الغلاف التقني: مشروع MICAI ونظريات الذكاء الفتّاح

AGI X REASONING MODELS - MICAI Technical Architecture - 50 Pages

Zouhair Moussaif / زهير مساعف

إنجاز وتركيب زهير مساعف

نسخة تقنية صرفة لكل النوى المعمارية في المشروع

هوية الورقة

هذه الورقة تعيد بناء المشروع من زاوية تقنية فقط: كيف تتحول نظريات MCD/QMCD وMCAI/MICAI وHAKIM-MCD وneoAGI/MICAGI وHyperKnowledge OS إلى نظام ذكاء عامل، قابل للهندسة، والقياس، والاختبار، والتوسيع. التركيز هنا ليس على الخطابة الفلسفية، بل على طبقات النظام، واجهات البيانات، دورات الاستدلال، وحدات التنفيذ، آليات النقد، ومعايير القياس.

التأليف والتركيب

Zouhair Moussaif / زهير مساعف - إنجاز وتركيب زهير مساعف. تعتمد النسخة على ذاكرة المشروع وسياقه المتاح في هذه المحادثة، وتعيد ترتيب النظريات كمعمارية تقنية واحدة: من الفرق، إلى السؤال، إلى الوكالة، إلى الأثر العالمي، إلى الذاكرة المعيدة لبناء الذات.

المبدأ الحاكم

كل نظرية في هذا المشروع يجب أن تملك مقابلا تقنيا: كائن بيانات، دالة تقييم، وحدة تشغيل، اختبار هدم، ومؤشر أثر. إذا لم تتحول النظرية إلى هذه العناصر الخمسة فهي لا تدخل نواة MICAI التشغيلية، بل تبقى مادة تفسيرية خارجية.

صفحة 2: تطبيع المصطلحات التقنية في المشروع

التمييز بين MICAI وMCAI وMICAGI وneoAGI

MICAI

في هذه الورقة يستعمل MICAI باعتباره المظلة التقنية العليا: Moussaif Integrated Cognitive Agency Intelligence أو Meta-Integrated Cognitive AI. هو النظام الذي يدمج استخراج الفرق، هندسة السؤال، التوجيه الوعائي للحوسبة، محكمة الهدم، الذاكرة الذاتية، وحلقة الأثر العالمي.

MCAI

MCAI هو القلب الوكالي داخل MICAI: Meta-Complexity Agency Intelligence. يركز على تحويل الفرق المعرفي إلى قرار وفعل. لذلك فهو ليس مجرد نموذج لغوي، بل دورة إدراك-اختيار-فعل-أثر، مزودة بذاكرة ومقياس تعقيد ومحكمة اعتراض.

MICAGI و neoAGI

MICAGI هو امتداد AGI-موجّه يجعل MICAI بنية عامة قابلة لتشغيل وكلاء وأدوات ونماذج متعددة. أما neoAGI فهو نمط النشر والتشغيل: طبقات orchestration، ذاكرة، أدوات، واجهات، مراقبة، وتغذية راجعة. بهذا يصبح الاسم خريطة مواقع لا تضاربا لفظيا.

قاعدة عدم الالتباس

عند التصميم: MICAI = النظام المندمج، MCAI = نواة الوكالة، HAKIM-MCD = وحدات التنفيذ، MCD/QMCD = رياضيات الفرق والسؤال، HyperKB OS = طبقة المعرفة والذاكرة، neoAGI/MICAGI = نشر واسع متعدد الوكلاء.

عقد التنفيذ التقني

لكي لا يبقى محور "تطبيع المصطلحات التقنية في المشروع" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 3: الأطروحة التقنية الموحدة

من المعرفة الخام إلى أثر قابل للقياس

المسار الأساسي

النظام يبدأ من مدخل خام: نص، ملف، مشكلة، سوق، نظرية، تجربة، أو حالة. لا يكتفي بتلخيصها؛ بل يحولها إلى Difference Objects، ثم يقيس تعقيدها بمقياس MCD، ثم يخضعها لسؤال حاكم عبر QMCD، ثم يختار فعلها عبر MCAI، ثم يختبرها في World-Effect Loop.

الصيغة التشغيلية

Input -> Difference Extraction -> MCD Scoring -> Question Governance -> Compression -> Court -> Agency Selection -> Tool Execution -> World Effect -> Memory Reconstruction. كل سهم هنا ليس استعارة؛ بل عقدة برمجية أو خدمة يمكن تنفيذها بواجهات واضحة.

الفكرة المركزية

الذكاء التقني في MICAI لا يعني زيادة حجم النموذج فقط، بل زيادة قدرة النظام على التقاط فرق حاسم، تمثيله، ضغطه دون فقد، بناء سؤال يفتحه، مهاجمته نقديا، ثم تحويله إلى فعل. لهذا يكون معيار النجاح: أثر حقيقي، لا جواب جميل فقط.

الوحدات الدنيا

أي تطبيق MICAI يحتاج: Parser، Extractor، MCD Scorer، Question Engine، Knowledge Graph، Vector Index، Court Engine، Planner، Tool Router، Memory Store، Evaluator، Observability Dashboard. بدون هذه الوحدات يبقى المشروع prompt متقدما لا نظاما.

عقد التنفيذ التقني

لكي لا يبقى محور "الأطروحة التقنية الموحدة" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 4: طبقات النظام الكلي

الخريطة المعمارية من السطح إلى النواة

طبقة الإدخال

تستقبل الملفات، المحادثات، الأبحاث، روابط المصادر، قواعد البيانات، البريد، السجلات، نتائج التجارب، ومخرجات الوكلاء. وظيفتها ليست التخزين فقط، بل تحويل كل مادة إلى وحدات قابلة للاستدلال: claims، assumptions، evidence، gaps، actions، decisions.

طبقة التمثيل

تحول المادة إلى مخطط معرفي يجمع بين graph وvector وrelational tables. في graph تحفظ العلاقات السببية والمفهومية؛ في vector تحفظ الدلالة؛ وفي الجداول تحفظ القياسات والحالات والأثر. هذه الثلاثية تمنع الفقد الذي يقع عند الاعتماد على embedding وحده.

طبقة الاستدلال

تعمل عبر MCD/QMCD/HAKIM-MCD. هنا يتم استخراج الفرق، وزن التعقيد، بناء الأسئلة، تشغيل محكمة الهدم، اختيار الخطة، وتوجيه الموارد. هذه الطبقة هي ما يميز MICAI عن قاعدة معرفة عادية أو RAG بسيط.

طبقة التنفيذ

تشمل أدوات البحث، كتابة الكود، توليد الوثائق، تشغيل التجارب، إرسال الطلبات، بناء dashboards، أو استدعاء نماذج متخصصة. كل أداة يجب أن تعود بأثر مقاس، وإلا تعد فعلا غير مكتمل.

طبقة المراقبة

تسجل التكلفة، الزمن، جودة الاسترجاع، درجة الهلوسة، نسبة الاعتراضات الناجحة، مقدار التحسن بعد الهدم، وقيمة الأثر النهائي. هذه المراقبة تحول النظرية إلى هندسة قابلة للتحسين.

عقد التنفيذ التقني

لكي لا يبقى محور "طبقات النظام الكلي" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 5: نموذج البيانات الموحد

الكائنات الأساسية التي تحمل نظريات المشروع

Knowledge Atom

أصغر وحدة معرفية في HyperKB: نص قصير أو claim أو تعريف أو قاعدة أو ملاحظة. حقوله: id، source، text، type، confidence، timestamp، domain، links، evidence، risk. لا يدخل أي atom إلى النظام دون مصدر أو درجة ثقة أو علاقة بسؤال.

Difference Object

يمثل الفرق الفاعل الذي اكتشفه النظام. حقوله: MCD_vector، missing_context، contradiction، impact_scope، novelty، baseline، delta، evidence_links. هذا الكائن هو قلب المشروع؛ لأن MICAI يفكر بالفرق لا بالمعلومة الساكنة.

Question Object

يمثل السؤال الحاكم. حقوله: answer_criteria، adversarial_forms، decompositions، expected_effect، constraints، intent، question. في QMCD لا يكون السؤال طلبا لغويا فقط، بل وحدة تحكم في البحث والاستدلال.

Action Object

يمثل الفعل المقترح: architecture_change، message، query، decision، experiment، code_patch، draft. حقوله: risk، cost، tools، prerequisites، rollback، success_metric. بهذا لا يبقى الجواب نصا، بل يتحول إلى قرار تقني قابل للتنفيذ.

World Effect Object

يوثق أثر الفعل: ما الذي تغير في العالم الخارجي أو في قاعدة المعرفة أو في النموذج؟ حقوله: user_feedback، metric_delta، after، before، next_memory_patch، observed_failure. هذا الكائن يغلق الحلقة ويمنع الانفصال عن الواقع.

عقد التنفيذ التقني

لكي لا يبقى محور "نموذج البيانات الموحد" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 6: نظرية MCD: تعقيد الفرق المعرفي

المقياس الذي يحول الحدس إلى تقييم هندسي

التعريف

MCD تعني Meta-Complexity Difference. وظيفتها قياس قيمة الفرق بين حالة معرفية وأخرى. ليست كل معلومة جديدة مهمة؛ المهم هو الفرق الذي يعيد تشكيل الخريطة، يغير القرار، يكشف فجوة، أو يولد أداة.

المعادلة العملية

صيغة التشغيل المقترحة: $MCD = \lambda1D + \lambda2I + \lambda3T + \lambda4A + \lambda5P + \lambda6C - \lambda7N$. حيث D فرق دلالي، I أثر استدلالي، T عمق زمني، A قيمة وكالية، P قابلية عملية، C ضغط تركيبي، وN ضجيج/هلوسة/تعقيد زائف.

