلة للهدم. لذلك يضاف إلى كل محور سؤال: ما الذي قد يكسره؟ هل هو جديد فعلا أم إعادة تسمية؟ هل توجد آلية أم استعارة؟ وهل يمكن قياسه أم أنه يظل خطابا؟

بهذا يتحول المشروع إلى MICAI Operating System: نواة لا تكتب الجواب فقط، بل تتحكم في شروط إنتاج الجواب، وتراقب أثره، وتعيد بناء ذاكرتها بعد الفشل.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتّاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور المدخل المنهجي: من الورقة السطحية إلى الورقة التركيبية بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور المدخل المنهجي: من الورقة السطحية إلى الورقة التركيبية إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور المدخل المنهجي: من الورقة السطحية إلى الورقة التركيبية داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور المدخل المنهجي: من الورقة السطحية إلى الورقة التركيبية إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور المدخل المنهجي: من الورقة السطحية إلى الورقة التركيبية أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور المدخل المنهجي: من الورقة السطحية إلى الورقة التركيبية لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدّعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور المدخل المنهجي: من الورقة السطحية إلى الورقة التركيبية وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور المدخل المنهجي: من الورقة السطحية إلى الورقة التركيبية قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور المدخل المنهجي: من الورقة السطحية إلى الورقة التركيبية سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور المدخل المنهجي: من الورقة السطحية إلى الورقة التركيبية في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

2. سلم مراتب المعرفة في النسخة التقنية

هذه الورقة لا تدعي أن MICAI نظام AGI مكتمل أو حقيقة مثبتة تجريبيا، بل تقدمه كإطار تقني ناشئ منظم حول فرضيات قابلة للتطوير. المراتب المعرفية هنا ضرورية حتى لا تختلط اللغة الطموحة بالدليل، ولا يلتبس النموذج المقترح بالواقع المنفذ.

ما هو صلب في الورقة هو البنية المنطقية: وجود pipeline، ووحدات، وكائنات بيانات، ومحكمة هدم، ومؤشرات قياس. وما هو راجح هو أن هذه البنية تحسن قدرة النظام على ضبط الاسترجاع والتوليد والتقييم مقارنة بالاعتماد على prompt منفرد. وما هو فرضي هو ادعاء أن تجميع هذه الطبقات ينتج قفزة نوعية في الوكالة أو التفكير الفتّاح.

كل مفهوم يأخذ رتبة: MCD أداة تقديرية، QMCD محرك حاكمية سؤالانية، HAKIM-MCD معمارية وحداتية، HyperKB OS طبقة معرفة قابلة للتنفيذ، والذكاء الفتّاح قدرة نظامية مقترحة على فتح الأسئلة والنماذج والاختبارات. لا يوصف أي منها بأنه مثبت إلا بعد تشغيل وتقييم.

إذا لم تملك النظرية اختبارا أو أثر عالم أو سجل فشل فهي تخفض من نظرية إلى استعارة منتجة. وإذا لم يملك المقياس بيانات calibration فهو يظل مقياسا أوليا. هذا التخفيض ليس إضعافا، بل حماية للنموذج من التضخم.

بهذا السلم، يصبح المشروع قابلا لدخول البحث العلمي والبرمجة معا: ما يثبت يتحول إلى معيار، وما يرجح يتحول إلى تجربة، وما يخمن يتحول إلى سؤال بحثي، وما ينهدم يتحول إلى patch في الذاكرة.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتّاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور سلم مراتب المعرفة في النسخة التقنية بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغيّر، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور سلم مراتب المعرفة في النسخة التقنية إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور سلم مراتب المعرفة في النسخة التقنية داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور سلم مراتب المعرفة في النسخة التقنية إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور سلم مراتب المعرفة في النسخة التقنية أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور سلم مراتب المعرفة في النسخة التقنية لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور سلم مراتب المعرفة في النسخة التقنية وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور سلم مراتب المعرفة في النسخة التقنية قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور سلم مراتب المعرفة في النسخة التقنية سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور سلم مراتب المعرفة في النسخة التقنية في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

3. النسب الاصطلاحي الداخلي: MICAI وMCAI وMICAGI وneoAGI

أول خلل يجب منعه هو الخلط بين أسماء المشروع. MICAI هو المظلة المندمجة التي تجمع السؤال والفرق والوكالة والذاكرة والأثر. MCAI هو القلب الوكالي الميتا-تعقيدي الذي يحول الفرق إلى اختيار وفعل. MICAGI هو التمديد العام نحو أنظمة وكلاء متعددة. neoAGI هو نمط التشغيل والنشر الذي يحول النظرية إلى منظومة أدوات وذاكرة ومزودين ونماذج.

عند التصميم، لا بد أن يكون لكل اسم موضع في المعمارية. MICAI يساوي النظام، MCAI يساوي حلقة القرار، HAKIM-MCD يساوي نواة الوحدات التسع، MCD/QMCD يساوي طبقة القياس والسؤال، HyperKB OS يساوي طبقة المعرفة، وneoAGI يساوي orchestration وdeployment وruntime governance.

الكائن التقني الذي يمنع الالتباس يسمى Theory Binding Record. يربط كل اسم بمكانه، وبالوظيفة التي يؤديها، وبمدخلاته ومخرجاته، وبالاختبارات التي تثبت أنه يعمل. فإذا ذكرنا السؤالية مثلا لا نترك كفلسفة، بل نربط بQuestion Object وQuestion Graph وQuestion Compiler.

الاعتراض الأقوى على كثرة الأسماء أنها قد تكون زخرفة اصطلاحية. الجواب التقني أن الاسم لا يقبل في MICAI إلا إذا أنتج واجهة تشغيل، أو مقياسا، أو دورا في pipeline. الاسم الذي لا يفعل شيئا يحذف أو يدمج.

بهذا تصبح العائلة الاصطلاحية خريطة طبقات: من المدخل الخام إلى فرق، ومن الفرق إلى سؤال، ومن السؤال إلى قرار، ومن القرار إلى فعل، ومن الفعل إلى أثر، ومن الأثر إلى ذاكرة تعيد بناء النظام.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور النسب الاصطلاحي الداخلي: MICAI وMCAI وMICAGI وneoAGI بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور النسب الاصطلاحي الداخلي: MICAI وMCAI وMICAGI وneoAGI إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور النسب الاصطلاحي الداخلي: MICAI وMCAI وMICAGI وneoAGI داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور النسب الاصطلاحي الداخلي: MICAI وMCAI وMICAGI وneoAGI إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور النسب الاصطلاحي الداخلي: MICAI وMCAI وMICAGI وneoAGI أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور النسب الاصطلاحي الداخلي: MICAI وMCAI وMICAGI وneoAGI لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور النسب الاصطلاحي الداخلي: MICAI وMCAI وMICAGI وneoAGI وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور النسب الاصطلاحي الداخلي: MICAI وMCAI وMICAGI وneoAGI قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور النسب الاصطلاحي الداخلي: MICAI وMCAI وMICAGI وneoAGI سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور النسب الاصطلاحي الداخلي: MICAI وMCAI وMICAGI وneoAGI في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

4. الأطروحة التقنية الموحدة: الذكاء بوصفه حاكمية فرق وسؤال وأثر

الأطروحة الجامعة تقول إن الذكاء في MICAI لا يقاس بجمال الجواب ولا بطول السياق فقط، بل بقدرة النظام على اكتشاف الفرق الفاعل، وإدخاله في سؤال حاكم، وتخصيص الموارد له، وهدمه قبل تبنيه، ثم تحويله إلى أثر قابل للرصد وإعادة بناء الذاكرة بعد الأثر.

المسار الأدنى هو: Input ثم Difference Extraction ثم MCD Scoring ثم QMCD Question Governance ثم Compression ثم Court ثم Agency Selection ثم Tool Execution ثم World Effect ثم Self-Reconstruction. كل سهم في هذه السلسلة يجب أن يحمل contract تقنيا، وإلا صار الكلام صورة معمارية لا نظاما.

في هذا المسار يتحول النص إلى Knowledge Atom، والاختلاف إلى Difference Object، والحاكمية إلى Question Object، والخطة إلى Action Object، والنتيجة إلى World Effect Object، والتعلم إلى Memory Patch. هذه الكائنات هي القابلية الهندسية للمشروع.

ما يهدم الأطروحة هو أن يثبت أن هذه الطبقات لا تضيف شيئا على RAG عادي أو prompt chain. لذلك يحتاج MICAI إلى تجارب تقارن بين جواب بلا MCD/QMCD وجواب بنفس النموذج مع هذه الطبقات، مع قياس جودة الاسترجاع، انخفاض الهلوسة، زيادة قابلية الاختبار، وتحسن القرار.

إذا نجحت التجارب يصبح MICAI ليس نموذجا بديلا عن LLM، بل طبقة تشغيل فوق النماذج: نظام حاكمية واستدلال وذاكرة يركب فوق أي نموذج تفكير أو كود أو رؤية أو صوت.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور الأطروحة التقنية الموحدة: الذكاء بوصفه حاكمية فرق وسؤال وأثر بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور الأطروحة التقنية الموحدة: الذكاء بوصفه حاكمية فرق وسؤال وأثر إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور الأطروحة التقنية الموحدة: الذكاء بوصفه حاكمية فرق وسؤال وأثر داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور الأطروحة التقنية الموحدة: الذكاء بوصفه حاكمية فرق وسؤال وأثر إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور الأطروحة التقنية الموحدة: الذكاء بوصفه حاكمية فرق وسؤال وأثر أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور الأطروحة التقنية الموحدة: الذكاء بوصفه حاكمية فرق وسؤال وأثر لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور الأطروحة التقنية الموحدة: الذكاء بوصفه حاكمية فرق وسؤال وأثر وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور الأطروحة التقنية الموحدة: الذكاء بوصفه حاكمية فرق وسؤال وأثر قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور الأطروحة التقنية الموحدة: الذكاء بوصفه حاكمية فرق وسؤال وأثر سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور الأطروحة التقنية الموحدة: الذكاء بوصفه حاكمية فرق وسؤال وأثر في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

5. نموذج البيانات الموحد: الكائنات التي تحمل النظرية

كل نظرية تقنية تحتاج جسم بيانات. إذا بقيت MCD أو السؤالية أو محكمة الهدم في مستوى اللغة فلن يمكن تنفيذها أو اختبارها. لذلك يقوم MICAI على نموذج بيانات يحول الأفكار إلى سجلات قابلة للاسترجاع والتحديث والمقارنة.

Knowledge Atom يحمل النص والمصدر والرتبة والثقة والعلاقات. Difference Object يحمل baseline والفرق والنطاق والأثر ومكونات MCD. Question Object يحمل السؤال والغرض والقيود ومعايير الاكتفاء. Action Object يحمل الأداة والمخاطر والميزانية ومعيار النجاح. World Effect Object يحمل before/after وmetric_delta وفشل التنفيذ. Memory Patch يحمل ما تغير في النظام بعد التعلم.

لا توجد ذاكرة واحدة مسطحة، بل ثلاث ذاكرات متداخلة: ذاكرة دلالية تحفظ المفاهيم والعلاقات، ذاكرة إجرائية تحفظ الخطوات والأدوات، وذاكرة نقدية تحفظ الاعتراضات والفشل والتعديلات. هذه الذاكرة النقدية هي ما يمنع تكرار أخطاء النظام.

إذا لم يحمل atom مصدرا أو رتبة ثقة، فهو خطر هلوسة. وإذا لم يحمل action معيار نجاح، فهو فعل غير قابل للحكم. وإذا لم يحمل world effect قياسا، فهو ادعاء أثر. لذلك يصبح فقر الحقول في البيانات نوعا من الفشل المعرفي.

هذا النموذج يربط كل فروع المشروع: MCD يحتاج QMCD، Difference Object يحتاج HAKIM، Question Object يحتاج Action وEffect، وHyperKB يحتاج Atom وRelation وMemory Patch. بهذا يتحد الاصطلاح مع التخزين.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور نموذج البيانات الموحد: الكائنات التي تحمل النظرية بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور نموذج البيانات الموحد: الكائنات التي تحمل النظرية إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور نموذج البيانات الموحد: الكائنات التي تحمل النظرية داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور نموذج البيانات الموحد: الكائنات التي تحمل النظرية إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور نموذج البيانات الموحد: الكائنات التي تحمل النظرية أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة، يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور نموذج البيانات الموحد: الكائنات التي تحمل النظرية لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور نموذج البيانات الموحد: الكائنات التي تحمل النظرية وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور نموذج البيانات الموحد: الكائنات التي تحمل النظرية قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور نموذج البيانات الموحد: الكائنات التي تحمل النظرية سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور نموذج البيانات الموحد: الكائنات التي تحمل النظرية في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

6. نظرية الفرق الفاعل داخل MCD

الفرق الفاعل هو أصغر تغير معرفي يمكنه إعادة ترتيب قرار أو سؤال أو بنية تفسيرية. ليست كل جدة فرقا فاعلا، فقد تكون المعلومة غريبة لكنها لا تغير شيئا. لذلك يميز MCD بين novelty سطحية وdifference حاكمة قادرة على توليد أثر.

تقنيا يبدأ النظام ببناء baseline يمثل ما يعرفه أو ما يفترضه المستخدم أو المجال. ثم يقارن المدخل بهذا baseline عبر اختلافات دلالية ومنطقية وسببية وزمنية. الفرق لا يسجل فقط بوصفه مخالفة، بل يربط بما يغيره: فرضية، مسار بحث، أداة، مخاطرة، أو قرار.

Difference Object يجب أن يحمل delta_text وbaseline_link وaffected_nodes وcontradiction_type وnovelty_source وMCD_vector وrequired_evidence. هذه الحقول تجعل الفرق قابلا للترتيب والحاكمية، فلا يبتلع النظام كل جديد بنفس الشهية.

أخطر خطأ هو تحويل الغرابة إلى أهمية. لذلك تسأل محكمة الهدم: هل الفرق يفسر أكثر؟ هل يغير قرارا؟ هل يقلل غموضا؟ هل يفتح اختبارا؟ إذا كان يلمع فقط ولا يعمل، يخفض إلى مادة هامشية.

من هنا يبدأ MICAI كله: لا سؤال حاكم بلا فرق، ولا وكالة بلا فرق، ولا تخصيص موارد بلا فرق. الفرق هو الشرارة التي تحرك الراوتر والسؤال والمحكمة والذاكرة.

من الناحية التطبيقية، لا يعمل مقياس الفرق إلا إذا امتلك النظام baseline واضحا. هذا baseline قد يكون ذاكرة المشروع، أو corpus متخصصا، أو فرضيات المستخدم، أو حالة تشغيل سابقة. وعندما يكتشف النظام delta، فهو لا يرفعها لأنها مختلفة فقط، بل لأنها تغير graph أو decision أو risk. هنا يصبح الفرق فاعلا بمعنى هندسي: له أثر قابل للتتبع داخل trace لا مجرد قيمة بلاغية.

عند الاختبار، يمكن إعطاء النظام نصين طويلين أحدهما يحمل فكرة حاسمة في هامش صغير والآخر يحمل معلومات كثيرة عادية. إذا رفع MCD الهامش الحاسم، فهذا دليل أولي على أنه يقيس الأثر لا الحجم. وإذا فضل النص الأطول دائما، فهذا يعني أن N وI وC تحتاج إعادة معايرة.

من جهة الوضع الأنطولوجي، يعامل محور نظرية الفرق الفاعل داخل MCD بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور نظرية الفرق الفاعل داخل MCD إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور نظرية الفرق الفاعل داخل MCD داخل traçe قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور نظرية الفرق الفاعل داخل MCD إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور نظرية الفرق الفاعل داخل MCD أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور نظرية الفرق الفاعل داخل MCD لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور نظرية الفرق الفاعل داخل MCD وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور نظرية الفرق الفاعل داخل MCD قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور نظرية الفرق الفاعل داخل MCD سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور نظرية الفرق الفاعل داخل MCD في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

7. MCD كمتجه تقييم لا كرقم زائف

الصيغة MCD = λ1D + λ2I + λ3T + λ4A + λ5P + λ6C - λ7N لا تعني دقة رياضية جاهزة، بل تقدم متجها تفسيريا يشرح لماذا يستحق عنصر ما اهتماما. القوة ليست في الرقم، بل في تفكيك الحكم إلى أبعاد قابلة للنقاش.

D يقيس كثافة الفرق، I يقيس أثره الاستدلالي، T يقيس عمقه الزمني، A يقيس قابليته لتوليد وكالة، P يقيس قابليته العملية، C يقيس ضغطه التركيبي، وN يعاقب الضجيج والادعاء الزائد. في البحث العلمي ترفع أوزان I وC وT، وفي المنتج العملي ترفع A وP، وفي السلامة ترفع عقوبة N.

MCD_Vector يخزن القيم مع تفسير كل قيمة. لا يكفي أن نقول D=0.8؛ يجب تسجيل أن الفرق أعاد تصنيف claim معين، أو كشف علاقة وسيطة، أو فتح فرضية فارقة. هذه الشفافية تجعل المقياس قابلا للتفتيش.

الاعتراض الأول أن MCD قد يتحول إلى مظهر رياضي بلا بيانات. لذلك يجب استعماله في البداية كتقدير ordinal لا كقياس ratio. يتحسن تدريجيا عندما تتراكم نتائج بشرية وتجارب مقارنة وتغذية راجعة.

داخل MICAI، يعمل MCD كطبقة triage: ما يدخل الذاكرة العالية، ما يستحق محكمة، ما يحتاج بحثا، وما يمر بجواب سريع. هكذا يصبح المقياس حاكما لتكلفة التفكير.

من الناحية التطبيقية، لا يعمل مقياس الفرق إلا إذا امتلك النظام baseline واضحا. هذا baseline قد يكون ذاكرة المشروع، أو corpus متخصصا، أو فرضيات المستخدم، أو حالة تشغيل سابقة. وعندما يكتشف النظام delta، فهو لا يرفعها لأنها مختلفة فقط، بل لأنها تغير graph أو decision أو risk. هنا يصبح الفرق فاعلا بمعنى هندسي: له أثر قابل للتتبع داخل trace لا مجرد قيمة بلاغية.