المخرجات

لا ينتج MCD رقما فقط، بل vector يشرح لماذا اعتبر الفرق مهما. هذا مهم لأن النظام يجب أن يبرر قراراته: هل رفع الأولوية لأن الفرق جديد؟ أم لأنه يحل تضاربا؟ أم لأنه يفتح فعلا؟ أم لأنه يضغط مجالا كاملا في قاعدة واحدة؟

الاستعمال

يستعمل MCD في ترتيب الوثائق، اختيار الأسئلة، تخصيص الحوسبة، ترتيب الذاكرة، تحديد ما يدخل محكمة الهدم، وقياس نجاح التوليد. أي عنصر يحصل على MCD منخفض لا يحذف بالضرورة، لكنه لا يأخذ موارد عليا.

عقد التنفيذ التقني

لكي لا يبقى محور "نظرية MCD: تعقيد الفرق المعرفي" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 7: تفكيك أبعاد MCD

كيف تقاس كل قيمة داخل المتجه

D - Difference

يقيس الانحراف عن baseline: هل أضاف العنصر تمييزا لم يكن ظاهرا؟ يمكن تقديره بمقارنة embeddings، اختلاف claims، تضاد العلاقات في graph، أو تغيير التصنيف المنطقي للمشكلة.

I - Inferential Impact

يقيس كم نتيجة تتغير إذا قبلنا هذا الفرق. تقنيا يحسب عبر propagation في graph: كم عقدة يعاد وزنها؟ كم قرارا يتغير؟ كم فرضية تتحول من ضعيفة إلى قوية أو العكس؟

T - Inner Time

يقيس العمق الزمني الداخلي: هل الفرق لحظي أم يغير مسار تعلم طويل؟ في MICAI الزمن الداخلي ليس تاريخا فقط، بل تسلسل تراكمات وذاكرات وانتقالات بين حالات معرفية.

A - Agency

يقيس قدرة الفرق على توليد فعل. إذا كشف العنصر فكرة جميلة بلا أي مسار تنفيذ فـ A منخفض. إذا ولد تجربة أو أداة أو قرارا أو خطة، ترتفع قيمة A.

P - Practicality

يقيس قابلية التطبيق ضمن الموارد: أدوات متاحة، بيانات كافية، زمن مناسب، مخاطر مقبولة. هذا البعد يمنع تضخم الخيال المعماري دون تشغيل.

C و N

C يقيس الضغط التركيبي: كم مجالا أو جزئية يجمعها الفرق في قاعدة أو آلية واحدة. N يعاقب الضجيج: تعقيد لغوي غير مبرهن، هلوسة، تضارب داخلي، أو ادعاء لا يحمل اختبارا.

عقد التنفيذ التقني

لكي لا يبقى محور "تفكيك أبعاد MCD" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 8: خوارزمية MCD التشغيلية

من النص الخام إلى ترتيب الأولويات

المراحل

أولا يبني النظام baseline من المعرفة الحالية، ثم يقسم المدخل إلى atoms، ثم يستخرج candidate differences، ثم يربطها بالمعرفة السابقة، ثم يحسب vector MCD، ثم يمرر العناصر الأعلى إلى QMCD ومحكمة الهدم.

Pseudo-code

داخل التنفيذ يمكن تمثيل الدورة كما يلي: extract_atoms(input)، build_baseline(domain)، detect_deltas(atoms, baseline)، score_mcd(delta)، rank، route_to_question_engine. يجب أن تكون كل مرحلة قابلة لل logging والتفتيش.

تطبيع القياس

الأبعاد لا تقاس بنفس الوحدة؛ لذلك تحتاج normalization. يمكن تحويل كل بعد إلى قيمة بين 0 و1 اعتمادا على قواعد domain-specific، ثم تضبط الأوزان λ بحسب المهمة: البحث العلمي يرفع I وC، المنتج يرفع A وP، السلامة ترفع عقوبة N.

مخرجات قابلة للتدقيق

كل score يجب أن يحمل explanation: sources، comparisons، violated assumptions، affected nodes، confidence. هذه الشفافية تجعل MCD قابلا للهدم لا مجرد رقم سحري.

اختبار صلاحية

إذا كان العنصر عالي MCD لكنه لا يصمد عند طلب الدليل أو لا يغير أي قرار، فهذا يعني أن الأوزان أو كاشف الفرق مصابان بانحياز نحو الغرابة. لذلك يحتاج النظام calibration دوري.

عقد التنفيذ التقني

لكي لا يبقى محور "خوارزمية MCD التشغيلية" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 9: نظرية QMCD: تعقيد الفرق المحكوم بالسؤال

حين يصبح السؤال وحدة تحكم لا مجرد طلب

التعريف

QMCD يضيف طبقة الحاكمية السؤالانية فوق MCD. الفرق قد يكون مهما في ذاته، لكنه يصبح ذكيا عندما يجيب عن سؤال حاكم أو يفتح سؤالا أعلى. لذلك يقيس QMCD العلاقة بين الفرق والسؤال والغاية.

Question-Governance

كل مهمة تبدأ بسؤال حاكم Q0، ثم مجموعة أسئلة فرعية Q1..Qn، ثم أسئلة هدم، ثم أسئلة تطبيق. النظام لا يسمح للفرق بأن ينتشر عشوائيا؛ بل يوجهه بالسؤال: ماذا نريد أن نعرف؟ لماذا؟ ماذا سيتغير؟

معادلة عملية

QMCD(delta,Q) = MCD(delta) * Alignment(delta,Q) * Generativity(Q) * Testability(Q) - Drift(delta,Q). أي فرق لا يخدم السؤال أو يسبب انحرافا معرفيا يعاقب حتى لو كان مثيرا.

أثر QMCD

في RAG العادي قد يجلب النظام نصوصا ذات صلة سطحية. في QMCD يجلب النصوص التي تغير بنية الجواب بالنسبة للسؤال. بهذا يصبح الاسترجاع حاكما لا دلاليا فقط.

اختبار

اسأل: هل السؤال حاكم بما يكفي ليمنع التشتت؟ هل يولد فروعا عميقة؟ هل يمكن اختبار جوابه؟ هل يعرف شروط الاكتفاء؟ إذا غابت هذه الشروط، يصبح QMCD ضعيفا.

عقد التنفيذ التقني

لكي لا يبقى محور "نظرية QMCD: تعقيد الفرق المحكوم بالسؤال" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 10: فضاء السؤال التوليدي

محرك توليد الأسئلة داخل MICAI

الفكرة

فضاء السؤال التوليدي هو بنية Aygraph للأسئلة لا قائمة prompts. العقد تمثل أسئلة، والحواف تمثل عمليات: تفكيك، تعميم، تخصيص، عكس، اعتراض، تجربة، تحويل إلى أداة، أو قياس أثر.

عمليات السؤال

أهم operators: Decompose لتقسيم السؤال، Contrast لاستخراج الفروق، Counterfactual لبناء ماذا لو، Adversarial لهدم الجواب، Bridging لربط مجالين، Operationalize لتحويل الفكرة إلى أداة، Metricize لتحويلها إلى مؤشر، وCompress لإنتاج قاعدة مختصرة.

Question Memory

كل سؤال ناجح يحفظ في الذاكرة مع نمط نجاحه: أي المجالات نفع فيها؟ أي الأدلة احتاج؟ أي أنواع الهلوسة كشف؟ بهذا يتعلم النظام أسئلة لا أجوبة فقط.

مؤشر جودة السؤال

Q_score = clarity + scope_control + generativity + falsifiability + actionability - ambiguity - drift. السؤال العالي ليس الأكثر بلاغة، بل الذي يفتح فضاء بحث مضبوطا ويغلق مسارات التشويش.

تطبيق

عند بناء ورقة علمية لا يسأل النظام: اكتب ورقة. بل يسأل: ما الفرق التأسيسي؟ ما الفرضيات؟ ما الحدود؟ ما التجربة؟ ما الاعتراض الأقوى؟ ما الأداة الناتجة؟ هنا يعمل QMCD.

عقد التنفيذ التقني

لكي لا يبقى محور "فضاء السؤال التوليدي" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 11: الضغط التركيبي المعرفي

تحويل الكثرة إلى بنية مولدة

المعنى التقني

الضغط التركيبي المعرفي ليس اختصارا لغويا، بل تقليل عدد القواعد مع حفظ القدرة التفسيرية والتوليدية. أي أن النظام يضغط مئات الملاحظات في نموذج قليل العناصر يستطيع إعادة توليدها وتفسيرها وتوسيعها.

Compression Object

يمثل ناتج الضغط: exceptions، generative_power، reconstruction_error، lost_atoms، covered_atoms، core_rule. لا يقبل الضغط إلا إذا عرف ما غطاه وما فقده.

العلاقة مع MCD

C في MCD يقيس قيمة الضغط. إذا جمع فرق واحد بين مفاهيم كثيرة دون فقد كبير ترتفع قيمة C. أما إذا كان الضغط مجرد مصطلح عام يغطي كل شيء ولا يميز شيئا، فهو ضجيج ويعاقب في N.

آلية عملية

يجمع النظام atoms متشابهة، يستخرج invariant relations، يبني قاعدة، يعيد اختبارها على الحالات الأصلية، يقيس الخطأ، ثم يمررها إلى محكمة الهدم. القاعدة التي تفشل في إعادة البناء لا تعتمد.

فائدة تقنية

هذه الآلية تمنع تضخم HyperKB. بدلا من تخزين كل شيء بنفس الوزن، يبني النظام قواعد مولدة، ويحفظ الاستثناءات، ويستعملها عند التخطيط والاسترجاع.