عند الاختبار، يمكن إعطاء النظام نصين طويلين أحدهما يحمل فكرة حاسمة في هامش صغير والآخر يحمل معلومات كثيرة عادية. إذا رفع MCD الهامش الحاسم، فهذا دليل أولي على أنه يقيس الأثر لا الحجم. وإذا فضل النص الأطول دائما، فهذا يعني أن N وI وC تحتاج إعادة معايرة.

من جهة الوضع الأنطولوجي، يعامل محور MCD كمتجه تقييم لا كرقم زائف بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور MCD كمتجه تقييم لا كرقم زائف إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور MCD كمتجه تقييم لا كرقم زائف داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور MCD كمتجه تقييم لا كرقم زائف إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور MCD كمتجه تقييم لا كرقم زائف أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور MCD كمتجه تقييم لا كرقم زائف لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور MCD كمتجه تقييم لا كرقم زائف وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور MCD كمتجه تقييم لا كرقم زائف قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور MCD كمتجه تقييم لا كرقم زائف سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور MCD كمتجه تقييم لا كرقم زائف في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

8. خوارزمية MCD: من النص الخام إلى prioritzation

خوارزمية MCD لا تبدأ بالحكم على النص كله، بل تقسمه إلى claims ومفاهيم وعلاقات واعتراضات. ثم تبني لكل عنصر مقارنة مع الذاكرة الحالية لتعرف هل أضاف فرقا، وهل الفرق موصول بالغاية، وهل يستحق تكلفة أعلى.

الخطوات هي: parsing، atomization، baseline retrieval، delta detection، graph impact propagation، score، explanation generation، noise penalty، normalization، ثم routing. في كل خطوة توجد سجلات تفتيش تسمح بمعرفة أين حدث الخطأ عند فشل النتيجة.

المدخل هو مجموعة Atoms، والمخرج ranked_deltas مع حقول score وreason وrisk وnext_question. هذا يعني أن MCD لا ينتهي بترتيب فقط، بل يقترح السؤال التالي الذي يجب أن يحكم الفرق.

إذا كان النظام يرفع أي عنصر غريب فالمشكلة في novelty bias. وإذا كان لا يرفع إلا ما يشبه الذاكرة فالمشكلة في conservatism bias. لذلك يحتاج MCD إلى calibration بأمثلة موجبة وسالبة من المجال نفسه.

بعد الترتيب، لا تنتقل العناصر مباشرة إلى الجواب، بل تمر إلى QMCD حتى يحكم السؤال انتشارها. هنا يظهر الفرق بين ذكاء يبحث عن المثير وذكاء يبحث عن الحاسم.

من الناحية التطبيقية، لا يعمل مقياس الفرق إلا إذا امتلك النظام baseline واضحا. هذا baseline قد يكون ذاكرة المشروع، أو corpus متخصصا، أو فرضيات المستخدم، أو حالة تشغيل سابقة. وعندما يكتشف النظام delta، فهو لا يرفعها لأنها مختلفة فقط، بل لأنها تغير graph أو decision أو risk. هنا يصبح الفرق فاعلا بمعنى هندسي: له أثر قابل للتتبع داخل trace لا مجرد قيمة بلاغية.

عند الاختبار، يمكن إعطاء النظام نصين طويلين أحدهما يحمل فكرة حاسمة في هامش صغير والآخر يحمل معلومات كثيرة عادية. إذا رفع MCD الهامش الحاسم، فهذا دليل أولي على أنه يقيس الأثر لا الحجم. وإذا فضل النص الأطول دائما، فهذا يعني أن N وI وC تحتاج إعادة معايرة.

من جهة الوضع الأنطولوجي، يعامل محور خوارزمية MCD: من النص الخام إلى prioritzation بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور خوارزمية MCD: من النص الخام إلى prioritzation إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور خوارزمية MCD: من النص الخام إلى prioritzation داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور خوارزمية MCD: من النص الخام إلى prioritzation إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور خوارزمية MCD: من النص الخام إلى prioritzation أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة، يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور خوارزمية MCD: من النص الخام إلى prioritzation لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور خوارزمية MCD: من النص الخام إلى prioritzation وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور خوارزمية MCD: من النص الخام إلى prioritzation قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور خوارزمية MCD: من النص الخام إلى prioritzation سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور خوارزمية MCD: من النص الخام إلى prioritzation في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

9. QMCD والسؤالية التقنية: السؤال كحاكمية تشغيل

السؤالية في المشروع ليست مهارة طرح أسئلة جميلة، بل نظرية تشغيلية تجعل السؤال مركز توزيع الموارد والمعنى والاختبار. في QMCD، السؤال يحدد ما الذي يعد فرقا، وما الذي يعد جوابا كافيا، وما الذي يعد انحرافا، وما الذي يحتاج هدمه.

السؤال يتحول إلى Question Object يحمل intent وscope وconstraints وexpected_effect وanswer_criteria وadversarial_forms. ثم يبنى منه Question Graph تتفرع منه أسئلة تفكيك وأدلة واعتراض وأثر. لا يسمح للنظام أن يجيب قبل تحديد معيار الاكتفاء.

Question Object يرتبط بDifference Objects عبر relation اسمها governs أو opens أو falsifies. السؤال قد يحكم فرقا، أو يفتحه، أو يهدمه، أو يحوله إلى تجربة. بهذه العلاقات تصبح السؤالية جزءا من الرسم المعرفي لا نصا خارجيا.

أقوى اعتراض على السؤالية أن السؤال قد يصبح قيدا يضيق الأفق. لذلك يجب أن يمتلك QMCD عملية question expansion المضبوطة: يوسع السؤال عندما يكشف MCD فرقاً حاسماً، لكنه يعاقب drift عندما يتحول التوسع إلى تشتت.

بهذا تصبح السؤالية kernel داخل MICAI. لا RAG بلا سؤال حاكم، لا توليد بلا معيار جواب، لا محكمة بلا أسئلة هدم، ولا ذاكرة بلا حفظ للأسئلة التي نجحت أو فشلت.

من الناحية التطبيقية، لا يعمل مقياس الفرق إلا إذا امتلك النظام baseline واضحا. هذا baseline قد يكون ذاكرة المشروع، أو corpus متخصصا، أو فرضيات المستخدم، أو حالة تشغيل سابقة. وعندما يكتشف النظام delta، فهو لا يرفعها لأنها مختلفة فقط، بل لأنها تغير graph أو decision أو risk. هنا يصبح الفرق فاعلا بمعنى هندسي: له أثر قابل للتتبع داخل trace لا مجرد قيمة بلاغية.

عند الاختبار، يمكن إعطاء النظام نصين طويلين أحدهما يحمل فكرة حاسمة في هامش صغير والآخر يحمل معلومات كثيرة عادية. إذا رفع MCD الهامش الحاسم، فهذا دليل أولي على أنه يقيس الأثر لا الحجم. وإذا فضل النص الأطول دائما، فهذا يعني أن N وI وC تحتاج إعادة معايرة.

من جهة الوضع الأنطولوجي، يعامل محور QMCD والسؤالية التقنية: السؤال كحاكمية تشغيل بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور QMCD والسؤالية التقنية: السؤال كحاكمية تشغيل إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور QMCD والسؤالية التقنية: السؤال كحاكمية تشغيل داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور QMCD والسؤالية التقنية: السؤال كحاكمية تشغيل إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور QMCD والسؤالية التقنية: السؤال كحاكمية تشغيل أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور QMCD والسؤالية التقنية: السؤال كحاكمية تشغيل لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور QMCD والسؤالية التقنية: السؤال كحاكمية تشغيل وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور QMCD والسؤالية التقنية: السؤال كحاكمية تشغيل قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور QMCD والسؤالية التقنية: السؤال كحاكمية تشغيل سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور QMCD والسؤالية التقنية: السؤال كحاكمية تشغيل في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

10. Question Graph وQuestion Compiler

لا يكفي تمثيل السؤال كسطر لغوي. السؤال في MICAI يتحول إلى graph من العمليات، ثم يترجم عبر Question Compiler إلى خطة تنفيذ تشمل الاسترجاع، والتحليل، والهدم، والقياس، والتوليد.

العقد هي أسئلة أصلية وفرعية واعتراضية وتطبيقية. الحواف هي عمليات مثل Decompose وContrast وCounterfactual وAdversarial وBridge وOperationalize وMetricize وCompress. Question Compiler يقرأ هذا الرسم وينتج execution plan مرتبة بالموارد والمخاطر.

الخطة الناتجة تحمل retrieval_queries وneeded_tools وcourt_prompts وevidence_thresholds وoutput_schema. هذا يحول السؤال من prompt إلى برنامج مصغر. وبه يمكن إعادة تشغيل السؤال على مادة جديدة أو مقارنة خطتين لنفس السؤال.

إذا فشل الجواب، لا يكفي لوم النموذج؛ قد يكون السؤال غامضا أو الرسم ناقصا أو الcompiler اختار أدوات غير مناسبة. لذلك تسجل الذاكرة ليس فقط الجواب الخاطئ، بل بنية السؤال التي أنتجته.

Question Graph يربط QMCD بHAKIM-MCD: المستخرج يستقبل أسئلة الفرق، مخطط التعقيد يستقبل أسئلة المستويات، المحكمة تستقبل أسئلة الهدم، وحلقة الأثر تستقبل سؤال ما الذي تغير في العالم.

تقنيا، السؤال الجيد يضبط ثلاثة حدود: ما الذي يدخل، وما الذي يخرج، ومتى نعد الجواب كافيا. بدون هذه الحدود، يصبح الاسترجاع مفتوحا والتوليد متضخما والمحكمة بلا هدف. لذلك يجب أن يصدر Question Compiler ليس قائمة أسئلة فقط، بل شروط قبول ورفض ومؤشرات اكتفاء.

نتعلم ذاكرة الأسئلة من الفشل. إذا أنتج سؤال ما جوابا عاما، يسجل النظام أنه احتاج قيدا أوضح. وإذا أنتج سؤال اعتراضي كشف خطأ قاتلا، يحفظ كنمط هدم. ومع الزمن يصبح للنظام repertory من الأسئلة الفعالة، وهو ما يميز السؤالية التقنية عن prompt engineering العادي.

من جهة الوضع الأنطولوجي، يعامل محور Question Graph وQuestion Compiler بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور Question Graph وQuestion Compiler إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور Question Graph وQuestion Compiler داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور Question Graph وQuestion Compiler إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور Question Graph وQuestion Compiler أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور Question Graph وQuestion Compiler لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور Question Graph وQuestion Compiler وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور Question Graph وQuestion Compiler قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور Question Graph وQuestion Compiler سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور Question Graph وQuestion Compiler في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

11. الضغط التركيبي المعرفي كمحرك Compression/Reconstruction

الضغط التركيبي المعرفي ليس تلخيصا، بل قدرة على تحويل كثرة متفرقة إلى قاعدة مولدة تستطيع إعادة بناء التفاصيل وتوليد تفاصيل جديدة. الفرق بينه وبين الاختصار أن الاختصار يحذف، أما الضغط التركيبي فيحفظ منطق التوليد.

النظام يجمع Atoms متفرقة، يبحث عن invariants، يقترح core rule، ثم يختبر القاعدة عبر reconstruction: هل تستطيع تفسير الحالات الأصلية؟ هل تستطيع التعامل مع الاستثناءات؟ هل تولد سؤالا جديدا أو أداة؟ إذا فشلت يعاد تفكيكها.

Compression Object يحمل core_rule وcovered_atoms وexceptions وreconstruction_error وgenerative_power وlost_meaning. هذا يمنع الجمل العامة من ادعاء الضغط وهي في الحقيقة تذيب الفروق.

المحكمة تسأل: ما الذي ضاع في الضغط؟ ما الذي زيد بلا دليل؟ هل القاعدة تفسر أم تعمم؟ هل أنتجت أداة أم شعارا؟ إذا كان lost_meaning عاليا، يخفض الضغط أو يقسم إلى قاعدتين.

MICAI يحتاج هذه الآلية كي لا تتحول HyperKB إلى أرشيف ضخم. المعرفة لا تكبر فقط؛ بل تضغط في قواعد حاكمة، ثم تحفظ الاستثناءات والحدود لتظل قابلة للاستعمال.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور الضغط التركيبي المعرفي كمحرك Compression/Reconstruction بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور الضغط التركيبي المعرفي كمحرك Compression/Reconstruction إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور الضغط التركيبي المعرفي كمحرك Compression/Reconstruction داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور الضغط التركيبي المعرفي كمحرك Compression/Reconstruction إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور الضغط التركيبي المعرفي كمحرك Compression/Reconstruction أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور الضغط التركيبي المعرفي كمحرك Compression/Reconstruction لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور الضغط التركيبي المعرفي كمحرك Compression/Reconstruction وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور الضغط التركيبي المعرفي كمحرك Compression/Reconstruction قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور الضغط التركيبي المعرفي كمحرك Compression/Reconstruction سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور الضغط التركيبي المعرفي كمحرك Compression/Reconstruction في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

12. التراكب الوعائي V0-V6 كطبقات تمثيل تقنية

في النسخة التقنية، V0-V6 لا تقدم دعوى بيولوجية نهائية عن الوعي، بل مخطط طبقات لتمثيل البيانات والمعنى والوكالة والغاية والمراقبة. قيمتها أنها تمنع خلط النص الخام بالمعنى، والمعنى بالفعل، والفعل بالأثر.

V0 هو المادة أو البيانات الخام، V1 الإشارة والملمح، V2 التصنيف والإدراك، V3 المعنى والعلاقات، V4 الوكالة والخطة، V5 الغاية والمعيار، V6 المراقبة الفوقية وإعادة البناء. كل مستوى يضيف وظيفة ولا يلغي ما قبله.

كل Atom يمكن أن يحمل multi-level tags. قد يكون النص في V0، والclaim في V3، والخطوة المقترحة في V4، والغاية في V5، والتقييم في V6. هذا التراكب يسمح للعنصر الواحد أن يعمل بأدوار متعددة دون فوضى.

الفشل يقع عندما يعامل النظام V0 كحقيقة أو V3 كفعل أو V5 كمعلومة وصفية. لذلك تحتاج كل عملية إلى type-checking معرفي يضمن أن ما يدخل planner هو فعل، وما يدخل evaluator هو معيار، وما يدخل memory هو أثر.

V0-V6 تمنح MICAI عمودا طبقيا فوق graph/vector. إنها تحدد أين تعيش المعلومة داخل الوعاء الحاسوبي، ومن أي مستوى يجب أن تستدعى عند السؤال أو الهدم أو التنفيذ.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور التراكب الوعائي V0-V6 كطبقات تمثيل تقنية بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور التراكب الوعائي V0-V6 كطبقات تمثيل تقنية إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور التراكب الوعائي V0-V6 كطبقات تمثيل تقنية داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور التراكب الوعائي V0-V6 كطبقات تمثيل تقنية إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور التراكب الوعائي V0-V6 كطبقات تمثيل تقنية أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور التراكب الوعائي V0-V6 كطبقات تمثيل تقنية لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور التراكب الوعائي V0-V6 كطبقات تمثيل تقنية وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور التراكب الوعائي V0-V6 كطبقات تمثيل تقنية قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور التراكب الوعائي V0-V6 كطبقات تمثيل تقنية سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور التراكب الوعائي V0-V6 كطبقات تمثيل تقنية في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

13. Vascular Compute Router كاقتصاد انتباه وحوسبة

النظام الذكي لا يوزع التفكير بالتساوي. Vascular Compute Router هو وحدة الاقتصاد الداخلي التي تقرر أين تصرف tokens، وأي نموذج يستدعى، وأي أداة تعمل، ومتى نحتاج محكمة هدم، ومتى يكفي جواب سريع.

r_i = f(MCD_i, QMCD_i, risk_i, uncertainty_i, value_i, latency_budget, cost_budget). هذه الصيغة تمنع تشغيل أقوى نموذج لكل شيء، وتمنع أيضا تمرير المسائل العالية الخطر عبر مسار سريع. الموارد تشمل الزمن والنماذج والأدوات والبحث والذاكرة وعمق التحقق.

Routing Decision يحمل task_type وselected_model وselected_tools وbudget وfallback وverification_depth. كل قرار routing يجب أن يبرر: لماذا اختير هذا المسار؟ ما البديل؟ ما خطر المسار الأسرع؟

إذا زادت الكلفة بلا أثر فالمشكلة over-routing. وإذا كثرت الأخطاء في مهام خطرة فالمشكلة under-routing. لذلك يقاس الراوتر بcost_per_valid_effect وerror_preventability وlatency_quality_curve.

الراوتر هو الشريان الذي يربط MCD بالسلوك العملي. الفرق العالي والسؤال الحاكم والمخاطر الكبيرة تفتح موارد أعلى، بينما الضجيج والتكرار ينقلان المهمة إلى مسار خفيف أو أرشفة.

لا تعني تعددية النماذج أن كل مهمة يجب أن تمر على كل نموذج. الحكمة التقنية هي اختيار الحد الأدنى الكافي من الموارد لتحقيق أثر مقبول. لذلك يحفظ الراوتر سجلا لتاريخ أداء كل نموذج في نوع المهمة، لا في متوسط عام مضلل.

الاستراتيجية المثلى غالبا هجينة: نموذج سريع لاستخراج أولي، نموذج عميق للمسائل العالية MCD، ونموذج نقدي أو محكمة لمراجعة الادعاءات. هذا يجعل التكلفة تابعة لقيمة السؤال لا عادة ثابتة.

من جهة الوضع الأنطولوجي، يعامل محور Vascular Compute Router كاقتصاد انتباه وحوسبة بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور Vascular Compute Router كاقتصاد انتباه وحوسبة إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور Vascular Compute Router كاقتصاد انتباه وحوسبة داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور Vascular Compute Router كاقتصاد انتباه وحوسبة إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور Vascular Compute Router كاقتصاد انتباه وحوسبة أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة، يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور Vascular Compute Router كاقتصاد انتباه وحوسبة لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور Vascular Compute Router كاقتصاد انتباه وحوسبة وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور Vascular Compute Router كاقتصاد انتباه وحوسبة قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور Vascular Compute Router كاقتصاد انتباه وحوسبة سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور Vascular Compute Router كاقتصاد انتباه وحوسبة في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

14. HAKIM-MCD كنواة وحداتية للنظام

HAKIM-MCD هو تحويل مشروع MICAI إلى نواة modules. قوته أنه لا يترك الذكاء في مستوى نموذج واحد، بل يقسمه إلى وظائف قابلة للتشخيص: استخراج فرق، خريطة تعقيد، زمن داخلي، توجيه موارد، مساحة عمل، اختيار وكالة، محكمة اضطراب، أثر عالم، وذاكرة إعادة بناء.

كل وحدة تتلقى كائنات محددة وتعيد كائنات محددة. Difference Extractor يعيد Deltas. Meta-Complexity Mapper يعيد graph وInner Time Engine يعيد scores. Router يعيد trajectories. Global Workspace يعيد routing decisions. Agency Selector يعيد actions. Court يعيد objections. World Loop يعيد effects. Memory يعيد patches. synthesis state يعيد.

تحتاج النواة إلى Event Bus يربط الوحدات. كل حدث يحمل trace_id حتى يمكن تتبع الجواب من السؤال إلى الأثر. بدون trace يصبح النظام غير قابل للتدقيق، ويضيع سبب الخطأ بين النموذج والذاكرة والأداة.

الاعتراض أن تعدد الوحدات قد يزيد التعقيد بلا فائدة. لذلك يجب قياس كل وحدة منفصلة: هل حسنت استخراج الفرق؟ هل خفضت الهلوسة؟ هل حسنت اختيار الأداة؟ الوحدة التي لا تضيف أثرها تزال أو تدمج.

بهذا يصبح HAKIM-MCD ليس رسما للزينة، بل architecture قابلة للobservability والاختبار والتطوير التدريجي. إنه يتيح بناء MICAI كمنصة لا كبرومبت.

من الناحية التطبيقية، لا يعمل مقياس الفرق إلا إذا امتلك النظام baseline واضحا. هذا baseline قد يكون ذاكرة المشروع، أو corpus متخصصا، أو فرضيات المستخدم، أو حالة تشغيل سابقة. وعندما يكتشف النظام delta، فهو لا يرفعها لأنها مختلفة فقط، بل لأنها تغير graph أو decision أو risk. هنا يصبح الفرق فاعلا بمعنى هندسي: له أثر قابل للتتبع داخل trace لا مجرد قيمة بلاغية.

عند الاختبار، يمكن إعطاء النظام نصين طويلين أحدهما يحمل فكرة حاسمة في هامش صغير والآخر يحمل معلومات كثيرة عادية. إذا رفع MCD الهامش الحاسم، فهذا دليل أولي على أنه يقيس الأثر لا الحجم. وإذا فضل النص الأطول دائما، فهذا يعني أن N وI وC تحتاج إعادة معايرة.

من جهة الوضع الأنطولوجي، يعامل محور HAKIM-MCD كنواة وحداتية للنظام بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور HAKIM-MCD كنواة وحداتية للنظام إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور HAKIM-MCD كنواة وحداتية للنظام داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور HAKIM-MCD كنواة وحداتية للنظام إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور HAKIM-MCD كنواة وحداتية للنظام أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور HAKIM-MCD كنواة وحداتية للنظام لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور HAKIM-MCD كنواة وحداتية للنظام وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور HAKIM-MCD كنواة وحداتية للنظام قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور HAKIM-MCD كنواة وحداتية للنظام سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور HAKIM-MCD كنواة وحداتية للنظام في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

15. Difference Extractor: استخراج الفرق قبل استخراج الجواب

الوحدة الأولى في HAKIM-MCD تبدأ من رفض التلخيص السطحي. وظيفتها أن تسأل: ما الفرق الذي أدخله هذا النص أو السؤال أو الملف؟ ما الذي خالف المتوقع؟ ما الذي يكشف ثغرة؟ ما الذي يغير خريطة المجال؟

تعمل الوحدة عبر مقارنة المدخل بالذاكرة والسياق والمهمة. تستخرج فروقا مفهومية وسببية ومنهجية وزمنية وتشغيلية. ثم تصنفها إلى فرق صلب، راجح، خلاف، فرضي، تخميني، أو ضجيج. التصنيف المبكر يمنع أن تدخل كل المواد إلى النظام بنفس القوة.

تخرج الوحدة قائمة Difference Objects مع روابط evidence وaffected_nodes وambiguity_flags. كما تضيف missing_information عندما يتوقف فهم الفرق على مصدر غير متاح أو ملف لم يقرأ.

المحكمة هنا تسأل: هل الفرق مصنوع من سوء فهم؟ هل هو تكرار باسم جديد؟ هل هناك baseline خاطئ؟ إذا كان baseline ضعيفا فالفرق كله قد يكون وهما. لذلك تحتاج الوحدة إلى تحديث baseline باستمرار.

كل ما يلي في MICAI يعتمد على جودة هذه الوحدة. سؤال ضعيف قد يتحسن، لكن فرق مستخرج خطأ يجعل الراوتر والمحكمة والذاكرة تعمل على مادة مشوهة.

من الناحية التطبيقية، لا يعمل مقياس الفرق إلا إذا امتلك النظام baseline واضحا. هذا baseline قد يكون ذاكرة المشروع، أو corpus متخصصا، أو فرضيات المستخدم، أو حالة تشغيل سابقة. وعندما يكتشف النظام delta، فهو لا يرفعها لأنها مختلفة فقط، بل لأنها تغير graph أو decision أو risk. هنا يصبح الفرق فاعلا بمعنى هندسي: له أثر قابل للتتبع داخل trace لا مجرد قيمة بلاغية.

عند الاختبار، يمكن إعطاء النظام نصين طويلين أحدهما يحمل فكرة حاسمة في هامش صغير والآخر يحمل معلومات كثيرة عادية. إذا رفع MCD الهامش الحاسم، فهذا دليل أولي على أنه يقيس الأثر لا الحجم. وإذا فضل النص الأطول دائما، فهذا يعني أن N وI وC تحتاج إعادة معايرة.

من جهة الوضع الأنطولوجي، يعامل محور Difference Extractor: استخراج الفرق قبل استخراج الجواب بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور Difference Extractor: استخراج الفرق قبل استخراج الجواب إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور Difference Extractor: استخراج الفرق قبل استخراج الجواب داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور Difference Extractor: استخراج الفرق قبل استخراج الجواب إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور Difference Extractor: استخراج الفرق قبل استخراج الجواب أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور Difference Extractor: استخراج الفرق قبل استخراج الجواب لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور Difference Extractor: استخراج الفرق قبل استخراج الجواب وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور Difference Extractor: استخراج الفرق قبل استخراج الجواب قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور Difference Extractor: استخراج الفرق قبل استخراج الجواب سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور Difference Extractor: استخراج الفرق قبل استخراج الجواب في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

16. Meta-Complexity Mapper: خريطة المستويات والعلاقات

بعد استخراج الفروق، لا بد من معرفة موقعها في شبكة التعقيد. Meta-Complexity Mapper يبني خريطة تربط المفاهيم بالآليات والمستويات والعتبات والاعتراضات والأدلة. إنه يمنع التعامل مع الفرق كجزيرة.

الخريطة تجمع graph معرفي، ومتجهات دلالية، وجداول قياس. في graph تظهر علاقات governs وcauses وmediates وcontradicts وtests وdepends_on. في vector تظهر القرابة الدلالية. في الجداول تحفظ الدرجات والتجارب.

Complexity Map يحمل nodes وedges وlevels وthresholds وuncertainty_regions وcourt_targets. كل عقدة يجب أن تملك type: مفهوم، فرضية، أداة، اعتراض، مصدر، أثر، أو قرار. كل حافة يجب أن تحمل علاقة لا مجرد ارتباط غامض.

الخطر الأكبر هو خريطة جميلة بلا دقة. لذلك تعاقب الحواف غير المسماة والعلاقات العامة من نوع relates_to. العلاقة الجيدة تقول كيف ولماذا وتحت أي شرط. بدون ذلك يصبح graph زينة.

هذه الخريطة هي ذاكرة العمل لـGlobal Workspace ومحرك الاسترجاع لـHyperKB. بدونها يتراجع MICAI إلى embedding search عادي، ويضيع البعد الحاكمي للمعرفة.

كل وحدة في HAKIM يجب أن تمتلك test fixture. يعطى لها مدخل معروف ومخرج متوقع، ثم تختبر بمعزل عن بقية النظام. إذا فشل الجواب النهائي، يمكن حينها تحديد هل الخلل في الاستخراج أم الخريطة أم الوكالة أم الأثر، وهذا شرط أساسي لبناء نظام قابل للصيانة.

الربط بين الوحدات يجب ألا يكون نصيا فقط. كل انتقال يحتاج schema. عندما ينتقل Difference Object إلى Mapper لا يفقد مصدره ورتبته. وعندما ينتقل Candidate Action إلى Court لا يفقد معيار نجاحه. حفظ الحقول عبر المراحل هو ما يحمي النظام من فقد السياق.

من جهة الوضع الأنطولوجي، يعامل محور Meta-Complexity Mapper: خريطة المستويات والعلاقات بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور Meta-Complexity Mapper: خريطة المستويات والعلاقات إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور Meta-Complexity Mapper: خريطة المستويات والعلاقات داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور Meta-Complexity Mapper: خريطة المستويات والعلاقات إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور Meta-Complexity Mapper: خريطة المستويات والعلاقات أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور Meta-Complexity Mapper: خريطة المستويات والعلاقات لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور Meta-Complexity Mapper: خريطة المستويات والعلاقات وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور Meta-Complexity Mapper: خريطة المستويات والعلاقات قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور Meta-Complexity Mapper: خريطة المستويات والعلاقات سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور Meta-Complexity Mapper: خريطة المستويات والعلاقات في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

17. Inner Time Engine: الزمن الداخلي للمعرفة

الزمن الداخلي في MICAI لا يعني التاريخ الخارجي فقط، بل مسار تحول الحالة المعرفية: كيف ظهر السؤال، كيف تغيرت الفرضية، ما الاعتراض الذي كسرها، وكيف أعيد بناؤها. هذا الزمن هو ذاكرة تطورية للنموذج.

الوحدة تمثل كل نظرية كمسار states. state_0 فرضية خام، state_1 بعد state_2، MCD بعد state_3، QMCD بعد المحكمة، state_4 بعد الأثر. تقاس التحولات عبر delta_of_delta: هل أصبح النموذج أوضح؟ أكثر قابلية للاختبار؟ أقل ادعاء؟

Trajectory Object يحمل version وchange_reason وbroken_assumptions وadded_constraints وnew_tests. بهذا لا تضيع أسباب التحسن. الذاكرة لا تحفظ النتيجة الأخيرة فقط، بل تحفظ لماذا وصل النظام إليها.

إذا لم يسجل النظام تاريخه الداخلي، يعيد الأخطاء نفسها. وإذا حفظ التاريخ بلا ضغط، يغرق في logs. لذلك يحتاج Inner Time إلى ضغط تطوري: حفظ التحولات الحاسمة لا كل تفصيل.

هذه الوحدة تغذي Self-Reconstruction Memory وتمنح البرنامج البحثي قدرة على التطور. إنها تجعل MICAI نظاما يتعلم من مسار تفكيره الظاهر لا من الجواب النهائي وحده.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور Inner Time Engine: الزمن الداخلي للمعرفة بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغيّر، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور Inner Time Engine: الزمن الداخلي للمعرفة إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور Inner Time Engine: الزمن الداخلي للمعرفة داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور Inner Time Engine: الزمن الداخلي للمعرفة إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور Inner Time Engine: الزمن الداخلي للمعرفة أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور Inner Time Engine: الزمن الداخلي للمعرفة لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور Inner Time Engine: الزمن الداخلي للمعرفة وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور Inner Time Engine: الزمن الداخلي للمعرفة قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور Inner Time Engine: الزمن الداخلي للمعرفة سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور Inner Time Engine: الزمن الداخلي للمعرفة في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

18. Global Workspace: مساحة العمل الجامعة بلا خلط

Global Workspace في MICAI هو المكان الذي تجتمع فيه الفروق والأسئلة والذاكرة والأدوات والسياق قبل القرار. ليس المقصود وعيا اصطناعيا، بل مساحة تشغيلية تسمح بتبادل المعلومات بين الوحدات دون أن تلغي حدودها.

تدخل إلى المساحة Deltas عالية MCD، وأسئلة QMCD، وخرائط التعقيد، وقيود المستخدم، ونتائج الأدوات. ثم تبنى synthesis state تحمل ما هو معلوم وما هو مجهول وما هو خطر وما يجب فعله. هذه الحالة تكون المصدر المباشر لاختيار الوكالة.

Workspace State يحمل active_question وcritical_deltas وevidence_state وavailable_tools وrisk_level وcandidate_actions. لا يسمح باستدعاء أداة أو إصدار جواب نهائي قبل اكتمال الحد الأدنى من هذه الحقول.

الخطر أن تتحول مساحة العمل إلى تجميع عشوائي. لذلك توجد قواعد gating: لا يدخل إلا ما له score أو علاقة بالسؤال أو أثر محتمل. كما يطرد drift الذي يزيد النص ولا يزيد القرار.

Global Workspace هو جسر بين المعرفة والفعل. هنا يلتقي HyperKB مع HAKIM، ويصبح السؤال قابلا للتحول إلى خطة، لا مجرد إجابة.

كل وحدة في HAKIM يجب أن تمتلك test fixture. يعطى لها مدخل معروف ومخرج متوقع، ثم تختبر بمعزل عن بقية النظام. إذا فشل الجواب النهائي، يمكن حينها تحديد هل الخلل في الاستخراج أم الخريطة أم الوكالة أم الأثر، وهذا شرط أساسي لبناء نظام قابل للصيانة.

الربط بين الوحدات يجب ألا يكون نصيا فقط. كل انتقال يحتاج schema. عندما ينتقل Difference Object إلى Mapper لا يفقد مصدره ورتبته. وعندما ينتقل Candidate Action إلى Court لا يفقد معيار نجاحه. حفظ الحقول عبر المراحل هو ما يحمي النظام من فقد السياق.

من جهة الوضع الأنطولوجي، يعامل محور Global Workspace: مساحة العمل الجامعة بلا خلط بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور Global Workspace: مساحة العمل الجامعة بلا خلط إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور Global Workspace: مساحة العمل الجامعة بلا خلط داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور Global Workspace: مساحة العمل الجامعة بلا خلط إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور Global Workspace: مساحة العمل الجامعة بلا خلط أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور Global Workspace: مساحة العمل الجامعة بلا خلط لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور Global Workspace: مساحة العمل الجامعة بلا خلط وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور Global Workspace: مساحة العمل الجامعة بلا خلط قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور Global Workspace: مساحة العمل الجامعة بلا خلط سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور Global Workspace: مساحة العمل الجامعة بلا خلط في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

19. Agency Selector: من الفهم إلى الفعل

الوكالة في MICAI ليست ادعاء أن النظام حر، بل قدرة عملية على اختيار فعل مناسب تحت قيود وسياق ومخاطر. Agency Selector يقرر هل نكتب، نبحث، نختبر، نسأل، نؤجل، نهدم، أو نستدعي أداة.

الوحدة تقارن candidate_actions وفق قيمة الأثر، التكلفة، الخطر، عدم اليقين، وإمكانية rollback. لا يكفي أن يكون الفعل مفيدا؛ يجب أن يكون مناسبا لرتبة المعرفة. الفرضية الضعيفة لا تعامل كقرار تنفيذي قوي، بل كتجربة أو سؤال.

Action Object يحمل action_type وtool_chain وprerequisites وsuccess_metric وfailure_mode وrollback_plan. كل فعل بلا معيار نجاح يعد ناقصا، وكل فعل خارجي بلا rollback يعد خطرا.

إذا أخطأ النظام في الفعل، تفحص المحكمة: هل كان الفهم صحيحا لكن الوكالة خاطئة؟ هل اختيرت أداة خاطئة؟ هل لم يقدّر الخطر؟ هل حول فرضية إلى أمر؟ هذا التشخيص يمنع تعميم الفشل على كل النواة.

Agency Selector يترجم الذكاء الفتّاح إلى أثر. فالنظام لا يفتح الأسئلة ليتأمل فقط، بل ليختار أفعالا أفضل وأكثر تواضعا وأكثر قابلية للتقييم.

كل وحدة في HAKIM يجب أن تمتلك test fixture. يعطى لها مدخل معروف ومخرج متوقع، ثم تختبر بمعزل عن بقية النظام. إذا فشل الجواب النهائي، يمكن حينها تحديد هل الخلل في الاستخراج أم الخريطة أم الوكالة أم الأثر، وهذا شرط أساسي لبناء نظام قابل للصيانة.

الربط بين الوحدات يجب ألا يكون نصيا فقط. كل انتقال يحتاج schema. عندما ينتقل Difference Object إلى Mapper لا يفقد مصدره ورتبته. وعندما ينتقل Candidate Action إلى Court لا يفقد معيار نجاحه. حفظ الحقول عبر المراحل هو ما يحمي النظام من فقد السياق.

من جهة الوضع الأنطولوجي، يعامل محور Agency Selector: من الفهم إلى الفعل بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغيّر، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور Agency Selector: من الفهم إلى الفعل إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور Agency Selector: من الفهم إلى الفعل داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور Agency Selector: من الفهم إلى الفعل إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور Agency Selector: من الفهم إلى الفعل أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور Agency Selector: من الفهم إلى الفعل لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور Agency Selector: من الفهم إلى الفعل وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور Agency Selector: من الفهم إلى الفعل قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور Agency Selector: من الفهم إلى الفعل سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور Agency Selector: من الفهم إلى الفعل في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

20. Perturbation Court ومحكمة الهدم

محكمة الهدم ليست فقرة نقدية تضاف في النهاية، بل بروتوكول تشغيلي يدخل قبل الفعل وبعده. وظيفتها بناء أقوى اعتراض ممكن على الفرق والسؤال والنموذج والخطة والأثر. بهذا يتحول النقد إلى وحدة هندسية.