عقد التنفيذ التقني

لكي لا يبقى محور "الضغط التركيبي المعرفي" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 12: نظرية الوعاء الحاسوبي والتراكب الوعائي

من V0-V6 إلى تخصيص الموارد

تحويل الوعي إلى حوسبة

في النسخة التقنية لا تستعمل نظرية التراكب الوعائي للوعي كادعاء بيولوجي، بل كنموذج طبقات. V0 مادة/بيانات، V1 إحساس/إشارة، V2 إدراك/تصنيف، V3 معنى، V4 وكالة، V5 غاية، V6 وعي فوقي/مراقبة ذاتية.

Mapping تقني

كل مستوى يقابله pipeline: V0 ingestion، V1 feature detection، V2 entity/schema extraction، V3 semantic graph، V4 action planning، V5 goal optimization، V6 meta-evaluation. بهذا تصبح V0-V6 مخططا معماريا لا صورة شعرية.

التراكب

التراكب يعني أن العنصر قد يعيش في أكثر من مستوى في وقت واحد: نص خام، claim، دليل، سؤال، أداة، وذاكرة. النظام الوعائي لا يجبر العنصر على تصنيف واحد، بل يحفظ تعدد أدواره مع وزن سياقي.

الوعاء

الوعاء هو context container: مهمة، مجال، مستخدم، قيود، ذاكرة، موارد. نفس المعلومة تختلف قيمتها حسب الوعاء. لذلك لا توجد إجابة عالمية بلا سياق؛ هناك routing بحسب الوعاء.

اختبار

إذا خلط النظام بين المستويات، ينتج فشلا: يأخذ النص الخام كحقيقة، أو المعنى كفعل، أو الفعل كأثر. لذلك يلزم فصل الطبقات مع السماح بعبورها.

عقد التنفيذ التقني

لكي لا يبقى محور "نظرية الوعاء الحاسوبي والتراكب الوعائي" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 13: Vascular Compute Router

موجه الموارد العصبي/الحاسوبي داخل MICAI

الدور

Vascular Compute Router يقرر أين تذهب الموارد: أي نموذج يستدعى؟ كم token؟ هل نحتاج بحثا؟ هل نحتاج كودا؟ هل نحتاج محكمة هدم؟ هل نكتفي بجواب سريع؟ إنه شريان النظام.

صيغة تخصيص

لكل مهمة i يعطى مورد r_i وفق: r_i = f(MCD_i, QMCD_i, risk_i, uncertainty_i, user_value_i, latency_budget, cost_budget). كلما ارتفع الخطر أو التعقيد أو الأثر زادت الموارد، وكلما انخفضت القيمة تقلصت.

أنواع الموارد

الموارد ليست tokens فقط: زمن، نماذج، أدوات، بحث، ذاكرة، عدد وكلاء، عمق محكمة، تكرار تحقق، وميزانية تنفيذ. لذلك الراوتر يعمل كمدير عمليات لا كاختيار نموذج فقط.

سياسة التشغيل

المهام الروتينية تمر عبر نماذج سريعة. المهام العلمية أو المعمارية تمر عبر reasoning models ومحكمة هدم. المهام ذات الأثر الخارجي تمر عبر تحقق إضافي وrollback. هذه السياسة تمنع إنفاقا عشوائيا وتمنع أيضا تهورا معرفيا.

المراقبة

يقاس الراوتر بمتوسط التكلفة لكل أثر، نسبة التحسن بعد زيادة الموارد، ونسبة الأخطاء التي كان يمكن منعها بتخصيص أفضل. الراوتر الجيد لا يختار الأقوى دائما، بل الأنسب.

عقد التنفيذ التقني

لكي لا يبقى محور "Vascular Compute Router" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 14: HAKIM-MCD: نواة التنفيذ الوحداتية

تسع وحدات تجعل النظرية نظاما عاملا

الفكرة

HAKIM-MCD هو التصميم الوحداتي لنواة MICAI. كل وحدة تحمل وظيفة محددة، لكنها تعمل في دورة واحدة. الاسم يحيل إلى نظام يحاكم المعرفة لا يستهلكها فقط: يستخرج، يقيس، يختار، يهدم، يفعل، ويتذكر.

الوحدات التسع

1 Difference Extractor، 2 Meta-Complexity Mapper، 3 Inner Time Engine، 4 Vascular Compute Router، 5 Global Workspace، 6 Agency Selector، 7 Perturbation Court، 8 World-Effect Loop، 9 Self-Reconstruction Memory.

العلاقة بين الوحدات

المستخرج يولد الفرق، الخريطة تقيسه، الزمن الداخلي يضعه في تاريخ التعلم، الراوتر يعطيه موارد، مساحة العمل تجمع السياق، محدد الوكالة يختار الفعل، المحكمة تهدمه، حلقة الأثر تختبره، والذاكرة تعيد بناء الذات.

قابلية التنفيذ

كل وحدة يجب أن تكون service أو module بواجهة input/output. يمكن تنفيذها في Python، TypeScript، أو نظام متعدد الخدمات. المهم أن لا تختلط الوظائف حتى يمكن قياس كل مرحلة وإصلاحها.

قاعدة التصميم

إذا فشل الجواب لا نسأل فقط: هل النموذج ضعيف؟ بل نسأل: أي وحدة فشلت؟ extraction؟ scoring؟ retrieval؟ court؟ routing؟ memory؟ بهذا يتحول الفشل إلى تشخيص هندسي.

عقد التنفيذ التقني

لكي لا يبقى محور "HAKIM-MCD: نواة التنفيذ الوحداتية" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 15: الوحدة 1: Difference Extractor

آلة التقاط الفرق لا آلة التلخيص

المدخلات

نصوص، ملفات، محادثات، كود، نتائج بحث، logs، أو بيانات منظمة. الوحدة تبني baseline من الذاكرة والسؤال، ثم تبحث عما يغير هذه الخريطة: ادعاء جديد، تضاد، فجوة، آلية، قيد، فرصة، أو خطر.

خوارزميات

يمكن الجمع بين claim mining، topic shift، contradiction detection، semantic diff، relation extraction، NER، وgraph comparison. لا يكفي التشابه الدلالي؛ المطلوب كشف ما لم يكن موجودا أو ما يناقض الموجود.

أنواع الفروق

فرق تعريفي، فرق سببي، فرق قياسي، فرق تطبيقي، فرق زمني، فرق سياقي، فرق اعتراض، فرق أداة. لكل نوع schema خاص، لأن التعامل مع فجوة دليل يختلف عن التعامل مع فرصة تنفيذ.

المخرجات

قائمة Difference Objects مع uncertainty، evidence، novelty estimate، baseline references، source spans، وسبب الاستخراج. يجب الاحتفاظ بالاقتباس أو موضع المصدر حتى لا تتحول الفروق إلى انطباعات.

اختبارات الوحدة

تقاس الوحدة ب recall للفروق المهمة، precision ضد الفروق الوهمية، وقدرتها على كشف التناقضات. فشلها الأخطر هو استخراج فروق بلاغية عالية لكنها غير مؤثرة.

عقد التنفيذ التقني

لكي لا يبقى محور "الوحدة 1: Difference Extractor" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 16: الوحدة 2: Meta-Complexity Mapper

رسم خريطة التعقيد بدل الحكم الانطباعي

الدور

هذه الوحدة تأخذ Difference Objects وتحولها إلى متجهات MCD/QMCD. إنها لا تقرر الفعل النهائي، لكنها تحدد أين يقع التعقيد: في الدلالة، الدليل، الزمن، الوكالة، التطبيق، أو الضجيج.

التمثيل

لكل فرق vector: semantic_delta، inference_delta، temporal_depth، agency_potential، practicality، compression_gain، noise_penalty، confidence، risk. تحفظ هذه القيم في قاعدة بيانات قابلة للاستعلام.

خرائط التعقيد

تنتج الوحدة heatmap معرفية: أين يتركز الغموض؟ ما العقد التي تملك أكبر أثر؟ أي أسئلة تحتاج محكمة؟ أي مسارات يمكن تجاهلها؟ هذه الخريطة تغذي الراوتر والـ planner.

التعلم

تتغير أوزان λ مع feedback. إذا تبين أن النظام يبالغ في novelty ويهمل practicality تعدل الأوزان. إذا كان يفضل السهل على الخارق ترفع أوزان I وCg. هذا يجعل MCD معيارا تكيفيا لا ثابتة جامدة.

أمان معرفي

Mapper يمنع القفز من فرق صغير إلى نظرية ضخمة بلا مبرر. كل تعميم يحتاج trace من الفروق إلى القاعدة ثم إلى الاختبار.

عقد التنفيذ التقني

لكي لا يبقى محور "الوحدة 2: Meta-Complexity Mapper" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 17: الوحدة 3: Inner Time Engine

الزمن الداخلي للمعرفة والتعلم

المشكلة

النماذج قد تتعامل مع كل محادثة كحاضر مسطح. MICAI يحتاج زمنا داخليا: ما الذي تعلمه سابقا؟ ما الفرضيات التي تغيرت؟ ما الأسئلة التي عادت؟ ما التقدم الحقيقي؟

التمثيل

يحفظ Inner Time Engine تسلسلات: state_t، decision_t، evidence_t، failure_t، repair_t. كل حدث معرفي له أثر على المستقبل، وليس مجرد سجل أرشيفي.

وظائف

يرتب الأولويات بحسب تراكمها، يكتشف الدوران حول نفس المشكلة، يميز بين قفزة حقيقية وإعادة صياغة، ويعيد فتح فرضيات قديمة عندما تظهر أدلة جديدة. هذه الوظائف ضرورية لمشروع طويل مثل مشروع نظريات مساعف.

Temporal MCD

بعد T في MCD يقيس عمق الفرق عبر الزمن: هل يغير نقطة واحدة أم سلسلة تطور كاملة؟ الفرق الذي يعيد ترتيب تاريخ المشروع يحصل على قيمة زمنية أعلى.