تعمل المحكمة عبر أنماط: causal confusion، level-mixing، missing evidence، strongest competitor، reverse assumption، implementation failure، cost-risk، source gap، overclaim. كل اعتراض يحصل على رتبة: قاتل، معدل، محدود، أو ملاحظ.

Objection Object يحمل target وclaim وattack_type وstrength وrequired_fix وstatus. لا يكفي أن تذكر المحكمة الاعتراض؛ يجب أن تصدر patch أو تخفيض رتبة أو اختبارا فاصلا.

المحكمة تهدم نفسها أيضا: هل الاعتراض عادل أم خصم كرتوني؟ هل يطالب بدليل غير ممكن؟ هل يخلط بين نقد التنفيذ ونقد النظرية؟ المحكمة القوية لا تبحث عن إسقاط رخيص، بل عن نسخة أصلب.

كل فروع المشروع تمر هنا: MCD يهدم فرقه، QMCD يهدم أسئلته، HAKIM يهدم خطته، HyperKB يهدم مصادره، والذكاء الفتّاح يهدم ادعاءه بالفتح. بهذا لا يوجد خرق بلا ضبط.

محكمة الهدم تعمل كاختبار قبل النشر. لا تطلب من النموذج أن يكون متشائما، بل أن يكون دقيقا في حدود ما يدعيه. الفرق بين المحكمة والرفض أن المحكمة تقدم repair path. لذلك كل اعتراض لا ينتج تعديلا أو اختبارا أو تخفيضا للرتبة يعد اعتراضا ناقصا.

يمكن قياس المحكمة بنسبة الأخطاء التي كشفتها قبل المستخدم، وبنسبة الادعاءات التي خفضت رتبتها، وبجودة الإصلاح بعد النقد. إذا زادت المحكمة طول الجواب دون تحسين، فقد تحولت إلى طقس لا أداة، ويجب ضغطها أو تخصيصها للمهام العالية الأثر.

من جهة الوضع الأنطولوجي، يعامل محور Perturbation Court ومحكمة الهدم بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور Perturbation Court ومحكمة الهدم إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور Perturbation Court ومحكمة الهدم داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور Perturbation Court ومحكمة الهدم إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور Perturbation Court ومحكمة الهدم أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور Perturbation Court ومحكمة الهدم لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور Perturbation Court ومحكمة الهدم وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور Perturbation Court ومحكمة الهدم قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور Perturbation Court ومحكمة الهدم سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور Perturbation Court ومحكمة الهدم في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

21. World-Effect Loop: معيار الحقيقة العملية

الجواب لا يكتمل عند توليده. World-Effect Loop تسأل: ماذا تغير بسبب هذا الجواب أو الفعل؟ هل زادت دقة الذاكرة؟ هل بنيت أداة؟ هل نجح المستخدم؟ هل ظهر فشل؟ هذه الحلقة تربط النموذج بالعالم.

قبل الفعل تسجل الحالة الأولية، وبعد الفعل تسجل الحالة الناتجة، ثم يقاس metric_delta. قد يكون الأثر جودة وثيقة، نجاح تجربة، انخفاض تكلفة، وضوح قرار، أو اكتشاف خطأ. الأثر اللفظي وحده لا يكفي.

World Effect Object يحمل before_state وafter_state وobserved_metric وuser_feedback وfailure_signal وnext_patch. هذا الكائن يغذي الذاكرة ويعدل أوزان MCD والراوتر والمحكمة.

إذا كان الأثر غير قابل للملاحظة، تخفض الدعوى. وإذا كان الأثر سلبيا، لا يدفن في log؛ بل يتحول إلى اعتراض وإلى Memory Patch. الفشل هنا ليس نهاية، بل مادة تعلم.

هذه الحلقة هي الفرق بين نظام يولد نصوصا ونظام يتعلم من أثره. إنها مركز neoAGI كتشغيل طويل النفس لا كجلسة واحدة.

كل وحدة في HAKIM يجب أن تمتلك test fixture. يعطى لها مدخل معروف ومخرج متوقع، ثم تختبر بمعزل عن بقية النظام. إذا فشل الجواب النهائي، يمكن حينها تحديد هل الخلل في الاستخراج أم الخريطة أم الوكالة أم الأثر، وهذا شرط أساسي لبناء نظام قابل للصيانة.

الربط بين الوحدات يجب ألا يكون نصيا فقط. كل انتقال يحتاج schema. عندما ينتقل Difference Object إلى Mapper لا يفقد مصدره ورتبته. وعندما ينتقل Candidate Action إلى Court لا يفقد معيار نجاحه. حفظ الحقول عبر المراحل هو ما يحمي النظام من فقد السياق.

من جهة الوضع الأنطولوجي، يعامل محور World-Effect Loop: معيار الحقيقة العملية بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور World-Effect Loop: معيار الحقيقة العملية إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور World-Effect Loop: معيار الحقيقة العملية داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور World-Effect Loop: معيار الحقيقة العملية إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور World-Effect Loop: معيار الحقيقة العملية أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور World-Effect Loop: معيار الحقيقة العملية لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور World-Effect Loop: معيار الحقيقة العملية وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور World-Effect Loop: معيار الحقيقة العملية قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور World-Effect Loop: معيار الحقيقة العملية سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور World-Effect Loop: معيار الحقيقة العملية في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

22. Self-Reconstruction Memory: الذاكرة التي تصلح نفسها

الذاكرة في MICAI لا تحفظ الإجابات فقط، بل تحفظ كيف انكسرت الإجابات وكيف أصلحت. Self-Reconstruction Memory تجعل النظام قادرا على تحديث تمثيله وقواعده وأوزانه بعد المحكمة والأثر.

كل Patch يحدد السبب: routing policy change، score recalibration، rule split، concept refinement، source correction، tool failure lesson، court pattern addition. الذاكرة لا تتضخم عشوائيا، بل تعيد بناء نفسها حول الفروق التي ثبت أثرها.

Memory Patch يحمل target_node وchange_type وold_value وnew_value وevidence وtrigger وexpiry_or_review_date. وجود expiry مهم لأن بعض الدروس ظرفية ولا يجب أن تتحول إلى قانون دائم.

الخطر هو أن تتعلم الذاكرة من حادثة واحدة فتتصلب. لذلك تميز بين patch مؤقت وقاعدة مؤكدة. وتطلب تكرار الأثر قبل رفع الثقة. كما تحفظ الاعتراضات التي لم تحسم بدل حذفها.

بهذا تصبح HyperKB طبقة حية، لا أرشيفا. ويتحول MICAI إلى نظام يتراكم داخليا: يسأل أفضل، يوجه موارد أفضل، ويهدم نفسه بصرامة أكبر.

القيمة العملية للذاكرة لا تظهر في حجمها، بل في قدرتها على منع إعادة العمل. لذلك لا تسجل HyperKB كل شيء بنفس الوزن؛ تسجل ما يغير سؤالا أو يعدل قاعدة أو يثبت فشلا أو يحسن أداة. الذكاء هنا ليس الامتلاء، بل تنظيم الامتلاء عبر MCD وQMCD.

في التنفيذ، تحتاج الذاكرة إلى سياسات انتهاء ومراجعة. بعض الدروس تخص مجالا أو مستخدما أو فترة زمنية. إذا تحولت كل ملاحظة إلى قاعدة أبدية فسوف تصبح الذاكرة متصلبة. لذلك يضاف لكل patch scope وconfidence وreview_condition.

من جهة الوضع الأنطولوجي، يعامل محور Self-Reconstruction Memory: الذاكرة التي تصلح نفسها بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور Self-Reconstruction Memory: الذاكرة التي تصلح نفسها إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور Self-Reconstruction Memory: الذاكرة التي تصلح نفسها داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور Self-Reconstruction Memory: الذاكرة التي تصلح نفسها إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور Self-Reconstruction Memory: الذاكرة التي تصلح نفسها أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور Self-Reconstruction Memory: الذاكرة التي تصلح نفسها لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور Self-Reconstruction Memory: الذاكرة التي تصلح نفسها وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور Self-Reconstruction Memory: الذاكرة التي تصلح نفسها قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور Self-Reconstruction Memory: الذاكرة التي تصلح نفسها سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور Self-Reconstruction Memory: الذاكرة التي تصلح نفسها في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

23. MICAI كمنظومة تشغيل لا كبرومبت مطول

الفرق بين MICAI وprompt متقدم أن MICAI يملك حالة وذاكرة ووحدات ومقاييس ومراقبة وأثر. البرومبت يصف سلوكا مرغوبا، أما النظام فينفذه عبر عقود وlogs وschemas وسياسات routing.

يتكون runtime من API Gateway، وTask Queue، وEvent Bus، وMemory Services، وModel Router، وTool Executor، وCourt Service، وEvaluation Service، وDashboard. كل خدمة قابلة للاستبدال، لكن العقود تبقى ثابتة.

Run Trace هو الكائن الجامع. يبدأ من user_request وينتهي بworld_effect، وبينهما question_graph وdeltas وrouting_decisions وtool_calls وobjections وpatches. هذا trace هو ما يسمح بإعادة التشغيل والتحقيق.

إذا ادعى MICAI أنه نظام وهو لا يملك trace ولا state ولا evaluation، فهو مجرد وصف. لذلك يشترط في النسخة التقنية الدنيا أن تسجل القرارات الداخلية الظاهرة القابلة للتفتيش، دون ادعاء كشف سلسلة تفكير خفية.

بهذا يمكن بناء MICAI فوق نماذج مختلفة: نموذج تفكير، نموذج كود، نموذج رؤية، نموذج صوت، أو أدوات بحث. النظرية ليست بديلا عن المزودين، بل عقل تنظيمي فوقهم.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور MICAI كمنظومة تشغيل لا كبرومبت مطول بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور MICAI كمنظومة تشغيل لا كبرومبت مطول إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور MICAI كمنظومة تشغيل لا كبرومبت مطول داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور MICAI كمنظومة تشغيل لا كبرومبت مطول إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور MICAI كمنظومة تشغيل لا كبرومبت مطول أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور MICAI كمنظومة تشغيل لا كبرومبت مطول لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور MICAI كمنظومة تشغيل لا كبرومبت مطول وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور MICAI كمنظومة تشغيل لا كبرومبت مطول قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور MICAI كمنظومة تشغيل لا كبرومبت مطول سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور MICAI كمنظومة تشغيل لا كبرومبت مطول في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

24. HyperKnowledge OS: من قاعدة معرفة إلى نظام تشغيل معرفي

HyperKB OS هو طبقة المعرفة التي تجعل MICAI طويل النفس. القاعدة العادية تخزن، أما HyperKB فتسترجع، وتربط، وتضغط، وتولد مهارات، وتحفظ الاعتراضات، وتتعلم من الأثر. إنها ليست مستودعا بل نظام تشغيل معرفي.

يتكون HyperKB من ingestion pipelines، وsemantic store، وknowledge graph، وsource registry، وhypothesis registry، وskills library، وcourt archive، وmemory patches. عند كل إدخال، لا يسأل النظام: أين أخزنه فقط؟ بل يسأل: ما رتبته؟ ما علاقاته؟ ما السؤال الذي يخدمه؟

العلاقة المركزية في HyperKB هي أن كل مفهوم يرتبط بنظرية، وكل نظرية ترتبط بأداة، وكل أداة ترتبط بمقياس، وكل مقياس يرتبط باعتراض، وكل اعتراض يرتبط بpatch. هذا يخلق سلسلة استعمال لا أرشفة.

خطر HyperKB هو التراكم بلا حاكمية. إذا كبرت الذاكرة دون MCD/QMCD صارت مقبرة نصوص. لذلك تحتاج إلى pruning وcompression وsource validation وcontradiction tracking.

MICAI لا يمكن أن يكون فتّاحا بلا HyperKB، لأن فتح الأسئلة يحتاج ذاكرة أسئلة وأدلة وفشل. HyperKB هو الجسد الطويل للنظام، وHAKIM هو أعصابه التشغيلية.

القيمة العملية للذاكرة لا تظهر في حجمها، بل في قدرتها على منع إعادة العمل. لذلك لا تسجل HyperKB كل شيء بنفس الوزن؛ تسجل ما يغير سؤالا أو يعدل قاعدة أو يثبت فشلا أو يحسن أداة. الذكاء هنا ليس الامتلاء، بل تنظيم الامتلاء عبر MCD وQMCD.

في التنفيذ، تحتاج الذاكرة إلى سياسات انتهاء ومراجعة. بعض الدروس تخص مجالا أو مستخدما أو فترة زمنية. إذا تحولت كل ملاحظة إلى قاعدة أبدية فسوف تصبح الذاكرة متصلبة. لذلك يضاف لكل patch scope وconfidence وreview_condition.

من جهة الوضع الأنطولوجي، يعامل محور HyperKnowledge OS: من قاعدة معرفة إلى نظام تشغيل معرفي بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغيّر، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور HyperKnowledge OS: من قاعدة معرفة إلى نظام تشغيل معرفي إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور HyperKnowledge OS: من قاعدة معرفة إلى نظام تشغيل معرفي داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور HyperKnowledge OS: من قاعدة معرفة إلى نظام تشغيل معرفي إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور HyperKnowledge OS: من قاعدة معرفة إلى نظام تشغيل معرفي أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور HyperKnowledge OS: من قاعدة معرفة إلى نظام تشغيل معرفي لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور HyperKnowledge OS: من قاعدة معرفة إلى نظام تشغيل معرفي وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور HyperKnowledge OS: من قاعدة معرفة إلى نظام تشغيل معرفي قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور HyperKnowledge OS: من قاعدة معرفة إلى نظام تشغيل معرفي سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور HyperKnowledge OS: من قاعدة معرفة إلى نظام تشغيل معرفي في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

25. GraphRAG وRAG الحاكم بالسؤال

RAG العادي يسترجع مقاطع قريبة دلاليا؛ GraphRAG الحاكم في MICAI يسترجع ما يغير بنية الجواب بالنسبة للسؤال. الفارق أن الصلة ليست semantic similarity فقط، بل علاقة سؤال وفرق ودليل واعتراض.

عند الاسترجاع، يرسل Question Compiler queries متعددة: دلالية، وgraph، ومصدرية، واعتراضية. ثم يعاد ترتيب النتائج عبر QMCD: هل تدعم؟ هل تعارض؟ هل تفتح شرط حدود؟ هل تكشف فجوة؟ هل ترفع أو تخفض MCD؟

Retrieval Packet يحمل passages وgraph_paths وsource_quality وcontradictions وmissing_sources. هذا packet يدخل Global Workspace لا الجواب مباشرة. فالاسترجاع مادة حكم لا مادة نسخ.

الفشل الشائع هو أن embedding يجلب نصوصا تشبه السؤال لكنها لا تجيب عنه. لذلك تقيس الوحدة answer_relevance وevidence_diversity وcontradiction_coverage. كما تعاقب اعتمادا على مصدر واحد في مسائل عالية الخطر.

بهذه الطريقة يصبح RAG تابعا للسؤالية. السؤال هو الذي يحكم الاسترجاع، والفرق هو الذي يرفع قيمة المقطع، والمحكمة هي التي تمنع النص المسترجع من التحول إلى حقيقة بلا تحقق.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور GraphRAG وRAG الحاكم بالسؤال بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور GraphRAG وRAG الحاكم بالسؤال إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور GraphRAG وRAG الحاكم بالسؤال داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور GraphRAG وRAG الحاكم بالسؤال إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور GraphRAG وRAG الحاكم بالسؤال أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور GraphRAG وRAG الحاكم بالسؤال لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور GraphRAG وRAG الحاكم بالسؤال وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور GraphRAG وRAG الحاكم بالسؤال قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور GraphRAG وRAG الحاكم بالسؤال سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور GraphRAG وRAG الحاكم بالسؤال في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

26. HyperSkills Extraction: تحويل المعرفة إلى مهارات

المعرفة التي لا تتحول إلى مهارة تظل خاملة. HyperSkills Extraction يستخرج من الوثائق والمحادثات نمطا قابلا لإعادة التشغيل: كيف نفكر في ورقة، كيف نولد نظرية، كيف نكتب كودا، كيف نهدم فرضية، كيف نبني OS معرفيا.

النظام يميز بين knowledge content وskill procedure. المحتوى يقول ماذا، والمهارة تقول كيف. تستخرج الخطوات، الشروط، المخرجات، الأخطاء، ومؤشرات الجودة، ثم تحفظ كSkill Object يمكن استدعاؤه لاحقا.

Skill Object يحمل trigger_conditions وinputs وprocedure وquality_checks وfailure_modes وexample_runs وdependencies. المهارة الجيدة ليست prompt؛ هي إجراء مع معيار نجاح وسجل فشل.

قد يستخرج النظام مهارة عامة جدا لا تعمل. لذلك تختبر المهارة على حالات متعددة، وتقاس قابليتها لإعادة الإنتاج. إذا احتاجت كل مرة إلى تدخل واسع، فهي ليست مهارة ناضجة بل ملاحظة منهجية.

هذا المحور مهم لمشروعك لأن كل ميغابرومبت قوي يجب ألا يبقى نصا، بل يتحول إلى HyperSkill داخل MICAI: Skill لبناء الورقة، Skill لمحكمة الهدم، Skill لتوليد Question Graph، Skill لتحويل النظرية إلى KB.

القيمة العملية للذاكرة لا تظهر في حجمها، بل في قدرتها على منع إعادة العمل. لذلك لا تسجل HyperKB كل شيء بنفس الوزن؛ تسجل ما يغير سؤالا أو يعدل قاعدة أو يثبت فشلا أو يحسن أداة. الذكاء هنا ليس الامتلاء، بل تنظيم الامتلاء عبر MCD وQMCD.

في التنفيذ، تحتاج الذاكرة إلى سياسات انتهاء ومراجعة. بعض الدروس تخص مجالا أو مستخدما أو فترة زمنية. إذا تحولت كل ملاحظة إلى قاعدة أبدية فسوف تصبح الذاكرة متصلبة. لذلك يضاف لكل patch scope وconfidence وreview_condition.

من جهة الوضع الأنطولوجي، يعامل محور HyperSkills Extraction: تحويل المعرفة إلى مهارات بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور HyperSkills Extraction: تحويل المعرفة إلى مهارات إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور HyperSkills Extraction: تحويل المعرفة إلى مهارات داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور HyperSkills Extraction: تحويل المعرفة إلى مهارات إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور HyperSkills Extraction: تحويل المعرفة إلى مهارات أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور HyperSkills Extraction: تحويل المعرفة إلى مهارات لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور HyperSkills Extraction: تحويل المعرفة إلى مهارات وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور HyperSkills Extraction: تحويل المعرفة إلى مهارات قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور HyperSkills Extraction: تحويل المعرفة إلى مهارات سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور HyperSkills Extraction: تحويل المعرفة إلى مهارات في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

27. Prompt OS وبروتوكولات الوكلاء

Prompt OS في MICAI لا يعني برومبت أطول، بل طبقة بروتوكولات تنظم أدوار الوكلاء وسياقهم وأدواتهم وحدودهم. البرومبت يتحول إلى policy وschema وworkflow بدل أن يبقى تعليمات عامة.

لكل وكيل بطاقة نظام: role، capabilities، forbidden_claims، input_schema، output_schema، court_required، memory_scope. الوكيل البحثي لا يرسل بريدا، والوكيل التنفيذي لا يحسم فرضية علمية، والوكيل النقدي لا يبني قبل أن يهدم.

Agent Contract يحمل identity وtools وpermissions وhandoff_rules وevaluation_metrics. ويحدد Event Bus كيف تنتقل المهمة من مستخرج الفرق إلى الباحث إلى الكاتب إلى المحكمة إلى الناشر.

خطر الوكلاء المتعددين هو تضخم التعقيد وتناقض المخرجات. لذلك يحتاج Prompt OS إلى arbitration: أي وكيل له الحق في الحسم؟ ما معيار قبول الخلاف؟ متى يعاد العمل إلى Question Compiler؟

بهذا يصير المشروع قابلا للتطبيق في Hermes أو Codex أو أي بيئة agentic: ليس بنقل النص فقط، بل ببناء عقود تشغيل للوكلاء حول MCD/QMCD وHAKIM.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور Prompt OS وبروتوكولات الوكلاء بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور Prompt OS وبروتوكولات الوكلاء إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور Prompt OS وبروتوكولات الوكلاء داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور Prompt OS وبروتوكولات الوكلاء إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور Prompt OS وبروتوكولات الوكلاء أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور Prompt OS وبروتوكولات الوكلاء لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور Prompt OS وبروتوكولات الوكلاء وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور Prompt OS وبروتوكولات الوكلاء قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور Prompt OS وبروتوكولات الوكلاء سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور Prompt OS وبروتوكولات الوكلاء في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

28. neoAGI/MICAGI: التشغيل متعدد النماذج والوكلاء

neoAGI في هذه الورقة ليس ادعاء وصول إلى AGI، بل نمط نشر يجعل MICAI يعمل فوق نماذج متعددة وأدوات متعددة وذاكرة طويلة. MICAGI هو التوسيع المعماري الذي يجعل النواة عامة وقابلة للتخصص في مجالات كثيرة.

التشغيل يتطلب Model Registry، Provider Adapter، Cost Monitor، Latency Tracker، Capability Matrix، Safety Policy، وFallback Rules. الراوتر لا يختار المزود بالاسم فقط، بل بالقدرة المقاسة على نوع المهمة: retrieval، vision، coding، reasoning، summarization، أو critique.

Model Capability Record يحمل modality وcontext_window وstrengths وweaknesses وcost_profile وobserved_quality وfailure_patterns. هذه السجلات يجب أن تحدث من الاستخدام الحقيقي لا من الدعاية.

إذا تحول orchestration إلى قائمة نماذج بلا قياس، فهو تسويق لا هندسة. لذلك يقاس كل اختيار: هل النموذج المختار أعطى أثرا أفضل من البديل؟ هل وفّر تكلفة؟ هل زاد الهلوسة؟ هل احتاج محكمة أعمق؟

هنا تتصل نظريتك بفكرة تقليل التكلفة وتعظيم الأثر: Vascular Router يختار النموذج السريع للمسارات الخفيفة، والنموذج العميق للمحاكم أو القرارات الحاسمة، مع تسجيل live averages داخليا عندما تتوفر بيانات تشغيل.

لا تعني تعددية النماذج أن كل مهمة يجب أن تمر على كل نموذج. الحكمة التقنية هي اختيار الحد الأدنى الكافي من الموارد لتحقيق أثر مقبول. لذلك يحفظ الراوتر سجلا لتاريخ أداء كل نموذج في نوع المهمة، لا في متوسط عام مضلل.

الاستراتيجية المثلى غالبا هجينة: نموذج سريع لاستخراج أولي، نموذج عميق للمسائل العالية MCD، ونموذج نقدي أو محكمة لمراجعة الادعاءات. هذا يجعل التكلفة تابعة لقيمة السؤال لا عادة ثابتة.

من جهة الوضع الأنطولوجي، يعامل محور neoAGI/MICAGI: التشغيل متعدد النماذج والوكلاء بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور neoAGI/MICAGI: التشغيل متعدد النماذج والوكلاء إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور neoAGI/MICAGI: التشغيل متعدد النماذج والوكلاء داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رُفعت، أي موارد صُرفت، أي اعتراضات قبلت، وأي أثر سُجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور neoAGI/MICAGI: التشغيل متعدد النماذج والوكلاء إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور neoAGI/MICAGI: التشغيل متعدد النماذج والوكلاء أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة، يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور neoAGI/MICAGI: التشغيل متعدد النماذج والوكلاء لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور neoAGI/MICAGI: التشغيل متعدد النماذج والوكلاء وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور neoAGI/MICAGI: التشغيل متعدد النماذج والوكلاء قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور neoAGI/MICAGI: التشغيل متعدد النماذج والوكلاء سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور neoAGI/MICAGI: التشغيل متعدد النماذج والوكلاء في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

29. التدريب والتكييف: من prompts إلى data flywheel

تطوير MICAI لا يتم فقط بكتابة تعليمات أفضل، بل ببناء دولاب بيانات من الأسئلة والفروق والاعتراضات والأثر. كل تشغيل ناجح أو فاشل ينتج أمثلة تدريبية قابلة للاستعمال في تحسين الراوتر والمحكمة والاسترجاع.

تجمع البيانات من Run Traces: الأسئلة التي فشلت، الفروق التي كانت ضحيجا، الاعتراضات التي صححت النموذج، والقرارات التي أنتجت أثرا. ثم تبنى مجموعات تقييم وتفضيل، وقد تستعمل في fine-tuning أو prompt optimization أو retrieval policy learning بحسب الموارد.

Training Example يحمل input_state وchosen_action rejected_action وcourt_feedback وeffect_metric. لا تجمع أمثلة جواب فقط، بل أمثلة قرار بين بدائل. هذا يقرب التدريب من جوهر MCAI: اختيار فعل تحت تعقيد.

الخطر هو تدريب النظام على أخطاء المستخدم أو تفضيلاته اللحظية. لذلك تحتاج الأمثلة إلى رتبة ثقة، ومراجعة بشرية للعينات العالية الأثر، وفصل بين ذاكرة شخصية وذاكرة عامة.

بهذا ينتقل MICAI من مشروع معرفي إلى نظام يتحسن. MCD يتعلم أوزانه، QMCD يتعلم جودة الأسئلة، الراوتر يتعلم سياسة التكلفة، والمحكمة تتعلم أنماط الفشل المتكررة.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور التدريب والتكييف: من prompts إلى data flywheel بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور التدريب والتكييف: من prompts إلى data flywheel إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور التدريب والتكييف: من prompts إلى data flywheel داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور التدريب والتكييف: من prompts إلى data flywheel إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضحيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور التدريب والتكييف: من prompts إلى data flywheel أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور التدريب والتكييف: من prompts إلى data flywheel لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور التدريب والتكييف: من prompts إلى data flywheel وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور التدريب والتكييف: من prompts إلى data flywheel قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور التدريب والتكييف: من prompts إلى data flywheel سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور التدريب والتكييف: من prompts إلى data flywheel في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

30. التقييم والاختبارات: من الانطباع إلى بروتوكول

لا نثبت قيمة MICAI بالقول إنه عميق، بل بمقارنات واضحة. يجب بناء harness يقارن بين baseline عادي ونظام مزود بMCD/QMCD/HAKIM، عبر مهام ورقية وتقنية ومعرفية وعملية.

المهام تشمل: توليد ورقة تركيبية، تصحيح نظرية، بناء KB من ملف، اختيار أداة، حل تعارض مصادر، كتابة كود مع اختبار، وتحويل feedback إلى memory patch. تقاس النتائج عبر rubrics: الأصالة المضبوطة، غياب الحشو، الدقة المصدرية، قوة المحكمة، قابلية التنفيذ، والأثر.

Evaluation Run يحمل task وbaseline_output وmicai_output وhuman_scores وautomated_metrics وfailure_analysis. يجب ألا تقيم النسخة نفسها بلا منافس؛ المقارنة هي التي تكشف قيمة الطبقات.

قد تبدو MICAI أفضل لأنها أطول. لذلك يجب عزل الطول عن الجودة، ومقارنة مخرجات بنفس حد الكلمات أو نفس الوقت أو نفس التكلفة. كما يجب تقييم هل الإضافات زادت الفهم أم زادت التعقيد فقط.

هذا المحور يحول المشروع إلى برنامج بحث. كل نظرية فرعية يجب أن يكون لها اختبار: MCD باختيار الفروق، QMCD بجودة الأسئلة، Court بانخفاض الأخطاء، World Loop بتحسن الأثر، Memory بتحسن runs اللاحقة.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور التقييم والاختبارات: من الانطباع إلى بروتوكول بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور التقييم والاختبارات: من الانطباع إلى بروتوكول إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور التقييم والاختبارات: من الانطباع إلى بروتوكول داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور التقييم والاختبارات: من الانطباع إلى بروتوكول إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور التقييم والاختبارات: من الانطباع إلى بروتوكول أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور التقييم والاختبارات: من الانطباع إلى بروتوكول لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور التقييم والاختبارات: من الانطباع إلى بروتوكول وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور التقييم والاختبارات: من الانطباع إلى بروتوكول قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور التقييم والاختبارات: من الانطباع إلى بروتوكول سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور التقييم والاختبارات: من الانطباع إلى بروتوكول في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

31. محاربة الهلوسة وفقد السياق

الهلاوس في نظام معرفي ليست أخطاء معلومات فقط؛ قد تكون خلط رتبة، أو تضخيم فرضية، أو علاقة بلا دليل، أو أثر مدعى بلا قياس. لذلك يعالج MICAI الهلوسة عبر طبقات لا عبر تحذير لغوي واحد.

الطبقات هي source registry، rank-of-knowledge tagging، evidence requirements، contradiction checks، court and memory of known uncertainty، before claim. إذا كان المصدر غير متاح، يصرح النظام بأنه يبني من المادة الظاهرة فقط. وإذا احتاج معلومة حديثة، يفصلها عن البناء النظري.

Claim Record يحمل claim_text وevidence_state وknowledge_rank وsource_links وcounterclaims وconfidence. لا يسمح للclaim عالي الأثر بالمرور بلا evidence_state واضح.

أخطر هلوسة هي هلوسة الأصالة: أن يسمى تركيب قديم باسم جديد. لذلك تسأل المحكمة دائما عن النسب المعرفي: ما الموجود سابقا؟ ما الجديد الحقيقي؟ هل الجديد مفهوم أم علاقة أم أداة أم مقياس؟

بهذا يصبح ضبط الهلوسة جزءا من MCD/QMCD لا طبقة خارجية. الضجيج N يعاقب الادعاء الزائد، والسؤال يحكم الدليل، والمحكمة تخفض الرتبة عندما يسبق الحماس البرهان.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور محاربة الهلوسة وفقد السياق بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور محاربة الهلوسة وفقد السياق إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور محاربة الهلوسة وفقد السياق داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور محاربة الهلوسة وفقد السياق إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور محاربة الهلوسة وفقد السياق أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور محاربة الهلوسة وفقد السياق لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور محاربة الهلوسة وفقد السياق وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور محاربة الهلوسة وفقد السياق قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور محاربة الهلوسة وفقد السياق سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور محاربة الهلوسة وفقد السياق في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

32. واجهات API والعقود الداخلية

النظام التقني يحتاج واجهات صريحة. لا يمكن تشغيل MICAI ككتلة واحدة غامضة؛ يجب تعريف endpoints أو functions لكل وحدة حتى يمكن اختبارها واستبدالها وتوسيعها.

الواجهات الأساسية: ingest(input)، extract_differences(context_id)، build_question_graph(task_id)، score_mcd(delta_id)، route_task(task_id)، run_court(candidate_id)، execute_action(action_id)، patch_memory(effect_id)، record_effect(action_id). كل واجهة تعيد كائنا لا نصا فقط.

العقد يعتمد JSON-like schemas لكنه يحفظ العربية في الحقول النصية. الأهم هو فصل text عن metadata. النص يعطي المعنى، والmetadata يعطي الحاكمية: رتبة، مصدر، ثقة، أداة، أثر، ونسخة.

واجهة بلا validation مصدر فشل. لذلك يجب أن ترفض الواجهة Question Object بلا criteria، أو Action بلا success_metric، أو Claim بلا rank، أو Effect بلا before/after. الرفض هنا حماية للنظام.

هذه العقود تجعل المشروع قابلا للتطبيق في backend حقيقي: Python/FastAPI أو Node/TypeScript، مع workers للنماذج والأدوات، وقاعدة graph/vector، وobservability dashboard.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور واجهات API والعقود الداخلية بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور واجهات API والعقود الداخلية إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور واجهات API والعقود الداخلية داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور واجهات API والعقود الداخلية إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور واجهات API والعقود الداخلية أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور واجهات API والعقود الداخلية لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور واجهات API والعقود الداخلية وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور واجهات API والعقود الداخلية قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور واجهات API والعقود الداخلية سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور واجهات API والعقود الداخلية في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

33. المكدس البرمجي المقترح

المكدس لا يجب أن يقيد النظرية، لكنه يقدم نسخة تنفيذ أولى. المهم أن يدعم graph وvector وevent bus وworkers وlogs وواجهات نماذج متعددة، مع قدرة على تشغيل محاكم وتقييمات متكررة.

يمكن بناء النواة بخدمة API، وقاعدة PostgreSQL للحقول العلائقية، وvector store للاسترجاع الدلالي، وgraph database أو طبقة graph فوق PostgreSQL للعلاقات، وqueue للمهام، وobject storage للملفات، وdashboard للمراقبة. نماذج التفكير والكود والرؤية تدخل عبر Provider Adapter.

كل run يحفظ في قاعدة traces. وكل وثيقة تدخل source registry. وكل أداة تحمل Tool Contract. وكل مزود نموذج يحمل Capability Record. هذا التنظيم يجعل النظام قابلا للتدقيق والمقارنة.

الخطر أن يصبح المكدس كبيرا قبل إثبات القيمة. لذلك تبدأ النسخة MVP بثلاث وحدات: Court، MCD Scorer، Question Graph Engine، فوق RAG بسيط. ثم يضاف الراوتر والذاكرة الذاتية بعد إثبات الفرق.

بهذا لا يتحول المشروع إلى بناء ضخم غير قابل للإنجاز. يبدأ صغيرا لكنه يحفظ هندسة التوسع: كل طبقة لها مكان حتى لو نفذت لاحقا.

لا تعني تعددية النماذج أن كل مهمة يجب أن تمر على كل نموذج. الحكمة التقنية هي اختيار الحد الأدنى الكافي من الموارد لتحقيق أثر مقبول. لذلك يحفظ الراوتر سجلا لتاريخ أداء كل نموذج في نوع المهمة، لا في متوسط عام مضلل.

الاستراتيجية المثلى غالبا هجينة: نموذج سريع لاستخراج أولي، نموذج عميق للمسائل العالية MCD، ونموذج نقدي أو محكمة لمراجعة الادعاءات. هذا يجعل التكلفة تابعة لقيمة السؤال لا عادة ثابتة.

من جهة الوضع الأنطولوجي، يعامل محور المكدس البرمجي المقترح بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور المكدس البرمجي المقترح إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور المكدس البرمجي المقترح داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور المكدس البرمجي المقترح إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور المكدس البرمجي المقترح أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور المكدس البرمجي المقترح لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور المكدس البرمجي المقترح وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور المكدس البرمجي المقترح قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور المكدس البرمجي المقترح سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور المكدس البرمجي المقترح في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

34. المجدول وEvent Bus وحلقة العمل

النظام الوكالي يحتاج زمنا تشغيليا. Event Bus يجعل الوحدات تتفاعل دون اعتماد مباشر، والمجدول يسمح بتشغيل مهام مراقبة وتحديث وتقييم وإعادة فهرسة دون تدخل دائم.

الأحداث الأساسية: route.decided، question.compiled، deltas.detected، atoms.created، input.received، memory.patched، effect.recorded، action.executed، court.completed. كل حدث يطلق وحدة أو أكثر، مع trace_id موحد.

Event Record يحمل type وpayload_ref وtimestamp وactor وstatus وerror. المجدول يطلق model، periodic evaluations، contradiction scan، memory pruning، capability refresh، question library review.

التشغيل المستمر قد يضاعف الأخطاء إذا لم توجد حدود. لذلك يجب وضع rate limits، وحواجز actions الخارجية، ومراجعة بشرية للمهام العالية الأثر، وقواعد توقف عند ارتفاع failure signals.