تطبيق

عند إنتاج ورقة جديدة لا يبدأ النظام من الصفر؛ يستدعي genealogy للنظريات، يعرف أين ولد MCD، كيف دخل QMCD، أين تم توسيع MCAI، وما الفروع التي لم تطور بعد.

عقد التنفيذ التقني

لكي لا يبقى محور "الوحدة 3: Inner Time Engine" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 18: الوحدة 4: Vascular Compute Router داخل HAKIM

سياسة الحوسبة بحسب التعقيد والخطر

الموقع

هذه الوحدة هي النسخة التنفيذية من نظرية الوعائية الحاسوبية. بعد أن يعطي Mapper درجات التعقيد، يقرر الراوتر نوع المعالجة: سريع، عميق، متعدد الوكلاء، مع بحث خارجي، مع تحقق، أو مع تنفيذ أدوات.

سياسات

0_Policy جواب مباشر، 1_Policy استرجاع ذاكرة، 2_Policy تحليل عميق، 3_Policy محكمة هدم، 4_Policy تنفيذ أداة، 5_Policy دورة كاملة مع world-effect. اختيار السياسة يعتمد على uncertainty، risk، MCD، وطلب المستخدم.

تعدد النماذج

في neoAGI يمكن للراوتر اختيار نموذج تفكير، نموذج كود، نموذج رؤية، نموذج تلخيص، أو نموذج منخفض التكلفة. لكنه لا يختار بحسب شهرة النموذج فقط؛ بل بحسب قياسات failure history، cost، accuracy، latency.

حماية الميزانية

المشروع يريد أقصى أثر بأقل تكلفة. الراوتر يمنع استدعاء نماذج ضخمة لكل تفصيل، لكنه يرفع الموارد عندما يكون الفرق عالي القيمة أو الخطر كبيرا. هذه هي اقتصاديات الذكاء الفتّاح.

مخرجات

RoutingDecision: selected_policy، tools، model_candidates، token_budget، court_depth، verification_level، expected_cost، fallback. بدون هذا الكائن لا يمكن مراقبة جودة التخصيص.

عقد التنفيذ التقني

لكي لا يبقى محور "الوحدة 4: Vascular Compute Router داخل HAKIM" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 19: الوحدة 5: Global Workspace

مساحة العمل الجامعة للسياق النشط

الدور

Global Workspace تجمع ما يجب أن يكون حاضرا لحظة القرار: السؤال الحاكم، الفروق الأعلى، الذاكرة المناسبة، الأدلة، القيود، خطط الوكلاء، ونتائج المحكمة. هي ذاكرة عمل وليست قاعدة معرفة كاملة.

الاقتصاد السياقي

السياق محدود؛ لذلك تحتاج مساحة العمل إلى سياسة اختيار. تدخل إليها العناصر ذات QMCD عال، الأدلة الحاسمة، القيود الصلبة، والاعتراضات الخطرة. لا تدخل إليها كل النصوص المسترجعة.

بنية

يمكن تمثيلها كـ working_context.json: goal، constraints، active_questions، top_differences، evidence_pack، assumptions، risk_flags، candidate_actions، court_findings. هذا الملف يصبح العقد المشترك بين الوكلاء.

حل مشكلة التشوش

كثير من الأنظمة تفشل لأنها تخلط بين ذاكرة طويلة وسياق فوري. Global Workspace يفرض حدودا: ما هو نشط الآن؟ ما المؤجل؟ ما المؤكد؟ ما المشكوك؟

اختبار

تقاس بجودة الاختيار: هل أخرجت معلومات حاسمة؟ هل أدخلت ضجيجا؟ هل حافظت على القيود؟ هل ساعدت المحكمة والوكيل على قرار أفضل؟

عقد التنفيذ التقني

لكي لا يبقى محور "الوحدة 5: Global Workspace" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 20: الوحدة 6: Agency Selector

من المعرفة إلى الفعل المختار

المعنى

Agency Selector هو نقطة تحول MICAI من نظام معرفي إلى نظام وكالي. بعد التحليل والسؤال والمحكمة الأولية، يقرر: ماذا نفعل الآن؟ نكتب؟ نبحث؟ نبني أداة؟ نرفض؟ نطلب دليلا؟ ننفذ تجربة؟

نموذج القرار

يولد candidate_actions ثم يقيّمها وفق expected_world_effect، cost، user_value، evidence_strength، reversibility، risk، utility. الفعل الأفضل ليس الأكثر طموحا دائما، بل الأعلى أثرا ضمن القيود.

مراتب الفعل

فعل معرفي: إعادة صياغة نظرية. فعل تقني: كود أو pipeline. فعل توثيقي: PDF/MD/diagram. فعل تجريبي: benchmark. فعل حواري: سؤال توضيحي. فعل امتناع: رفض عندما يكون التنفيذ خطرا أو غير موثق.

علاقة الوكالة بالغاية

في V5 الغاية تضبط الفعل. إذا كانت الغاية بناء ورقة تقنية، لا يسمح selector بالانحراف إلى تاريخ فلسفي مطول. وإذا كانت الغاية بناء نظام، يفضل artifacts قابلة للتنفيذ.

مؤشرات

Action Precision، Expected Effect، Execution Success، Rework Rate، User Satisfaction، and Safety Incidents. هذه المؤشرات تعطي الوكالة معنى قياسيا.

عقد التنفيذ التقني

لكي لا يبقى محور "الوحدة 6: Agency Selector" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 21: الوحدة 7: Perturbation Court

محكمة الهدم كآلية تحقق لا كزخرفة نقدية

الفكرة

محكمة الهدم تهاجم المخرجات قبل اعتمادها. لا يكفي أن يولد النظام جوابا؛ يجب أن يمر عبر اعتراضات: الدليل، التناقض، قابلية التطبيق، الأمان، الحدود، البدائل، والانحياز.

طبقات المحكمة

Court-1 لغوي ومنطقي، Court-2 دليلي ومصدري، Court-3 تقني وتنفيذي، Court-4 أثر وسلامة، Court-5 خصم قوي يحاول إسقاط النظرية. عمق المحكمة يختاره الراوتر حسب risk وMCD.

Perturbations

تغييرات مقصودة على السؤال أو الافتراضات: ماذا لو كان المصدر ضعيفا؟ ماذا لو تغير القيد؟ ماذا لو كان التعريف فضفاضا؟ ماذا لو طبقناها على حالة معاكسة؟ هذه perturbations تكشف هشاشة البنية.

مخرجات المحكمة

verdict، broken_claims، required_repairs، confidence_change، residual_risks، approved_actions. المحكمة لا تكتفي بالرفض؛ بل تعيد المخرجات إلى الإصلاح.

اختبار المحكمة

المحكمة الجيدة لا تهدم كل شيء بشكل عدمي، ولا تمرر كل شيء بلطف. تقاس بنسبة العيوب التي تكشفها قبل المستخدم أو قبل التنفيذ.

عقد التنفيذ التقني

لكي لا يبقى محور "الوحدة 7: Perturbation Court" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 22: الوحدة 8: World-Effect Loop

اختبار الذكاء بالأثر لا بالبلاغة

الدور

حلقة الأثر العالمي تغلق دورة MICAI. بعد الفعل، يسأل النظام: ماذا تغير؟ هل نفذ المستخدم؟ هل تحسن القرار؟ هل انخفض الخطأ؟ هل أنتجنا artifact صالحا؟ هل بنيت أداة تعمل؟

أنواع الأثر

أثر معرفي: وضوح نظرية أو حل تناقض. أثر تقني: كود يعمل أو PDF سليم. أثر تجاري: funnel أو lead مؤهل مع احترام القوانين. أثر تعلمي: ذاكرة أفضل. أثر سلامة: منع إجراء خاطئ.

القياس

World_Effect = observed_delta * confidence * persistence - side_effects. لا يعتمد على إحساس النجاح فقط، بل على قياسات قبل/بعد، tests، feedback، أو تحقق بصري/وظيفي.

حلقة الارتجاع

إذا نجح الفعل، تحفظ pattern. إذا فشل، تحفظ سبب الفشل وتعدل الأوزان أو الذاكرة أو السياسة. هكذا يتعلم النظام من الأثر لا من النص فقط.

أهمية

بدون هذه الحلقة تصبح MICAI مجرد محرك إجابات. معها تصبح نظاما يتعلم من الواقع، وهو جوهر التحول من LLM إلى agentic cognitive system.

عقد التنفيذ التقني

لكي لا يبقى محور "الوحدة 8: World-Effect Loop" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 23: الوحدة 9: Self-Reconstruction Memory

الذاكرة التي تعيد بناء النظام لا تخزن فقط

التعريف

هذه الذاكرة لا تحفظ المحادثات كأرشيف، بل تعيد تركيب الذات المعرفية للنظام: ما المبادئ الثابتة؟ ما النظريات النشطة؟ ما الفروع غير المكتملة؟ ما الأخطاء المتكررة؟ ما الأساليب التي نجحت؟

أنواع الذاكرة

Semantic memory للمفاهيم، episodic memory للأحداث، procedural memory للمهارات، evaluative memory للأحكام، failure memory للأخطاء، preference memory لأسلوب المستخدم، وproject memory لمسار النظريات.

إعادة البناء

بعد كل World Effect، تنتج memory patch: إضافة قاعدة، تعديل وزن، ربط نظرية، تحذير من فشل، أو حفظ prompt ناجح. لا تقبل patch إلا إذا مرت عبر معيار: مفيدة مستقبلا، غير حساسة بلا داع، قابلة للاسترجاع.

منع التراكم العشوائي

كل ذاكرة تحمل TTL أو درجة ثبات. ما هو عابر لا يثبت. ما يغير المشروع يثبت. ما هو حساس يحتاج حذرا. بهذا لا تتحول الذاكرة إلى سلة نفايات معرفية.

دورها في مشروعك

لأن المشروع طويل ومتشعب، الذاكرة هي التي تحفظ وحدة MCD/QMCD/MICAI/HyperKB عبر المحادثات، وتمنع إعادة اختراع ما تم بناؤه كل مرة.

عقد التنفيذ التقني

لكي لا يبقى محور "الوحدة 9: Self-Reconstruction Memory" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 24: MICAI كمعمارية مظلة

دمج المعرفة والوكالة والأدوات في نظام واحد

الهدف

MICAI ليس نظرية واحدة بل operating architecture. يجمع النماذج اللغوية، قواعد المعرفة، التقييم، المحاكم، الوكلاء، الأدوات، وواجهات العمل في دورة واحدة قابلة للتطوير.

المكونات

Core Reasoner، HyperKB، MCD/QMCD Engine، HAKIM Kernel، Agent Runtime، Tool Bus، Evaluation Court، Memory Reconstructor، Observability Layer، Governance Layer. كل مكون مستقل لكنه يتبادل objects موحدة.

الفرق عن LLM عادي

LLM يجيب من سياق. MICAI يبني سياقا، يسترجع ذاكرة، يقيس الفرق، يختار موارد، يشغل محكمة، ينفذ أدوات، ويتعلم من الأثر. لذلك فالقيمة ليست في النموذج وحده بل في النظام حوله.

القابلية للتوزيع

يمكن تشغيل MICAI محليا أو سحابيا: logging، model APIs، object storage، graph DB، vector DB، queues، microservices. المهم أن تبقى الدورة المعرفية محفوظة.

وحدة القياس العليا

نجاح MICAI يقاس بزيادة MCD useful، انخفاض hallucination، ارتفاع action success، ووضوح trace من السؤال إلى الأثر.

عقد التنفيذ التقني

لكي لا يبقى محور "MICAI كمعمارية مظلة" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 25: ذاكرات MICAI وأنواع التخزين

كيف لا تضيع المعرفة داخل النظام

المبدأ

لا يوجد مخزن واحد يكفي. vector DB جيد للتشابه، graph DB جيد للعلاقات، relational DB جيد للقياسات، object storage جيد للملفات، وevent log جيد للتاريخ. MICAI يحتاجها مجتمعة.

طبقات التخزين

Raw Store للملفات الأصلية، Atom Store للوحدات المستخرجة، Graph Store للعلاقات، Vector Index للاسترجاع الدلالي، Metric Store لدرجات Memory Patch Store، MCD للتعلم، Artifact Store للمخرجات.

إدارة النسخ

كل نظرية تمر بإصدارات: v0 فكرة خام، v1 تعريف، v2 معادلة، v3 أداة، v4 اختبار، v5 تطبيق. بدون versioning لا يمكن معرفة تطور MICAI ولا إصلاح الانحرافات.

Provenance

كل claim يجب أن يعرف مصدره وسلسلة تحوله: من أي ملف؟ أي فقرة؟ أي extractor؟ أي court؟ أي user approval؟ هذا مهم جدا لتقليل الهلوسة وإعادة التدقيق.

استرجاع متعدد

عند السؤال، لا يكفي retrieve النصوص المشابهة. يسترجع النظام: atoms، relations، prior failures، tools، metrics، and court verdicts. هذا هو الفرق بين HyperKB وRAG عادي.

عقد التنفيذ التقني

لكي لا يبقى محور "ذاكرات MICAI وأنواع التخزين" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 26: دورة الاستدلال في MICAI

Reasoning loop قابلة للتنفيذ

المدخل

يدخل المستخدم بمهمة. النظام يبني Task Object: goal، constraints، domain، expected_artifact، risk، deadline_context، user_style، needed_tools. هذه المرحلة تمنع سوء الفهم قبل التفكير.

الدورة

1 Clarify internally when possible، 2 Retrieve project memory، 3 Extract differences، 4 Score MCD/QMCD، 5 Build working context، 6 Generate candidates، 7 Court attack، 8 Repair، 9 Execute، 10 Verify، 11 Memory patch.

حالات التوقف

يتوقف النظام عندما يحقق criteria: artifact exists، tests pass، answer covers constraints، residual risks are declared. لا يتوقف فقط لأن النص اكتمل.

قابلية البرمجة

يمكن تنفيذ الدورة كـ state machine. الحالات: INGEST، RETRIEVE، DIFFERENCE، MAP، PLAN، COURT، EXECUTE، VERIFY، MEMORIZE، RESPOND. كل انتقال له شروط ومخرجات.

ميزة

هذه الدورة تجعل النظام قابلا للتحسين: إذا فشل VERIFY نعود إلى COURT أو EXECUTE؛ إذا فشل RETRIEVE نعيد indexing؛ إذا فشل PLAN نرفع الموارد. لا نعيد المحاولة عشوائيا.

عقد التنفيذ التقني

لكي لا يبقى محور "دورة الاستدلال في MICAI" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 27: neoAGI و MICAGI كنمط تشغيل متعدد الوكلاء

تحويل MICAI إلى منظومة agents

الفكرة

عندما يكبر النظام، لا يكفي agent واحد. neoAGI/MICAGI يوزع الدورة على وكلاء: Researcher، Architect، Coder، Critic، Evaluator، Memory Curator، Tool Operator، Safety Judge. لكنهم يعملون تحت سؤال حاكم واحد.

بروتوكول التواصل

كل وكيل لا يرسل نصا حرا فقط، بل objects: finding، objection، patch، metric، action. هذا يقلل الفوضى بين الوكلاء ويجعل orchestrator قادرا على الدمج.

منع حرب الوكلاء

تعدد الوكلاء قد ينتج ضجيجا. لذلك يحكمهم QMCD وGlobal Workspace. أي مخرج لا يخدم السؤال أو لا يحمل أثره يختصر أو يحذف.

سياسة التخصص

Researcher يجلب الأدلة، Architect يبني النظام، Coder ينفذ، Critic يهدم، Evaluator يقيس، Memory Curator يثبت التعلم. لا يطلب من كل وكيل فعل كل شيء.

مقياس النجاح

ليس عدد الوكلاء، بل انخفاض الأخطاء وزيادة الأثر مقارنة بالوكيل الواحد. إذا زاد التعدد التكلفة دون أثر، يخفضه الراوتر.

عقد التنفيذ التقني

لكي لا يبقى محور "neoAGI و MICAGI كنمط تشغيل متعدد الوكلاء" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 28: توجيه النماذج المتعددة

Model Orchestration وفق السرعة والكلفة والجودة

المبدأ

في MICAI لا يختار النظام النموذج الأعلى دائما. يختار حسب المهمة. التلخيص الرخيص لنموذج سريع، الاستدلال المعماري لنموذج تفكير، الكود لنموذج مبرمج، الصور لنموذج بصري، والتحقق لنموذج ناقد.

بيانات الراوتر

يحتاج سجل أداء لكل نموذج: tool_support، context_limit، hallucination_rate، code_score، reasoning_score، cost_per_1k، latency_avg، recent_failures. هذه البيانات تحدث دوريا عندما تتوفر.

دالة الاختيار

model_score = task_fit * quality * reliability / (cost * latency_penalty) مع عقوبة للمخاطر والسجل السيئ. في المهام الحساسة ترفع reliability على cost.

Cascade

يمكن تشغيل cascade: نموذج سريع يقترح، نموذج قوي يهدم، نموذج أداة ينفذ، evaluator يتحقق. هذا يقلل الكلفة مقارنة بتشغيل النموذج الأقوى في كل المراحل.

Fallback

إذا فشل نموذج أو تجاوز الحد، ينتقل النظام إلى بديل. كل فشل يسجل في Model Performance Memory حتى يتحسن routing لاحقا.

عقد التنفيذ التقني

لكي لا يبقى محور "توجيه النماذج المتعددة" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 29: HyperKnowledge OS

قاعدة المعرفة كنظام تشغيل للذكاء

التعريف

HyperKB OS ليس مجلد ملاحظات، بل نظام تشغيل معرفي يدعم hypothesis generation، skill extraction، planning، reasoning، retrieval، tool creation، critique، وmemory. هو البيئة التي يعيش فيها MICAI.

النواة

Kernel يتكون من: Versioning، Evaluation، Reasoning APIs، Retrieval، Indexing، Ontology، Atomization، Normalization، Ingestion، Permissions، and Artifact Registry.

الفرق عن KB عادية

قاعدة المعرفة العادية تجيب: أين المعلومة؟ HyperKB يجيب: ما الفرق؟ ما السؤال؟ ما الأداة؟ ما الاعتراض؟ ما الذاكرة؟ ما الأثر؟ لذلك هي OS لا أرشيف.

ملفات النظام

يمكن تنظيمها كـ /metrics/، /court/، /memory/، /artifacts/، /benchmarks/، /tools/، /agents/، /prompts/، /schemas/، /theories/. كل مجلد له عقود وواجهات لا مجرد نصوص.

دورها في مشروعك

كل ورقة أو نظرية أو prompt يتحول إلى module داخل HyperKB. عندما تطلب نسخة تقنية، يستدعي النظام modules المتعلقة بـ MICAI ويفصلها عن الفلسفة العامة.

عقد التنفيذ التقني

لكي لا يبقى محور "HyperKnowledge OS" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 30: Pipeline الإدخال والتطبيع

من الملفات الخام إلى atoms دقيقة

الاستقبال

يدخل النظام PDFs، MD، DOCX، صور، روابط، محادثات، كود. لكل نوع parser مناسب. الهدف هو استخراج النص والبنية: عناوين، جداول، هوامش، معادلات، claims، أمثلة، أدوات.

التنظيف

تنظيف لا يعني حذف التفاصيل. يعني إزالة الضجيج: تكرار، encoding، broken lines، OCR errors، headers، مع الحفاظ على مواضع المصدر. كل تعديل يجب أن يكون قابلا للتتبع.