Event Bus هو الجهاز الدوري لMCD .MICAI يحدد الأهمية، QMCD يحدد السؤال، الراوتر يحدد المسار، والمحكمة تحدد هل يستمر الحدث أم يعود للتعديل.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور المجدول وEvent Bus وحلقة العمل بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور المجدول وEvent Bus وحلقة العمل إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور المجدول وEvent Bus وحلقة العمل داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور المجدول وEvent Bus وحلقة العمل إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور المجدول وEvent Bus وحلقة العمل أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور المجدول وEvent Bus وحلقة العمل لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور المجدول وEvent Bus وحلقة العمل وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور المجدول وEvent Bus وحلقة العمل قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور المجدول وEvent Bus وحلقة العمل سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور المجدول وEvent Bus وحلقة العمل في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

35. النظريات الكبرى المقابلة للوحدات

كل وحدة في النظام تحمل نظرية كبرى. ليست النظرية عنوانا مستقلا، بل تفسير لماذا تحتاج الوحدة ولماذا تعمل. بهذا نحفظ النظريات داخل المعمارية بدل أن نضعها في قائمة منفصلة.

نظرية الفرق الفاعل تقابل Difference Extractor. نظرية الحاكمية السؤالانية تقابل QMCD وQuestion Compiler. نظرية الوكالة الميتا-تعقيدية تقابل Agency Selector. نظرية التوجيه الوعائي تقابل Router. نظرية الأثر العالمي تقابل World Loop. نظرية الذاكرة المعيدة للبناء تقابل Memory. نظرية محكمة الهدم تقابل Court.

Theory Record يحمل parent_system وtechnical_layer وcore_mechanism وfailure_mode وtest_protocol. لا تدخل النظرية الكبرى الورقة إلا ومعها طبقة تقنية تقابلها.

إذا عجزنا عن ربط نظرية بوحدة أو مقياس، فهي إما فلسفة عامة أو فرع يحتاج نضجا. لا يحذف ذلك قيمتها، لكنه يخفض رتبتها من نظرية تشغيلية إلى عدسة تحليلية.

هذه الخريطة تمنع ضياع المشروع. كل اسم يجد مكانه: MCD في القياس، QMCD في السؤال، MCAI في الوكالة، HAKIM في التنفيذ، HyperKB في الذاكرة، neoAGI في النشر.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور النظريات الكبرى المقابلة للوحدات بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور النظريات الكبرى المقابلة للوحدات إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور النظريات الكبرى المقابلة للوحدات داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور النظريات الكبرى المقابلة للوحدات إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور النظريات الكبرى المقابلة للوحدات أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور النظريات الكبرى المقابلة للوحدات لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور النظريات الكبرى المقابلة للوحدات وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور النظريات الكبرى المقابلة للوحدات قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور النظريات الكبرى المقابلة للوحدات سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور النظريات الكبرى المقابلة للوحدات في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

36. النظريات الفرعية: من العنوان إلى الوظيفة

النظريات الفرعية هي جسور دقيقة بين النظرية الكبرى والتطبيق. أهميتها أنها تمنع الفجوة بين الفكرة العظمى والكود. كل فرع يضبط مفهوما أو علاقة أو فشلا محددا.

من الفروع اللازمة: نظرية Question-to-Tool التي تدرس كيف يتحول السؤال إلى أداة، نظرية Difference-to-Memory التي تحدد متى يحفظ الفرق، نظرية Court-to-Patch التي تحول الاعتراض إلى تعديل، نظرية Cost-to-Cognition التي تربط الموارد بجودة التفكير، ونظرية Source-to-Rank التي تمنع خلط المصدر باليقين.

SubTheory Record يحمل scope وinputs وoutputs وmeasurement وboundary. وظيفة هذا السجل أن يجعل الفرع قابلا للتوسع لاحقا دون أن يبتلع النظرية الأم.

خطر الفروع أنها تتكاثر حتى تضيع النواة. لذلك يشترط أن يحل كل فرع مشكلة تشغيلية محددة. الفرع الذي لا يغير pipeline ولا يحسن اختبارا يدمج مع غيره.

بهذا يصبح المشروع مدرسة تقنية ناشئة لا مجموعة أسماء. الكبرى تعطي العمود، والفرعية تعطي المفاصل، والأدوات تعطي اليد، والمقاييس تعطي الحكم.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور النظريات الفرعية: من العنوان إلى الوظيفة بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور النظريات الفرعية: من العنوان إلى الوظيفة إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور النظريات الفرعية: من العنوان إلى الوظيفة داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور النظريات الفرعية: من العنوان إلى الوظيفة إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور النظريات الفرعية: من العنوان إلى الوظيفة أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور النظريات الفرعية: من العنوان إلى الوظيفة لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور النظريات الفرعية: من العنوان إلى الوظيفة وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور النظريات الفرعية: من العنوان إلى الوظيفة قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور النظريات الفرعية: من العنوان إلى الوظيفة سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور النظريات الفرعية: من العنوان إلى الوظيفة في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

37. الأدوات التقنية العشر المركزية

الأداة هي اختبار صدق النظرية. إذا لم تنتج النظرية أداة فهي لم تدخل الهندسة بعد. في MICAI، الأدوات ليست إضافات، بل أجسام تشغيلية للفروق والأسئلة والمحكمة والذاكرة.

الأدوات المركزية هي Court Engine، Difference Miner، Question Compiler، Question Graph Builder، MCD Scorer، Memory Patch Manager، World Effect Logger، HyperSkill Extractor، Vascular Router، Evidence Ranker. كل أداة لها مدخلات وخطوات ومخرجات وحدود.

Tool Contract يحمل name وpurpose وinput_schema وoutput_schema وquality_metric وfailure_mode. هذا يجعل الأداة قابلة للاستبدال: يمكن تطوير MCD Scorer دون كسر Question Compiler إذا حفظ العقد.

الأداة التي لا تقاس لا تعتمد. يجب أن نعرف هل تقلل الحشو، هل تحسن الدقة، هل تسرع القرار، هل تكشف الفشل، أم تزيد تعقيدا فقط. الأداة التي لا تضيف أثرها تزال.

هذه الأدوات هي الحد الأدنى لتحويل الورقة إلى مشروع برمجي. يمكن بناء بعضها كخدمات، وبعضها كـnotebooks، وبعضها كـprompt-protocols في البداية، لكن كلها يجب أن تنتج سجلات قابلة للتحليل.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور الأدوات التقنية العشر المركزية بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور الأدوات التقنية العشر المركزية إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور الأدوات التقنية العشر المركزية داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور الأدوات التقنية العشر المركزية إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور الأدوات التقنية العشر المركزية أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور الأدوات التقنية العشر المركزية لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور الأدوات التقنية العشر المركزية وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور الأدوات التقنية العشر المركزية قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور الأدوات التقنية العشر المركزية سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور الأدوات التقنية العشر المركزية في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

38. المقاييس والمؤشرات: ضبط الخرق لا قتله

المقاييس في MICAI لا تزعم قياس كل شيء بدقة مخبرية، لكنها تمنع الفوضى. وظيفتها تقدير جودة الفرق والسؤال والنظرية والأداة والأثر، ثم تحويل الانطباع إلى مقارنة قابلة للتطوير.

المؤشرات الأساسية هي مؤشر الخرق المعرفي المنضبط، مؤشر نضج المفهوم، مؤشر الحاكمية النظرية، مؤشر ضغط المعرفة، مؤشر مقاومة الهدم، مؤشر أثر العالم، ومؤشر اقتصاد الوكالة. كل مؤشر يتكون من أبعاد فرعية تقيم ordinally في البداية.

Metric Record يحمل definition وsubmetrics وscale وcalibration_examples وknown_biases. وجود known_biases مهم لأن كل مقياس قد يفضل الطول أو الغرابة أو الصرامة الشكلية على الأثر الحقيقي.

إذا استخدم المقياس لتزيين الورقة لا للحكم، فهو ضجيج. لذلك يجب أن ينتج قرارا: قبول، تعديل، تخفيض رتبة، طلب مصدر، أو تشغيل تجربة. المقياس الذي لا يغير شيئا لا حاجة له.

المقاييس تربط النظرية بمحكمة الهدم والبرنامج البحثي. وهي تجعل النسخ القادمة من MICAI قابلة للمقارنة: هل تحسنت جودة الأسئلة؟ هل قل N؟ هل زاد أثر العالم؟

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور المقاييس والمؤشرات: ضبط الخرق لا قتله بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور المقاييس والمؤشرات: ضبط الخرق لا قتله إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور المقاييس والمؤشرات: ضبط الخرق لا قتله داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور المقاييس والمؤشرات: ضبط الخرق لا قتله إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور المقاييس والمؤشرات: ضبط الخرق لا قتله أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور المقاييس والمؤشرات: ضبط الخرق لا قتله لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور المقاييس والمؤشرات: ضبط الخرق لا قتله وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور المقاييس والمؤشرات: ضبط الخرق لا قتله قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور المقاييس والمؤشرات: ضبط الخرق لا قتله سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور المقاييس والمؤشرات: ضبط الخرق لا قتله في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

39. الفرضيات البحثية الفارقة

الفرضية الجيدة في MICAI ليست عبارة عامة عن علاقة، بل دعوى تميز النموذج عن بديل. لذلك تصاغ الفرضيات حول أثر طبقات معينة: ماذا يحدث عندما نضيف QMCD؟ ماذا يحدث عندما نضيف Court؟ ماذا يحدث عندما نحفظ Memory Patches؟

فرضية قوية: إضافة Question Graph قبل RAG تحسن relevance وتقلل drift في المهام التركيبية. فرضية ثانية: محكمة الهدم قبل الإخراج تقلل الادعاءات الزائدة لكنها قد تزيد زمن التنفيذ. فرضية ثالثة: MCD Scoring يحسن ترتيب العناصر المهمة في أوراق طويلة مقارنة بالترتيب الدلالي وحده.

Hypothesis Record يحمل mechanism وexpected_effect وalternative_model وtest_method وbreaking_result. النتيجة التي تكسر الفرضية يجب أن تكتب قبل التجربة حتى لا نؤول كل شيء لصالح النموذج.

قد تفسر النماذج المنافسة التحسن بمجرد زيادة الوقت أو الطول أو تعدد المحاولات. لذلك يجب ضبط التكلفة والزمن وعدد tokens عند المقارنة، وإلا نسبت MICAI لنفسه أثرا سببه موارد إضافية.

الفرضيات تحول المشروع إلى برنامج بحث. كل نسخة جديدة من PDF أو نظام يجب أن تغلق فرضية أو تفتح أخرى، لا أن تزيد أسماء فقط.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور الفرضيات البحثية الفارقة بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور الفرضيات البحثية الفارقة إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور الفرضيات البحثية الفارقة داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور الفرضيات البحثية الفارقة إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور الفرضيات البحثية الفارقة أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور الفرضيات البحثية الفارقة لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور الفرضيات البحثية الفارقة وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور الفرضيات البحثية الفارقة قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور الفرضيات البحثية الفارقة سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور الفرضيات البحثية الفارقة في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

40. التنبؤات الفارقة

التنبؤ الفارق هو ما يرجح MICAI على بديله إذا ظهر. يجب ألا تكون التنبؤات عامة يستطيع أي نظام ادعاءها، بل مرتبطة مباشرة بوحدات MCD/QMCD/HAKIM.

يتنبأ MICAI أن المهام ذات الغموض العالي ستستفيد أكثر من QMCD من المهام البسيطة، وأن محكمة الهدم ستخفض نبرة اليقين الزائد في الأوراق النظرية، وأن World Effect Loop سيحسن runs اللاحقة عبر patches، وأن Vascular Router سيخفض التكلفة دون خفض الجودة إذا بنيت capability records جيدة.

Prediction Record يحمل conditions وobservable_metric وcompeting_prediction وfailure_threshold. مثلا إذا لم يحسن Question Graph جودة الاسترجاع في المهام المفتوحة بعد ضبط الزمن والكلفة، تضعف فرضية السؤالية التشغيلية.

التنبؤات يجب أن تكون قابلة للهدم. إذا صيغت بحيث كل نتيجة تؤيد النموذج فهي ليست تنبؤات. لذلك تكتب النتائج التي تخفض ثقة النظرية بوضوح.

بهذا يدخل المشروع من مستوى الورقة إلى مستوى المختبر: كل تنبؤ يطلب dataset أو benchmark أو مقارنة بشرية أو تجربة ميدانية صغيرة.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور التنبؤات الفارقة بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور التنبؤات الفارقة إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور التنبؤات الفارقة داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور التنبؤات الفارقة إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور التنبؤات الفارقة أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور التنبؤات الفارقة لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور التنبؤات الفارقة وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور التنبؤات الفارقة قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور التنبؤات الفارقة سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور التنبؤات الفارقة في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

41. النماذج المتنافسة الثلاثة

حتى لا تبنى النظرية في فراغ، يجب مقارنتها بثلاثة نماذج. النموذج المحافظ هو RAG/agent عادي مع prompts جيدة. النموذج التركيبي هو multi-agent مع أدوات وتقييم. النموذج الاختراقي هو MICAI الكامل بحاكمية فرق وسؤال ومحكمة وأثر وذاكرة ذاتية.

النموذج المحافظ قوي لأنه بسيط ورخيص ومفهوم، لكنه يضعف في المهام التي تحتاج ترتيب مراتب المعرفة والهدم. النموذج التركيبي أقوى في تقسيم العمل، لكنه قد يضاعف الضجيج إذا لم يملك سؤالا حاكما. MICAI يضيف مبدأ حاكمية ومقاييس وذاكرة، لكنه أكثر تعقيدا ويتطلب بنية بيانات وانضباطا.

Comparison Record يقيم التفسير، الكلفة، القابلية للاختبار، مقاومة الهلوسة، الأصالة، وقابلية التوسع. لا يفوز MICAI في كل شيء؛ يفوز عندما تكون المهمة عميقة ومركبة وعالية الأثر، ويخسر عندما تكون بسيطة وسريعة.

أقوى اعتراض أن MICAI قد يكون over-engineering. الرد ليس إنكارا، بل تحديد شروط الحدود: لا يستعمل MICAI الكامل لكل سؤال، بل عبر Router يفتح طبقات عميقة فقط عندما يبرر MCD/QMCD ذلك.

المقارنة تجعل النظرية متواضعة وقوية: MICAI ليس replacement لكل workflow، بل escalation architecture للمشكلات التي تفشل فيها الطرق الخفيفة.

لا تعني تعددية النماذج أن كل مهمة يجب أن تمر على كل نموذج. الحكمة التقنية هي اختيار الحد الأدنى الكافي من الموارد لتحقيق أثر مقبول. لذلك يحفظ الراوتر سجلا لتاريخ أداء كل نموذج في نوع المهمة، لا في متوسط عام مضلل.

الاستراتيجية المثلى غالبا هجينة: نموذج سريع لاستخراج أولي، نموذج عميق للمسائل العالية MCD، ونموذج نقدي أو محكمة لمراجعة الادعاءات. هذا يجعل التكلفة تابعة لقيمة السؤال لا عادة ثابتة.

من جهة الوضع الأنطولوجي، يعامل محور النماذج المتنافسة الثلاثة بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور النماذج المتنافسة الثلاثة إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور النماذج المتنافسة الثلاثة داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور النماذج المتنافسة الثلاثة إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور النماذج المتنافسة الثلاثة أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور النماذج المتنافسة الثلاثة لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور النماذج المتنافسة الثلاثة وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور النماذج المتنافسة الثلاثة قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور النماذج المتنافسة الثلاثة سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور النماذج المتنافسة الثلاثة في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

42. النظرية النهائية بعد الهدم: الذكاء الفتّاح المحكوم

بعد الهدم، تصاغ النظرية النهائية هكذا: الذكاء الفتّاح المحكوم هو قدرة نظامية على تحويل المدخلات المعقدة إلى فروق حاكمة، ثم إلى أسئلة مولدة، ثم إلى أفعال قابلة للهدم والقياس، مع إعادة بناء الذاكرة بناء على أثر العالم، ضمن حدود موارد ومخاطر معلنة.

الجديد ليس وجود RAG ولا تعدد الوكلاء ولا التقييم وحده، بل ربطها عبر قانون: لا فرق بلا سؤال، لا سؤال بلا معيار، لا فعل بلا محكمة، لا أثر بلا ذاكرة، ولا ذاكرة بلا تخفيض رتبة عند الفشل. هذا القانون هو مركز MICAI.

النظرية تتجسد في خمسة سجلات: Patch، Effect، Action، Question، Difference. إذا أمكن تنفيذ هذه السجلات وربطها وقياسها، أمكن القول إن لدينا نسخة تشغيلية أولى. إذا لم تنفذ، تبقى النظرية برنامج بحث.

ما قد يكسر النظرية أن لا يظهر فرق ملحوظ عن baseline بعد ضبط التكلفة، أو أن تزيد المحاكم التعقيد دون خفض الأخطاء، أو أن تكون الأوزان غير قابلة للمعايرة، أو أن تصبح الذاكرة ثقيلة ومتحيزة.

النسخة المعدلة أكثر تواضعا من الخطاب الأول، لكنها أقوى: MICAI إطار حاكمية معرفية فوق النماذج، لا دعوى AGI مثبتة. قوته تظهر في المهام المركبة التي تحتاج فهم الفروق والسؤال والهدم والأثر.

محكمة الهدم تعمل كاختبار قبل النشر. لا تطلب من النموذج أن يكون متشائما، بل أن يكون دقيقا في حدود ما يدعيه. الفرق بين المحكمة والرفض أن المحكمة تقدم repair path. لذلك كل اعتراض لا ينتج تعديلا أو اختبارا أو تخفيضا للرتبة يعد اعتراضا ناقصا.

يمكن قياس المحكمة بنسبة الأخطاء التي كشفتها قبل المستخدم، وبنسبة الادعاءات التي خفضت رتبتها، وبجودة الإصلاح بعد النقد. إذا زادت المحكمة طول الجواب دون تحسين، فقد تحولت إلى طقس لا أداة، ويجب ضغطها أو تخصيصها للمهام العالية الأثر.