Atomization

يقسم النص إلى atoms ذات حجم مناسب. atom كبير يضعف الاسترجاع، وatom صغير يفقد السياق. الحل: chunk هرمي: فقرة، قسم، صفحة، وثيقة، مع روابط بينها.

تصنيف

كل atom يصنف: assumption، claim، source، limitation، objection، example، algorithm، metric، method، theorem، definition. هذا التصنيف يسمح لـ QMCD بجلب النوع المناسب للسؤال.

إخراج

IngestionReport: عدد الملفات، atoms، جودة الاستخراج، أخطاء، مصادر ناقصة، عناصر تحتاج مراجعة بشرية. لا يبدأ reasoning قبل معرفة جودة الإدخال.

عقد التنفيذ التقني

لكي لا يبقى محور "Pipeline الإدخال والتطبيع" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 31: الأونتولوجيا والمخطط البياني

Graph يجعل المعرفة قابلة للسببية والدمج

العقد

أهم العقد: Artifact، Experiment، Objection، Evidence، Claim، Dataset، Tool، Algorithm، Module، Metric، Concept، SubTheory، Theory، Action، UserGoal. كل عقدة تحمل id وversion وconfidence.

الحواف

produces، evaluated_by، repairs، attacks، specializes، generalizes، depends_on، measures، implements، contradicts، supports، defines.
هذه الحواف تجعل العلاقات صريحة لا دفينة داخل النص.

مثال

MCD measures Difference Object. QMCD governs MCD by Question Object. HAKIM-MCD implements MICAI. Perturbation Court attacks Candidate Action. World-Effect Loop evaluates Action. Self-Reconstruction Memory updates Theory Version.

فوائد graph

يمكن للنظام تتبع أثر تعديل تعريف واحد على عشر نظريات، أو معرفة أي أدوات تعتمد على وحدة معينة، أو استخراج الاعتراضات غير المعالجة. هذا غير ممكن بملفات Markdown وحدها.

قيود

Graph قد يصبح ثقيلا إذا أدخلت كل علاقة. لذلك يستعمل MCD لاختيار العلاقات المهمة، وتستعمل المحاكم لتنظيف الحواف الضعيفة.

عقد التنفيذ التقني

لكي لا يبقى محور "الأونتولوجيا والمخطط البياني" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 32: الاسترجاع وGraphRAG داخل MICAI

استرجاع محكوم بالفرق والسؤال

RAG العادي

يبحث غالبا عن مقاطع مشابهة للسؤال. هذا مفيد لكنه ناقص، لأنه قد يجلب نصا مشابها لا يغير الجواب، أو يتجاهل اعتراضا حاسما غير مشابه لغويا.

GraphRAG الوعائي

يبدأ من Question Object، يستخرج مفاهيمه، يبحث vectorيا عن atoms، ثم graphويا عن الأدلة والاعتراضات والأدوات والعلاقات، ثم يعيد ترتيب النتائج بـ QMCD.

Evidence Pack

المخرج ليس قائمة نصوص، بل Evidence Pack: supporting_atoms، contradicting_atoms، missing_evidence، relevant_tools، prior_failures، key_metrics. هذا pack يدخل Global Workspace.

استرجاع الاعتراضات

جزء إلزامي: retrieve_objections. أي جواب كبير يجب أن يعرف أقوى اعتراضات المشروع عليه. هذه وظيفة مركزية في محكمة الهدم.

مؤشرات

Retrieval Precision، Evidence Coverage، Contradiction Recall، Citation Traceability، and Answer Improvement. الاسترجاع الجيد ليس الأكثر وفرة بل الأكثر تأثيرا.

عقد التنفيذ التقني

لكي لا يبقى محور "الاسترجاع وGraphRAG داخل MICAI" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 33: HyperSkills Extraction

استخراج المهارات من المعرفة

التعريف

HyperSkill هي تحويل معرفة إلى إجراء قابل لإعادة الاستخدام. مثال: من نظرية محكمة الهدم نستخرج skill: نقد ورقة علمية. من MCD نستخرج skill: ترتيب الفروق. من HyperKB نستخرج skill: بناء أونتولوجيا.

Schema

skill_id، trigger، inputs، procedure، tools، expected_outputs، metrics، failure_modes، examples، version. لا تعتمد skill إذا لم يعرف النظام متى يستدعيها ومتى لا يستدعيها.

التوليد

يقرأ النظام theory atoms، يستخرج الأفعال الضمنية، يحولها إلى خطوات، يربطها بالأدوات، يضيف اختبارات، ثم يمررها إلى المحكمة. هكذا تتحول النظرية إلى tool-thinking.

التراكم

كل مشروع ينتج skills جديدة. مثلا: AgentOrchestrator، KBIngestor، CourtRunner، MCDScorer، TheoryMapper، PaperBuilder، ArtifactVerifier. هذه المهارات تصبح مكتبة تشغيل.

اختبار

تقاس HyperSkill بقدرتها على إنتاج نتيجة متكررة بجودة ثابتة، لا بمجرد جمال وصفها.

عقد التنفيذ التقني

لكي لا يبقى محور "HyperSkills Extraction" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 34: Prompt OS وبروتوكولات الوكلاء

البرومبت كواجهة تشغيل لا كنص ساحر

المبدأ

في MICAI البرومبت ليس تعويذة، بل contract. يحدد الدور، المدخلات، المخرجات، القيود، معايير النجاح، أسئلة الهدم، وصيغة trace. كل prompt يجب أن يكون versioned ومقيسا.

Prompt Modules

System Identity، Task Contract، Context Pack، Tool Policy، Output Schema، Court Checklist، Memory Patch Rules، Safety Constraints. تجميع هذه الوحدات ينتج prompt قابل للصيانة.

Agent Protocol

كل وكيل يتلقى: mission، active_question، evidence_pack، output_schema، forbidden_actions، evaluation_metric. ويعيد: result، confidence، assumptions، evidence، objections، next_actions.

منع الانحراف

بروتوكولات الوكلاء تمنع المخرجات الطويلة غير المفيدة. إذا طلب من الوكيل اعتراضا، لا ينتج مقالا؛ ينتج broken_claims وrepairs. إذا طلب منه خطة، ينتج steps وrisks وmetrics.

صلة المشروع

الميغابرومبتات السابقة تصبح هنا وحدات OS: ليست نصوصا ضخمة فقط، بل templates قابلة للتركيب حسب المهمة.

عقد التنفيذ التقني

لكي لا يبقى محور "Prompt OS وبروتوكولات الوكلاء" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 35: منهجية تدريب MICAI

كيف يتعلم النظام من البيانات والأثر

أنواع التدريب

لا يعتمد MICAI على fine-tuning فقط. هناك training معرفي عبر الذاكرة، training تقييمي عبر المحاكم، training إجرائي عبر HyperSkills، training من الأثر عبر World-Effect، وtraining من الفشل عبر Failure Memory.

Self-supervised MCD

يمكن توليد أمثلة: نصان، ما الفرق المهم؟ ثم يدرب scorer على ترتيب الفروق. الهدف تعليم النظام رؤية الفرق لا حفظ الأجوبة.

Court Distillation

تجمع المحاكم اعتراضات وrepairs. يمكن تحويلها إلى بيانات تدريب لمقيم أو ناقد: claim -> objections -> repaired claim -> verdict. هذا يحسن الصرامة.

Outcome Feedback

كل artifact ينتج نتيجة: نجح/فشل/احتاج تعديل. تربط النتيجة بالقرارات السابقة: prompt، MCD، retrieval، routing. هكذا يتعلم النظام أي سياسات أفضل.

حذر

أي تدريب يحتاج provenance وتنظيف. لا يجوز إدخال مخرجات هلوسية كحقائق. لذلك تمر بيانات التدريب نفسها عبر محكمة ومؤشرات ثقة.

عقد التنفيذ التقني

لكي لا يبقى محور "منهجية تدريب MICAI" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 36: مجموعة التقييم والاختبارات

Benchmarks خاصة بالمشروع لا عامة فقط

لماذا

التقييمات العامة لا تكفي؛ مشروع MICAI يحتاج قياس قدرته على بناء نظريات، استخراج أدوات، هدم ادعاءات، إنتاج PDFs سليمة، وإدارة ذاكرة طويلة.

اختبارات MCD

يعطى النظام مادة خام وتطلب منه أهم 20 فرق. يقارن مع gold set بشري أو court-approved. تقاس recall، precision، وimpact ranking.

اختبارات QMCD

يعطى سؤال حاكم ومقاطع كثيرة. يقاس هل استرجع ما يخدم السؤال فعلا، وهل تجاهل المشتتات، وهل ولد أسئلة فرعية قوية.

اختبارات HAKIM

كل وحدة تختبر منفردة ثم في integration. Difference Extractor، Mapper، Router، Court، Memory. هذا يكشف موضع العطل.

اختبارات الأثر

تقييم artifact: هل الملف فتح؟ هل الصفحات سليمة؟ هل الكود يعمل؟ هل الخطة قابلة للتنفيذ؟ هل خفضت المحكمة الأخطاء؟ هذه الاختبارات أهم من انطباع الأسلوب.

عقد التنفيذ التقني

لكي لا يبقى محور "مجموعة التقييم والاختبارات" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 37: مكافحة الهلوسة وفقد السياق

من السلامة المعرفية إلى التنفيذ

المشكلة

مشروع نظري ضخم معرض لخطرين: هلوسة تفاصيل غير مؤسسة، وفقد أجزاء مهمة بسبب ضغط السياق. MICAI يعالج الخطرين ببنية sources، memory، court، graph، وverification.

ضد الهلوسة

كل claim جديد يأخذ status: sourced، inferred، speculative، rejected. لا يجوز عرض inference كحقيقة. المحكمة تطلب evidence links، وتخفض الثقة عند غيابها.

ضد الفقد

تستعمل ذاكرة هرمية: summary لا يلغي الأصل، atom لا يلغي الصفحة، graph لا يلغي النص. عند الحاجة يرجع النظام إلى المصدر. هذا يمنع اختزال المشروع في ملخصات سطحية.

Trace

كل مخرج كبير يجب أن يحمل داخليا trace: task -> retrieved atoms -> differences -> questions -> court -> output. يمكن إخفاء trace التفصيلي عن المستخدم، لكن يجب أن يكون موجودا للتدقيق.

مؤشرات الخطر

High novelty without evidence، broad claim with low coverage، missing opposition، source ambiguity، old memory conflict. عند ظهورها يرفع النظام عمق المحكمة أو يصرح بالحدود.

عقد التنفيذ التقني

لكي لا يبقى محور "مكافحة الهلوسة وفقد السياق" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 38: محكمة الهدم كبروتوكول رسمي

قوائم فحص قابلة للبرمجة

Checklist 1: الدليل

ما مصدر الادعاء؟ هل هو من ذاكرة المشروع أم من ملف مباشر؟ هل هو استنتاج؟ هل يحتاج citation؟ هل يوجد تعارض مع مصدر أعلى؟

Checklist 2: الاتساق

هل التعاريف ثابتة؟ هل MICAI وMCAI غير مختلطين؟ هل MCD يستعمل كمقياس لا كشعار؟ هل QMCD يضبط السؤال فعلا؟

Checklist 3: التشغيل

هل لكل نظرية وحدة أو أداة؟ هل توجد inputs/outputs؟ هل توجد metrics؟ هل يمكن تنفيذها في كود أو سير عمل؟

Checklist 4: الحدود

ما الذي لا تدعيه النظرية؟ أين تحتاج تجربة؟ أين هي استعارة تقنية لا حقيقة علمية مثبتة؟ هذا مهم للحفاظ على الرصانة.

Checklist 5: الإصلاح

كل اعتراض ينتج repair: حذف، تقييد، إضافة دليل، إعادة تعريف، تقسيم نظرية، أو تحويلها إلى فرضية. الاعتراض بلا إصلاح لا يكفي في النظام.

عقد التنفيذ التقني

لكي لا يبقى محور "محكمة الهدم كبروتوكول رسمي" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 39: التخطيط والتنفيذ بالأدوات

Tool Use داخل الوكالة التقنية

Tool Bus

MICAI يحتاج ناقل أدوات: documents، image، email، calendar، database، code، pdf، file، web. لكن الاستدعاء ليس عشوائيا؛ Agency Selector يقرر، والراوتر يخصص، والمحكمة تتحقق.

عقد الأداة

كل أداة تعرف: verification، failure_modes، latency، cost، auth_scope، side_effects، required_inputs، capability. الأدوات ذات أثر خارجي تحتاج تحقق وسلامة أكثر من أدوات القراءة.

خطة التنفيذ

Action Plan يتكون من rollback، checkpoints، expected_outputs، tools، steps. عند إنشاء PDF مثلا: تأليف MD، تحويل HTML، توليد PDF، render، فحص الصفحات، إصلاح، تسليم.

التحقق

لا يعتبر tool call ناجحا لمجرد أنه عاد بنتيجة. يجب فحص النتيجة: هل الملف موجود؟ هل يفتح؟ هل الصفحات صحيحة؟ هل لم تنكسر العربية؟

دور MICAI

هنا تتحول النظريات إلى هندسة إنتاج. الذكاء لا يكتفي بالجواب؛ يصنع artifact ويتحقق منه.

عقد التنفيذ التقني

لكي لا يبقى محور "التخطيط والتنفيذ بالأدوات" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 40: السلامة والحوكمة التقنية

حدود الفعل الذكي داخل MICAI

الحوكمة

أي نظام agentic يحتاج قواعد: ما الذي يجوز تنفيذه؟ ما الذي يحتاج تأكيدا؟ ما الذي يحتاج رفضا؟ ما البيانات الحساسة؟ ما حدود الاستخدام الخارجي؟

Provenance Governance

كل مصدر له ثقة وترخيص وسياق. لا تخلط ذاكرة محادثة مع حقيقة منشورة. لا تحول افتراضا إلى claim. لا تنسب للمشروع ما لم يظهر في الذاكرة أو الملفات.

Action Safety

الأفعال الخارجية مثل إرسال بريد، حذف ملفات، نشر محتوى، أو تعديل حسابات تحتاج موافقة صريحة أو قيود واضحة. أما التحليل والتأليف الداخلي فيمكن تنفيذه مباشرة ضمن الطلب.

Data Minimization

لا تحفظ إلا ما يفيد مستقبلا. لا تجمع بيانات أكثر من الحاجة. هذا مهم خصوصا في مشاريع outreach أو lead generation حتى لا يتحول النظام إلى جمع عشوائي غير منضبط.

مؤشر الحوكمة

Reversibility، User Confirmation Respect، Sensitive Data Exposure، Unverified Claim Rate، Compliance Pass Rate. هذه مؤشرات تجعل السلامة جزءا من التقنية.

عقد التنفيذ التقني

لكي لا يبقى محور "السلامة والحوكمة التقنية" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 41: واجهات API والعقود الداخلية

كيف تتكلم وحدات MICAI مع بعضها

MCD API

POST /mcd/score يستقبل DifferenceObject ويعيد MCDVector مع explanation. POST /mcd/rank يستقبل قائمة فروق ويعيد ترتيبا وسياسة تخصيص أولية.

Question API

POST /question/expand يعيد graph أسئلة. POST /question/align يقيس Alignment بين delta وPOST /question/court يخلق أسئلة Q. هدم.

Court API

POST /court/review يستقبل candidate_output وevidence_pack وrisk_level، ويعيد verdict وrepairs. يجب أن يكون idempotent وقابلا للتكرار عند نفس المدخلات قدر الإمكان.

Memory API

POST /artifact/register، GET /theory/version، GET /memory/retrieve، POST /memory/patch. هذه الواجهات تحفظ trace بين النظرية والمخرجات.

تنسيق موحد

يفضل JSON schemas صارمة لكل object. النص الحر يبقى في fields محددة، لكن القرارات والمقاييس تكون منظمة حتى يمكن البحث والتحليل.

عقد التنفيذ التقني

لكي لا يبقى محور "واجهات API والعقود الداخلية" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 42: المكدس البرمجي المقترح

من prototype إلى نظام إنتاج

Prototype

Python + FastAPI + SQLite/Postgres + Chroma/FAISS + NetworkX أو Neo4j + ملفات Markdown. هذا يكفي لإثبات HAKIM-MCD دون تعقيد بنية مبكر.

Production

PostgreSQL للقياسات، Neo4j أو ArangoDB للgraph، Qdrant/Weaviate للvector، S3-compatible storage للملفات، Redis للذاكرة السريعة، Celery/Temporal للjobs، OpenTelemetry للمراقبة.

Agent Runtime

يمكن استعمال LangGraph أو إطار state-machine خاص. المهم أن تكون الحالات واضحة وأن لا تتحول الوكالة إلى loop غير مراقب.

واجهة الاستخدام

Dashboard يعرض: memory patches، artifacts، routing costs، court verdicts، retrieved evidence، MCD heatmap، active task. هذا يجعل النظام قابلا للقيادة لا صندوقا أسود.

بساطة البداية

لا يلزم بناء كل شيء مرة واحدة. ابدأ بنواة: ingest -> atoms -> MCD -> question -> court -> artifact -> memory. ثم وسع إلى orchestration.

عقد التنفيذ التقني

لكي لا يبقى محور "المكدس البرمجي المقترح" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 43: المجدول وEvent Bus

تشغيل مستمر دون فوضى

Event-driven

كل فعل يولد event: file_ingested، delta_found، question_created، court_failed، artifact_built، effect_observed، memory_updated. الأحداث تجعل النظام قابلا للمراقبة والتراجع.

Scheduler

ينفذ jobs: تحديث indexes، إعادة تقييم نظريات، فحص artifacts، مراقبة مصادر، إنتاج ملخصات، تشغيل benchmarks. لكن كل job يجب أن يحمل goal وlimits.

Priority Queue

ترتب المهام حسب MCD/QMCD/risk. المهام العالية الأثر لا تضيع خلف مهام صغيرة. والمهام الخطرة لا تنفذ بلا تحقق.

Idempotency

أي job يجب أن يكون قابلا للإعادة دون تخريب: إذا أعدت ingestion لا تنسخ atoms بلا نهاية؛ إذا أعدت المحكمة لا تضيع verdicts السابقة.

Logs

السجل ليس debug فقط، بل ذاكرة هندسية: لماذا اختيرت هذه الخطة؟ لماذا فشل PDF؟ لماذا ارتفعت التكلفة؟ هذه الأسئلة تغذي Self-Reconstruction Memory.

عقد التنفيذ التقني

لكي لا يبقى محور "المجدول وEvent Bus" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 44: تجارب معيارية لإثبات MICAI

من النظرية إلى paper قابل للاختبار

Experiment 1

أدخل 20 ورقة أو ملفا من المشروع، واطلب استخراج كل النظريات والفروع. قارن output مع قائمة بشرية. قس coverage، missing theories، false merges، وdefinition consistency.

Experiment 2

اختبر MCD: أعط النظام 100 فقرة، منها 20 تحمل فروقا حاسمة. هل يرتبها في الأعلى؟ هل يشرح السبب؟

Experiment 3

اختبر المحكمة: أعطها ورقة فيها 30 خللا مقصودا: ادعاءات بلا دليل، خلط أسماء، تضخم، نقص تطبيق. قس كم خللا تكشف وكم repair صالح تنتج.