من جهة الوضع الأنطولوجي، يعامل محور النظرية النهائية بعد الهدم: الذكاء الفتّاح المحكوم بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور النظرية النهائية بعد الهدم: الذكاء الفتّاح المحكوم إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور النظرية النهائية بعد الهدم: الذكاء الفتّاح المحكوم داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور النظرية النهائية بعد الهدم: الذكاء الفتّاح المحكوم إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور النظرية النهائية بعد الهدم: الذكاء الفتّاح المحكوم أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور النظرية النهائية بعد الهدم: الذكاء الفتّاح المحكوم لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور النظرية النهائية بعد الهدم: الذكاء الفتّاح المحكوم وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور النظرية النهائية بعد الهدم: الذكاء الفتّاح المحكوم قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور النظرية النهائية بعد الهدم: الذكاء الفتّاح المحكوم سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور النظرية النهائية بعد الهدم: الذكاء الفتّاح المحكوم في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

43. حالة استعمال: إنتاج ورقة بحثية تقنية عميقة

هذه الحالة هي مثال مباشر على سبب بناء MICAI. المدخل هو مادة خام ضخمة وميغابروميت ونظريات كثيرة. الخطر هو إنتاج PDF شكلي مليء بالعناوين والشذرات. الحل هو تشغيل السؤال الحاكم: كيف تتحول كل نظرية إلى طبقة تقنية وفقرات مترابطة؟

يبدأ النظام بتصفية المادة، ثم يبني خريطة النظريات، ثم يستخرج الفروق: السؤالية كانت ناقصة، والفقرات كانت قصيرة، والربط بين النظرية والوحدة ضعيف. ثم يرفع MCD لهذه الفروق لأنها تغير جودة الورقة كلها. بعدها يحكم QMCD بأن المطلوب ليس زيادة صفحات، بل زيادة تركيبية.

Action Objects في هذه الحالة هي: إعادة بناء الفهرس، تحويل النقاط إلى فقرات، إضافة سجلات بيانات، إضافة محكمة هدم، بناء PDF Amiri RTL، والتحقق من الصفحات. World Effect يقاس هنا بعدد الصفحات، كثافة الفقرات، ووضوح الربط بين كل نظرية ووحدة.

الفشل الممكن أن تصبح الورقة طويلة لكنها متكررة. لذلك تضيف المحكمة معيارا: كل فقرة يجب أن تحمل آلية أو علاقة أو اعتراضا أو اختبارا. الفقرة التي لا تفعل ذلك تضغط أو تحذف.

بهذه الحالة يظهر أن MICAI ليس موضوع الورقة فقط، بل طريقة إنتاجها. الورقة نفسها تصبح Run Trace معرفيا لنظامها.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور حالة استعمال: إنتاج ورقة بحثية تقنية عميقة بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغيّر، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور حالة استعمال: إنتاج ورقة بحثية تقنية عميقة إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور حالة استعمال: إنتاج ورقة بحثية تقنية عميقة داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور النظرية النهائية بعد الهدم: الذكاء الفتّاح المحكوم إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور حالة استعمال: إنتاج ورقة بحثية تقنية عميقة أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور حالة استعمال: إنتاج ورقة بحثية تقنية عميقة لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور حالة استعمال: إنتاج ورقة بحثية تقنية عميقة وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور حالة استعمال: إنتاج ورقة بحثية تقنية عميقة قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور حالة استعمال: إنتاج ورقة بحثية تقنية عميقة سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور حالة استعمال: إنتاج ورقة بحثية تقنية عميقة في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

44. حالة استعمال: HyperKB لوكالة Neo Agency

عند تطبيق MICAI على وكالة أو مشروع أعمال، لا يكون الهدف كتابة محتوى فقط، بل بناء Operating Knowledge Base يحول خدمات الوكالة إلى أنظمة: outreach governance، SEO OS، content OS، lead intelligence، وvisual production pipeline.

يدخل وصف العلامة والخدمات والقطاع. Difference Extractor يحدد ما يميز العرض. QMCD يبني أسئلة الجمهور والفانل. HyperKB يحفظ الرسائل والأدلة والنماذج. Agency Selector يختار هل نكتب صفحة هبوط أم بريد أم World Loop .carousel يقيس التحويل أو الرد أو engagement.

لكل حملة Campaign Knowledge Record يحتوي metrics، variants، messages، proof، offer، pain points، audience، objections، وpatches. لا ترسل الرسائل عشوائيا، بل تخضع لقواعد سمعة وامتثال ومراجعة أثر.

الخطر في تطبيقات التسويق أن يتحول النظام إلى تضخيم وعود. محكمة الهدم تضبط الدليل، وتحذف الادعاءات الزائدة، وتطلب opt-out وآلية احترام الردود والسمعة عندما يكون الإرسال بريديا.

هكذا يعمل MICAI كـneo agency kernel: يحول المعرفة إلى عمليات يومية قابلة للقياس، لا إلى ملفات ضخمة لا تستعمل.

القيمة العملية للذاكرة لا تظهر في حجمها، بل في قدرتها على منع إعادة العمل. لذلك لا تسجل HyperKB كل شيء بنفس الوزن؛ تسجل ما يغير سؤالا أو يعدل قاعدة أو يثبت فشلا أو يحسن أداة. الذكاء هنا ليس الامتلاء، بل تنظيم الامتلاء عبر MCD وQMCD.

في التنفيذ، تحتاج الذاكرة إلى سياسات انتهاء ومراجعة. بعض الدروس تخص مجالا أو مستخدما أو فترة زمنية. إذا تحولت كل ملاحظة إلى قاعدة أبدية فسوف تصبح الذاكرة متصلبة. لذلك يضاف لكل patch scope وconfidence وreview_condition.

من جهة الوضع الأنطولوجي، يعامل محور حالة استعمال: HyperKB لوكالة Neo Agency بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور حالة استعمال: HyperKB لوكالة Neo Agency إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور حالة استعمال: HyperKB لوكالة Neo Agency داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور حالة استعمال: HyperKB لوكالة Neo Agency إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور حالة استعمال: HyperKB لوكالة Neo Agency أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور حالة استعمال: HyperKB لوكالة Neo Agency لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور حالة استعمال: HyperKB لوكالة Neo Agency وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور حالة استعمال: HyperKB لوكالة Neo Agency قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور حالة استعمال: HyperKB لوكالة Neo Agency سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور حالة استعمال: HyperKB لوكالة Neo Agency في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

45. حالة استعمال: orchestration لنماذج التفكير والكود والصورة

في بيئات متعددة النماذج، القيمة ليست في امتلاك قائمة طويلة من النماذج، بل في معرفة متى يستعمل كل نموذج ولماذا. MICAI يضع orchestration تحت حكم MCD/QMCD والراوتر لا تحت الانبهار بالمزود.

المهمة تقسم حسب النوع: reasoning عميق، كود، رؤية، صورة، فيديو، تلخيص، أو نقد. لكل نوع يختار الراوتر نموذجين أو أكثر عند الحاجة، ثم تقارن المحكمة المخرجات. في المسارات الرخيصة يستعمل نموذج سريع، وفي المسارات الحاسمة يفتح تحقق أعمق.

Provider Adapter يخفي اختلاف APIs. Capability Record يحدث من نتائج حقيقية. Routing Decision يسجل سبب الاختيار. Evaluation Packet يقارن الجودة والتكلفة والزمن. هذه الكائنات تمنع orchestration من أن يكون عشوائيا.

إذا استعمل النظام أقوى نموذج دائما فهو لا يملك ذكاء تكلفة. وإذا استعمل الأرخص دائما فهو لا يملك حاكمية جودة. الراوتر الناجح يوازن الأثر والكلفة والخطر، ويتعلم من الفشل.

هذه الحالة تربط مشروعك العملي بنظريتك: اقتصاد النماذج يصبح تطبيقا مباشرا للتوجيه الوعائي للحوسبة، لا مجرد إعدادات API.

لا تعني تعددية النماذج أن كل مهمة يجب أن تمر على كل نموذج. الحكمة التقنية هي اختيار الحد الأدنى الكافي من الموارد لتحقيق أثر مقبول. لذلك يحفظ الراوتر سجلا لتاريخ أداء كل نموذج في نوع المهمة، لا في متوسط عام مضلل.

الاستراتيجية المثلى غالبا هجينة: نموذج سريع لاستخراج أولي، نموذج عميق للمسائل العالية MCD، ونموذج نقدي أو محكمة لمراجعة الادعاءات. هذا يجعل التكلفة تابعة لقيمة السؤال لا عادة ثابتة.

من جهة الوضع الأنطولوجي، يعامل محور حالة استعمال: orchestration لنماذج التفكير والكود والصورة بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور حالة استعمال: orchestration لنماذج التفكير والكود والصورة إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور حالة استعمال: orchestration لنماذج التفكير والكود والصورة داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور حالة استعمال: orchestration لنماذج التفكير والكود والصورة إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور حالة استعمال: orchestration لنماذج التفكير والكود والصورة أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور حالة استعمال: orchestration لنماذج التفكير والكود والصورة لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور حالة استعمال: orchestration لنماذج التفكير والكود والصورة وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور حالة استعمال: orchestration لنماذج التفكير والكود والصورة قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور حالة استعمال: orchestration لنماذج التفكير والكود والصورة سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور حالة استعمال: orchestration لنماذج التفكير والكود والصورة في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

46. السلامة والحوكمة التقنية

نظام وكالي قادر على الفعل يحتاج حوكمة. السلامة في MICAI ليست رقابة خارجية فقط، بل جزء من النظرية: الرتبة المعرفية، ومحكمة الهدم، وAction rollback، وWorld Effect كلها أدوات أمان.

تصنف المهام حسب risk tier. النص منخفض الأثر يمر سريعا. الفعل الخارجي يحتاج تحقق. المسائل القانونية أو الطبية أو المالية أو الحساسة تحتاج مصادر حديثة وحدودا واضحة. أتمتة البريد أو البحث أو الأدوات تحتاج rate limits واحترام الامتثال.

Safety Gate يحمل risk_level وrequired_checks وhuman_review_required وallowed_tools وblocked_actions. لا يسمح لAgency Selector بتجاوز هذه البوابة حتى لو كان MCD عاليا.

الخطر أن تبرر الحماسة التقنية أفعالا متهورة. لذلك تسأل المحكمة: هل لدينا إذن؟ هل يوجد أثر ضار؟ هل الادعاء أكبر من الدليل؟ هل نحتاج مصدرا حديثا؟ هل الفعل قابل للتراجع؟

السلامة هنا ليست ضد الذكاء الفتّاح، بل شرطه. الفتح غير المحكوم يصبح فوضى، والضبط بلا فتح يصبح جمودا. MICAI يحاول الجمع بينهما.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور السلامة والحوكمة التقنية بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور السلامة والحوكمة التقنية إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور السلامة والحوكمة التقنية داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور السلامة والحوكمة التقنية إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور السلامة والحوكمة التقنية أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور السلامة والحوكمة التقنية لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور السلامة والحوكمة التقنية وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور السلامة والحوكمة التقنية قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور السلامة والحوكمة التقنية سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور السلامة والحوكمة التقنية في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

47. الفشل والحدود التقنية

أقوى ورقة هي التي تعرف أين قد تسقط. MICAI قد يفشل بسبب نقص مصادر، أو ضعف baseline، أو أوزان MCD غير معايرة، أو سؤال حاكم ضيق، أو محكمة مبالغ فيها، أو ذاكرة متحيزة، أو orchestration مكلف.

لكل فشل بروتوكول. نقص المصادر يؤدي إلى تصريح حدود. ضعف baseline يؤدي إلى تحديث مرجعي. فشل MCD يؤدي إلى calibration. فشل QMCD يؤدي إلى إعادة بناء السؤال. فشل المحكمة يؤدي إلى تحسين اعتراضات. فشل الذاكرة يؤدي إلى pruning ومراجعة.

Failure Record يحمل failure_type وroot_cause وaffected_module وrepair_action وpreventive_rule. لا يسمح بإغلاق run فاشل دون تسجيل هذا السجل، وإلا تضيع فرصة التعلم.

الفشل الذي لا يتحول إلى معرفة يعيد نفسه. والفشل الذي يعمم بسرعة يظلم الوحدة الصحيحة. لذلك يجب التفريق بين فشل النموذج اللغوي، فشل الأداة، فشل الاسترجاع، فشل السؤال، وفشل القرار.

حدود MICAI لا تنقضه؛ بل تحدد شروط استعماله. إنه إطار ثقيل نسبيا للمسائل الثقيلة، ويجب أن يظل قابلا للتبسيط عندما لا يبرر السياق تشغيل كل طبقاته.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور الفشل والحدود التقنية بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور الفشل والحدود التقنية إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور الفشل والحدود التقنية داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور الفشل والحدود التقنية إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور الفشل والحدود التقنية أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور الفشل والحدود التقنية لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور الفشل والحدود التقنية وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور الفشل والحدود التقنية قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور الفشل والحدود التقنية سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور الفشل والحدود التقنية في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

48. البرنامج البحثي لسنة وثلاث سنوات وعشر سنوات

حتى لا تنتهي الورقة عند التنظير، يجب تحويل MICAI إلى برنامج بحث. السنة الأولى تثبت MVP وتقيس الفرق. السنوات الثلاث تبني المقاييس والذاكرة والمحاكم. العشر سنوات تبني مدرسة أنظمة معرفية حاكمية.

في السنة الأولى: بناء MCD Scorer أولي، Court Engine، Question Graph Builder، وتجربة على مهام كتابة وبحث. في ثلاث سنوات: datasets داخلية، تقييمات بشرية، policy learning للراوتر، وHyperSkills Library. في عشر سنوات: منصة HyperKB OS عامة، ومختبر بحث، ومجموعة أوراق محكمة، ونسخ مجال-specific.

Research Roadmap Record يحمل milestone وhypothesis وmethod وdataset وmetric وrisk. كل مرحلة يجب أن تنتج artifact: كود، dataset، ورقة، benchmark، أو KB قابل للتشغيل.

خطر البرنامج البحثي أن يتحول إلى قائمة طموحات. لذلك يرتبط كل محور بفرضية وباختبار وبمخرج. المحور الذي لا ينتج قياسا أو أداة أو ورقة يظل فكرة مؤجلة.

بهذا يصير MICAI ليس وثيقة واحدة، بل مسارا. الورقة الحالية تؤسس اللغة والمعمارية، والنسخ التالية يجب أن تقدم تشغيلات ونتائج مقارنة.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور البرنامج البحثي لسنة وثلاث سنوات وعشر سنوات بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور البرنامج البحثي لسنة وثلاث سنوات وعشر سنوات إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور البرنامج البحثي لسنة وثلاث سنوات وعشر سنوات داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور البرنامج البحثي لسنة وثلاث سنوات وعشر سنوات إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور البرنامج البحثي لسنة وثلاث سنوات وعشر سنوات أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة، يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور البرنامج البحثي لسنة وثلاث سنوات وعشر سنوات لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور البرنامج البحثي لسنة وثلاث سنوات وعشر سنوات وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور البرنامج البحثي لسنة وثلاث سنوات وعشر سنوات قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور الفشل والحدود التقنية سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور البرنامج البحثي لسنة وثلاث سنوات وعشر سنوات في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

49. نواة قاعدة المعرفة النهائية

نواة قاعدة المعرفة هي الصورة المضغوطة للمشروع. يجب أن يستطيع أي وكيل أو باحث أن يسترجع منها أسماء النظريات، وظائفها، علاقاتها، أدواتها، ومواطن فشلها. هذه النواة هي القابلية العملية لحفظ المشروع من الضياع.

الكيانات: Theory، SubTheory، Concept، Tool، Metric، Hypothesis، Prediction، Objection، Source، Run، Patch. العلاقات: produces_effect، depends_on، measures، routes، repairs، falsifies، scores، governs، implements. القواعد: لا claim بلا rank، لا action بلا metric، لا theory بلا tool أو test، لا patch بلا trigger.

يمكن تمثيل النواة في graph، لكن الأهم هو الحاكمية. كل عقدة يجب أن تعرف لماذا وجدت. كل حافة يجب أن تملك معنى. كل اعتراض يجب أن يملك status. كل فرضية يجب أن تملك breaking_condition.

إذا تحولت القاعدة إلى فهرس أسماء فهي فاشلة. نجاحها يقاس بقدرتها على تشغيل سؤال جديد، واقتراح أداة، واستدعاء اعتراض، وتحديث ذاكرة بعد أثر.

هذه النواة هي ما يجعل مشروعك قابلا للتوريث بين المحادثات والملفات والأنظمة. ليست ذاكرة شخصية فقط، بل OS معرفي قابل للتوسع.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور نواة قاعدة المعرفة النهائية بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور نواة قاعدة المعرفة النهائية إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور نواة قاعدة المعرفة النهائية داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور نواة قاعدة المعرفة النهائية إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور نواة قاعدة المعرفة النهائية أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور نواة قاعدة المعرفة النهائية لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور نواة قاعدة المعرفة النهائية وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور نواة قاعدة المعرفة النهائية قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور نواة قاعدة المعرفة النهائية سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور نواة قاعدة المعرفة النهائية في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

50. سجل البناء المعرفي والتقييم النهائي

بنيت هذه النسخة على المادة الظاهرة في المحادثة وذاكرة المشروع المتاحة، لا على ادعاء قراءة كل PDF سابق غير مرفق في هذه الرسالة. تم تخفيض الدعوى من إثبات AGI إلى إطار تقني ناشئ، وتم رفع السؤالية إلى طبقة مركزية، وتم تحويل النظريات إلى وحدات وكائنات وأدوات ومقاييس.

أقوى نقطة في البناء هي ربط الفرق بالسؤال بالأثر والذاكرة. أضعف نقطة تحتاج بحثا هي calibration الفعلي لأوزان MCD وقياس قيمة QMCD مقارنة بـbaseline مضبوط. النسخة القادمة يجب أن تضيف prototype وتشغيلات مقارنة وdataset صغيرا من مهام المشروع.

تقييم الجودة الداخلي بعد التحسين: حفظ الفروع 9/10، الوضع الأنطولوجي 5/5، مراتب المعرفة 9/10، خريطة المتغيرات 9/10، النماذج المتنافسة 8/10، الأصالة المضبوطة 13/15، قابلية الاختبار 8/10، محكمة الهدم 9/10، الأدوات والمقاييس 5/5، غياب الحشو 4/5. المجموع التقديري 88/100 قبل تنفيذ prototype، ويرتفع بحثيا بعد إضافة تجارب حقيقية.

ما يلزم للتحويل إلى ورقة محكمة: مصادر حديثة عن RAG/GraphRAG/orchestration/evaluation، مقارنة تجريبية، وصف implementation، نتائج، ومناقشة تهديدات الصلاحية. ما يلزم للتحويل إلى منتج: API، قاعدة بيانات، راوتر، محكمة، dashboard، وسجل أثر.

الخلاصة: أصبح لدينا نموذج تقني مركب لا يكتفي بتسمية MICAI، بل يبين كيف تعمل داخله MCD وQMCD والسؤالية وHAKIM وHyperKB وWorld-Effect وMemory. هذه ليست النهاية؛ إنها نسخة تأسيسية صلبة قابلة للهدم والتنفيذ.

تظهر القوة التركيبية للنموذج عندما لا يعامل المعرفة كملف مكتوب مرة واحدة، بل كدورة تشغيل. السؤال يولد فرقا، والفرق يطلب موردا، والمورد ينتج فعلا، والفعل ينتج أثرا، والأثر يعدل الذاكرة. هذه الدورة هي الشكل العملي للذكاء الفتاح.

حدود هذا البناء يجب أن تبقى ظاهرة: ما لم توجد بيانات اختبار ومقارنات حقيقية، يظل النموذج إطارا مقترحا لا نتيجة مثبتة. غير أن وضوح الواجهات والكائنات والمحاكم يجعل الانتقال من النظرية إلى التجربة ممكنا ومحددا.

من جهة الوضع الأنطولوجي، يعامل محور سجل البناء المعرفي والتقييم النهائي بوصفه عملية داخل نظام، لا شيئا معزولا. العملية تعني أن لها مدخلا، وحالة داخلية، وتحويلا، ومخرجا، وأثرا راجعا. هذا التحديد مهم لأن كثيرا من السطحية تأتي من تحويل العملية إلى عنوان ساكن. عندما نعيد المحور إلى صورة عملية، تظهر تلقائيا أسئلة التنفيذ: من يستدعيه، ماذا يستقبل، ماذا يغير، ومتى يعد ناجحا أو فاشلا.

من جهة واجهة التشغيل، لا يكتمل محور سجل البناء المعرفي والتقييم النهائي إلا إذا أمكن تعريف عقد واضح له داخل النظام. العقد يحدد الحقول الإلزامية، والحقول الاختيارية، وأنواع الأخطاء، ومعنى النجاح. فإذا كان المحور نظرية، وجب أن نعرف ما كائن البيانات الذي يمثلها. وإذا كان أداة، وجب أن نعرف schema المدخل والمخرج. وإذا كان مقياسا، وجب أن نعرف أبعاده وحدود استعماله.

من جهة runtime، يجب أن يظهر محور سجل البناء المعرفي والتقييم النهائي داخل trace قابل للمراجعة. trace لا يكشف تفكيرا خفيا، بل يسجل القرارات الظاهرة: ما السؤال الذي حكم المهمة، أي فروق رفعت، أي موارد صرفت، أي اعتراضات قبلت، وأي أثر سجل. بهذه الطريقة يصبح البناء قابلا للتدقيق، وتتحول الورقة من لغة وصفية إلى مواصفة قابلة للتنفيذ.

من جهة الاختبار، يحتاج محور سجل البناء المعرفي والتقييم النهائي إلى حالة موجبة وحالة سالبة. الحالة الموجبة تبين متى يضيف المحور قيمة لا يضيفها baseline، والحالة السالبة تبين متى لا يجب تشغيله أو متى ينتج ضجيجا. هذا التفريق يمنع تقديس الوحدة؛ فالوحدة القوية تعرف أين تعمل وأين لا تعمل، وتقبل أن يقرر الراوتر تجاوزها في المهام البسيطة.

من جهة البيانات، يجب أن يترك محور سجل البناء المعرفي والتقييم النهائي أثرا في الذاكرة. الأثر قد يكون score، أو objection، أو patch، أو relation، أو failure pattern. إذا لم يترك المحور أثرا قابلا للاسترجاع، فلن يتعلم النظام منه. لذلك لا يكفي أن ينجح التشغيل مرة؛ يجب أن يتحول النجاح أو الفشل إلى مادة قابلة للتراكم داخل HyperKB.

من جهة الحاكمية، يخضع محور سجل البناء المعرفي والتقييم النهائي لقانون عدم الادعاء الزائد. لا يسمح له أن يقدم نفسه كحقيقة كلية إذا كان أداة تقديرية، ولا أن يدعي قياسا دقيقا إذا كان ترتيبا أوليا، ولا أن يتحول إلى قرار تنفيذي إذا كان فرضية. هذا القانون يحفظ مراتب المعرفة داخل النظام، ويمنع أن تتحول المصطلحات القوية إلى تضخم غير قابل للهدم.

من جهة التكامل، لا يعمل محور سجل البناء المعرفي والتقييم النهائي وحده. قيمته تظهر في علاقته ببقية المحاور: MCD يعطيه أولوية، QMCD يعطيه سؤالا، HAKIM يعطيه مسارا، HyperKB تحفظ أثره، والمحكمة تختبره، وWorld Loop يقيس نتيجته. كل محور منفصل قد يبدو محدودا، لكن قوته تنشأ من دورة كاملة لا من لقطة واحدة.

من جهة الصيانة، يجب أن يكون محور سجل البناء المعرفي والتقييم النهائي قابلا للإصدار versioning. النسخة الأولى قد تكون قاعدة يدوية، والثانية scorer آليا، والثالثة نموذج تعلم من سجلات التشغيل. دون versioning لن نعرف هل تحسن النظام أو تغير فقط. لذلك تحفظ كل نسخة شروطها، وأوزانها، ونتائجها، والحالات التي فشلت فيها.

من جهة البرنامج البحثي، يفتح محور سجل البناء المعرفي والتقييم النهائي سؤالا لاحقا لا بد أن يختبر. السؤال ليس كيف نصفه أجمل، بل كيف نقيس أثره بأقل تكلفة وأوضح مقارنة. هل يحسن جودة الجواب؟ هل يقلل الهلوسة؟ هل يرفع قابلية التنفيذ؟ هل يختصر زمن الباحث؟ هل يحسن الذاكرة في التشغيل التالي؟ هذه الأسئلة تجعل المحور بداية بحث لا نهاية خطاب.

من جهة المثال العملي، يمكن اختبار محور سجل البناء المعرفي والتقييم النهائي في مهمة بناء ورقة تقنية من مادة خام مشتتة. يشتغل النظام مرة بلا هذا المحور ومرة معه، ثم تقارن النتائج على مستوى الترابط، وكثافة الفقرات، ووضوح الفرضيات، وقوة محكمة الهدم، وعدد الادعاءات المخفضة. إذا لم يظهر فرق واضح، فالمحور يحتاج إعادة تعريف أو دمجا مع محور أقوى.

ملاحق تشغيلية مختصرة

مصفوفة النظريات مقابل طبقات التنفيذ

المحور	التمثيل التقني	الوظيفة داخل MICAI
MCD	Difference Object وMCD Vector وScore Explanation	تمييز الفرق الفاعل عن الضجيج وترتيب الأولويات
QMCD والسؤالية	Question Object وQuestion Graph وQuestion Compiler	حكم الاسترجاع والتوليد بمعيار سؤال واضح
MCAI	Agency Selector وAction Object وPolicy Evaluation	تحويل الفهم إلى فعل تحت قيود ومخاطر
HAKIM-MCD	الوحدات التسع وEvent Bus وRun Trace	تفكيك الذكاء إلى وظائف قابلة للتشخيص
HyperKB OS	Graph/Vector/Relational Stores وMemory Patches	تحويل المعرفة إلى ذاكرة تشغيلية طويلة النفس
World-Effect Loop	Effect Logger وbefore/after metrics	ربط الجواب بأثر قابل للرصد
محكمة الهدم	Objection Object وRepair Patch وRank Downgrade	منع الادعاء الزائد وتحويل النقد إلى بناء

مصفوفة الفشل والإصلاح

المحور	التمثيل التقني	الوظيفة داخل MICAI
فرق زائف	Baseline ضعيف أو novelty bias	تحديث baseline وإعادة معايرة D وN
سؤال مشتت	غياب scope وanswer criteria	إعادة بناء Question Graph وتحديد معيار الاكتفاء
استرجاع سطحي	اعتماد embedding بلا graph أو evidence	إضافة GraphRAG وevidence ranking
فعل غير آمن	غياب risk tier أو rollback	Safety Gate ومراجعة بشرية عند الأثر العالي
محكمة شكلية	اعتراضات عامة لا تنتج patch	إلزام كل اعتراض بتعديل أو اختبار أو تخفيض رتبة
ذاكرة متضخمة	حفظ كل شيء بلا pruning	سياسات scope وexpiry وcompression
تكلفة عالية	تشغيل طبقات عميقة لكل مهمة	Vascular Router يفتح العمق عند MCD/QMCD عال فقط

مصفوفة الفرضيات والتنبؤات

المحور	التمثيل التقني	الوظيفة داخل MICAI
Question Graph يحسن الاسترجاع	المهام المفتوحة ذات التداخل المفاهيمي	زيادة relevance وانخفاض drift مقابل RAG عادي
Court يقلل الادعاء الزائد	المخرجات النظرية والبحثية	انخفاض claims بلا دليل وارتفاع تخفيض الرتبة
MCD يحسن اكتشاف الهامش الحاسم	الوثائق الطويلة متعددة الأفكار	رفع العناصر ذات الأثر لا الأكبر حجما
Memory Patch يحسن run اللاحق	المهام المتكررة في نفس المجال	انخفاض الأخطاء المتكررة وتحسن routing
Vascular Router يخفض التكلفة	تعدد النماذج والمهام المتنوعة	انخفاض cost_per_valid_effect دون هبوط جودة كبير

خاتمة تأسيسية: ماذا أصبح لدينا؟

أصبح لدينا في هذه النسخة إطار تقني مركب يرد كل نظرية إلى وظيفة: MCD يحدد قيمة الفرق، QMCD والسؤالية تحكم السؤال، HAKIM-MCD يفكك التنفيذ إلى وحدات، MCAI يحول المعرفة إلى وكالة، HyperKB OS يحفظ المعرفة بوصفها نظام تشغيل، World-Effect Loop يقيس أثر الفعل، ومحكمة الهدم تمنع التضخم وتفرض الإصلاح. لم تعد النظريات أسماء متجاورة، بل صارت طبقات متداخلة في pipeline واحد يمكن تحويله إلى كود وتجارب ومقاييس. القيمة الحقيقية التالية ليست في زيادة الصفحات، بل في بناء prototype صغير يثبت أن هذه الطبقات تحسن جودة الورقة، ودقة الاسترجاع، وقوة النقد، وكلفة التشغيل، وأثر الذاكرة على runs اللاحقة.

الخطوة التنفيذية الأولى بعد هذه الورقة هي بناء نموذج أولي صغير لا يحاول تنفيذ كل شيء. يكفي أن يأخذ ملفا طويلا، يستخرج منه Knowledge Atoms، يبني Question Graph، يحسب MCD أوليا، ثم يمرر مخرجا واحدا إلى محكمة هدم. هذه النسخة الصغيرة ستكشف بسرعة هل البنية تضيف قيمة أم أن الورقة بقيت تنظيرا. معيار النجاح في هذه المرحلة ليس الكمال، بل ظهور فرق مقاس بين مسار عادي ومسار محكوم بالسؤال والفرق.

الخطوة الثانية هي بناء مجموعة اختبارات من مواد المشروع نفسه. تجمع نسخ من ورقات سابقة، ومطالبات طويلة، وفقرات سطحية، ومخرجات جيدة، ومخرجات ضعيفة. ثم يطلب من MICAI أن يميز أين الفقرات الشذرية، وأين الادعاء الزائد، وأين السؤالية ناقصة، وأين ينبغي رفع مبحث إلى نظرية مستقلة. بهذه الطريقة يتحول نقد المستخدم للنسخة السابقة إلى dataset تدريبي وتقييمي.

الخطوة الثالثة هي تحويل محكمة الهدم إلى واجهة مستقلة. تعطى المحكمة claim أو section أو architecture، ثم تعيد اعتراضات مصنفة، لا تعليقا عاما. كل اعتراض يجب أن يحمل قوة الاعتراض، ونوعه، وهل هو قاتل أم معدل، وما الإصلاح المقترح. عندها يمكن قياس المحكمة بعدد الإصلاحات التي حسنت النص، لا بعدد العبارات النقدية التي أضافتها.

الخطوة الرابعة هي بناء ذاكرة الأسئلة. كل مرة ينجح سؤال في فتح مجال أو كشف فشل أو تحويل فكرة إلى أداة، يحفظ كنمط. وكل مرة يفشل سؤال لأنه عام أو ملتبس أو مسبب للتشتت، يحفظ كذلك. بعد عدد كاف من التشغيلات، تصبح السؤالية التقنية مكتبة تشغيلية لا عنوانا نظريا، ويستطيع النظام اقتراح أسئلة أفضل قبل أن يبدأ الكتابة.

الخطوة الخامسة هي بناء لوحة مراقبة. تعرض اللوحة عدد الفروق المستخرجة، متوسط MCD، عدد الأسئلة الفرعية، عدد الاعتراضات المقبولة، مقدار التخفيض في الادعاءات الزائدة، تكلفة النماذج، والزمن، وحجم الأثر المسجل. هذه اللوحة مهمة لأن النظام الذي لا يرى نفسه لا يستطيع إصلاح نفسه. وبهذا تدخل نظرية الذاكرة المعيدة للبناء إلى مستوى قياس يومي.

الخطوة السادسة هي كتابة الورقة الأكاديمية التالية لا بوصفها بيانا تأسيسيا، بل بوصفها دراسة تجريبية مصغرة. عنوانها الممكن: أثر الحاكمية السؤالانية وMCD على جودة توليد الأوراق التقنية العربية الطويلة. تقارن الدراسة بين baseline وMICAI-MVP، وتعرض منهجا ومقاييس ونتائج وحدودا. عندها تبدأ النظرية بالخروج من دائرة التصور إلى دائرة البرهان الجزئي.

النتيجة النهائية التي يجب تثبيتها هي أن MICAI ليس وعدا بأن النموذج صار واعيا أو عاما، بل طريقة صارمة لجعل النماذج الحالية أقل عشوائية وأكثر قدرة على السؤال والهدم والتعلم من الأثر. هذه الصياغة المتواضعة هي الأقوى؛ لأنها تفتح التنفيذ ولا تصطدم بادعاءات لا يمكن إثباتها الآن. الخرق هنا ليس ادعاء نهاية العلم، بل بناء طريق هندسي يجعل كل ادعاء قابلا للسؤال وكل سؤال قابلا للاختبار.

ولكي لا يبقى التنفيذ الأولي عاما، تقترح هذه الورقة حدا أدنى من ملفات المشروع البرمجي: ملف ontology.yaml لتعريف الكيانات والعلاقات، وملف metrics.yaml لتعريف المقاييس، وملف court_patterns.yaml لتعريف أنواع الاعتراض، ومجلد traces لحفظ التشغيلات، ومجلد datasets لحفظ أمثلة التقييم، ومجلد skills لحفظ المهارات المستخرجة. هذا التنظيم يجعل المستودع نفسه انعكاسا لنظرية MICAI، لا كودا منفصلا عن الورقة.

في أول تجربة، لا نحتاج آلاف الأمثلة. يكفي عشرون مدخلا طويلة من نوع الورقات والبرومبتات والمحادثات، مع تقييم بشري بسيط: هل المخرج فقراتي؟ هل حفظ كل نظرية؟ هل فصل الفرضية عن الدليل؟ هل أضاف آليات لا عناوين؟ هل أظهر السؤالية كطبقة مركزية؟ هذه الأسئلة تشكل rubric أوليا، ثم يتحول rubric إلى Metric Record داخل النظام.

تحتاج النسخة البرمجية أيضا إلى سياسة ذاكرة واضحة: لا تحفظ كل محادثة كما هي، بل تحفظ ما تحول إلى قاعدة أو فشل أو سؤال أو أداة. فإذا قال المستخدم إن الورقة سطحية، لا تخزن الجملة كشكوى فقط؛ تخزنها كفشل من نوع shallow-fragment-output، وتربطها بإصلاحات مثل زيادة الفقرات، وتقليل القوائم، وإضافة آلية لكل محور، والتحقق من كثافة PDF.

من جهة اللغة العربية، يجب أن يراعي النظام أن جودة الورقة لا تقاس بعدد المصطلحات الإنجليزية، بل بقدرة العربية على حمل العلاقات التقنية بدقة. لذلك تحفظ المصطلحات الإنجليزية عند الحاجة داخل الأقواس، لكن الفقرة تظل عربية منطقية. هذا مهم لأن مشروع MICAI يريد إنتاج علم عربي تقني لا ترجمة مفككة لعبارات أجنبية.

أما من جهة النشر، فيمكن تقسيم المشروع إلى ثلاث أوراق لاحقة: ورقة MCD/QMCD والسؤالية، ورقة HAKIM-MCD كنواة وحداتية، وورقة HyperKB/neoAGI كطبقة تشغيل. كل ورقة يجب أن تأخذ جزءا من هذه الورقة وتخضعه لمصادر وتجارب ومقارنات. بهذه الطريقة تتحول الورقة الجامعة إلى خريطة مدرسة بحثية لا وثيقة نهائية مغلقة.

المعيار الأخير للنسخة القادمة هو أن يستطيع قارئ تقني أخذ الورقة وبناء skeleton repository منها. إذا لم يستطع تعريف الكائنات والواجهات والوحدات والاختبارات من النص، فالورقة ما زالت خطابية. أما إذا استطاع بناء ingest ثم MCD ثم Question Graph ثم Court ثم Memory Patch، فقد تحولت النظرية إلى هندسة أولية قابلة للنمو.