Experiment 4

اختبر HyperKB: اسأل أسئلة تحتاج ربط خمس نظريات. قارن RAG عادي مع GraphRAG-QMCD. هل الجواب أكثر اتساقا وأقل فقدا؟

Experiment 5

اختبر World Effect: اطلب artifact تقني PDF/MD/كود، ثم تحقق آليا من وجود الملف، الصفحات، الروابط، واكتمال الأقسام. هذا يثبت أن النظام يفعل لا يكتب فقط.

عقد التنفيذ التقني

لكي لا يبقى محور "تجارب معيارية لإثبات MICAI" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 45: حالة استعمال: توليد ورقة بحثية تقنية

كيف تعمل النظريات عند إنتاج وثيقة

المسار

Task Object يحدد: ورقة تقنية عن MICAI، 50 صفحة، عربية، تركيز نظامي. يسترجع HyperKB نظريات MCD/QMCD/HAKIM. يستخرج Difference Objects. يبني outline عبر QMCD. يمرر كل محور للمحكمة. ثم يولد MD/PDF ويتحقق.

MCD في الورقة

يرتب المحاور: النواة قبل الفروع، الخوارزمية قبل الأمثلة، القياس قبل الادعاء. أي جزء لا يضيف فرقا تقنيا يخفض أو يحذف.

QMCD

يحكم السؤال: كيف تتحول النظريات إلى نظام؟ لذلك لا يطيل في المعاني الروحية أو التاريخية إلا عند mapping تقني. هذا هو بالضبط طلبك: التقنية فقط.

Court

تهاجم الورقة: هل عرّفت MICAI؟ هل فصلت MCAI؟ هل لكل نظرية وحدة؟ هل يوجد API؟ هل توجد metrics؟ هل صرحت بالحدود؟

World Effect

الأثر هو ملف قابل للقراءة والتعديل، يحمل خارطة بناء النظام. إذا لم ينتج artifact واضحا فالعملية فشلت مهما كان النص جميلا.

عقد التنفيذ التقني

لكي لا يبقى محور "حالة استعمال: توليد ورقة بحثية تقنية" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 46: حالة استعمال: HyperKB لوكالة Neo Agency

تحويل المعرفة إلى تشغيل خدمات

السياق

ضمن مشاريع Beyond AI Era وAfripa Neo Agency يمكن استعمال MICAI لبناء خدمات agentic: محتوى، knowledge، branding، funnel، SEO، OS، أدوات داخلية، وفرق AI. لكن التقنية يجب أن تحترم الحوكمة والقانون والقياس.

المسار

يدخل brand brief. يستخرج النظام differences: موقع العلامة، الجمهور، العرض، القنوات، القيود. يبني سؤالا حاكما: ما النظام الذي يولد مخرجات تسويقية مستمرة؟ ثم يبني HyperSkills للمنشورات، المقالات، الرسائل، والتحليل.

الذاكرة

كل حملة تنتج memory patches: ما hook الذي نجح؟ ما الجمهور؟ ما الاعتراض؟ ما التحويل؟ بهذا يتحول العمل من خدمات يدوية إلى OS يتعلم.

السلامة

في outreach مثلا يجب احترام opt-out، المصدر المشروع، الحدود اليومية، وسمعة الدومين. MICAI لا يعني إرسال عشوائي، بل agency محكومة بالأثر والسلامة.

المقياس

Cost per useful artifact، conversion signal، reply quality، content learning rate، brand consistency. هذه تجعل neo agency نظاما لا مجرد إنتاج محتوى.

عقد التنفيذ التقني

لكي لا يبقى محور "حالة استعمال: HyperKB لوكالة Neo Agency" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 47: حالة استعمال: Orchestration لنماذج التفكير والصورة والكود

MICAI كمدير نماذج لا كنموذج واحد

المهمة

عندما تطلب بناء نظام أو وثيقة أو صورة أو فيديو أو كود، يقرر الراوتر أي نموذج أو أداة تخدم كل جزء. لا يوجد نموذج واحد هو الحل لكل شيء.

تقسيم العمل

Thinking model للتخطيط، coding model للتنفيذ، vision model للفحص، image model للتوليد، evaluator للنقد، cheap model للتلخيص. MICAI ينسق، لا يستبدل كل النماذج.

البيانات الحية

في الأنظمة المتقدمة، يراقب الراوتر متوسطات الأداء: زمن الاستجابة، جودة المخرجات، الكلفة، الأخطاء. ثم يحدث سياسات التوجيه. هذا يجعل orchestration اقتصاديا وديناميكيا.

منع التضخم

إضافة 30 نموذجا بلا سياسة لا تصنع AGI. القيمة في routing وcourt وmemory. إذا لم يسجل النظام لماذا اختار النموذج، لا يمكن تحسينه.

الناتج

OrchestrationBlock: task_type -> candidate_models -> selection_rule -> fallback -> eval_metric -> log_schema. هذا block يمكن وضعه في Hermes أو أي agentic runtime.

عقد التنفيذ التقني

لكي لا يبقى محور "حالة استعمال: Orchestration لنماذج التفكير والصورة والكود" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 48: الفشل والحدود التقنية

ما الذي يجب عدم ادعائه

حدود نظرية

MCD وQMCD مقاييس مصممة داخل المشروع وليست معايير علمية مثبتة عالميا بعد. قوتها في قابليتها للتشغيل والاختبار، لا في ادعاء أنها قانون نهائي.

حدود تنفيذ

HyperKB قد يفشل إذا كان الإدخال ضعيفا أو المصادر غير موجودة. Court قد يبالغ في النقد أو يفوت أخطاء. Router قد يختار نموذجا سيئا. Memory قد تحفظ ما لا يلزم أو تنسى ما يلزم.

خطر المصطلحات

كثرة الأسماء قد تخلق وهم اكتمال. لذلك يجب ربط كل اسم بـ schema وAPI وmetric وtest. الاسم الذي لا يقبل هذا الربط يؤجل أو يحذف.

خطر التعميم

لا ينبغي نقل نماذج الوعي V0-V6 إلى ادعاءات بيولوجية أو فيزيائية دون بحث مستقل. في هذه الورقة تستعمل كمخطط حاسوبي معرفي.

قاعدة الصرامة

كل فرع من المشروع يجب أن يقول: ماذا أدخل؟ ماذا أعالج؟ ماذا أخرج؟ كيف أقيس؟ كيف أفشل؟ كيف أصلح؟

عقد التنفيذ التقني

لكي لا يبقى محور "الفشل والحدود التقنية" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 49: ملحق تقني: مصفوفة النظريات مقابل الوحدات

جدول ذهني يربط كل نظرية بمقابلها الهندسي

مصفوفة الربط

التراكب الوعائي يقابله Layered Context Containers. قانون الفرق الفاعل يقابله Difference Extractor. MCD يقابله Scoring Engine. QMCD يقابله Question Governance Engine. الضغط التركيبي يقابله Compression/Reconstruction Engine. محكمة الهدم تقابل Court Engine. الذكاء الفتّاح يقابل Open-Ended Agency Loop. HyperKB OS يقابل Knowledge Runtime. World-Effect Loop يقابل Evaluation and Feedback Runtime.

قاعدة القبول

أي نظرية جديدة داخل المشروع لا تعتمد حتى تملأ خمس خانات: data object، algorithm، metric، court test، and world-effect. هذه القاعدة تمنع نمو المصطلحات بلا تشغيل، وتجعل المشروع قابلا للتحول إلى كود وواجهات وخدمات.

مصفوفة الأولوية

الأولوية الأولى: MCD/QMCD لأنها تقرر ما المهم. الأولوية الثانية: HyperKB لأنها تحفظ وتحضر المعرفة. الأولوية الثالثة: Court لأنها تمنع الوهم. الأولوية الرابعة: Tool/Agent Runtime لأنها تحول المعرفة إلى فعل. الأولوية الخامسة: Memory لأنها تجعل التعلم تراكميا.

خريطة التكامل

عند كل مهمة يمر النظام بهذه المصفوفة: النظرية تحدد نوع الإدراك، الوحدة تنفذ، المقياس يحكم، المحكمة تهدم، والأثر يقرر هل تحفظ النتيجة. هذا الربط هو جوهر النسخة التقنية من مشروع مساعف.

مخرج عملي

يمكن تحويل هذه الصفحة مباشرة إلى ملف YAML أو JSON يصف modules وschemas وmetrics. لذلك فهي ليست خاتمة وصفية، بل blueprint أولي لبناء repository حقيقي لـ MICAI.

عقد التنفيذ التقني

لكي لا يبقى محور "ملحق تقني: مصفوفة النظريات مقابل الوحدات" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.

صفحة 50: الخريطة النهائية لبناء MICAI

Roadmap من 12 مرحلة

مرحلة 1-3

1 بناء schema للكائنات: Atom، Difference، Question، Action، Effect. 2 بناء ingestion يحول ملفات المشروع إلى atoms. 3 بناء MCD scorer أولي بقواعد واضحة وأوزان قابلة للتعديل.

مرحلة 4-6

4 بناء Question Engine وفضاء السؤال. 5 بناء GraphRAG يسترجع الأدلة والاعتراضات. 6 بناء Global Workspace لتجميع السياق النشط.

مرحلة 7-9

7 بناء Court Engine بخمس قوائم هدم. 8 بناء Agency Selector يختار الأفعال. 9 بناء Artifact Builder ينتج MD/PDF/code مع verification.

مرحلة 10-12

10 بناء World-Effect Loop لقياس النجاح. 11 بناء Self-Reconstruction Memory. 12 بناء Dashboard وbenchmarks ودورة تحسين مستمرة.

الناتج المتوقع

بعد هذه المراحل يصبح المشروع ليس مجموعة نظريات فقط، بل نظاما تقنيا يستطيع قراءة مواده، بناء نظريات جديدة، نقدها، تحويلها إلى أدوات، إنتاج artifacts، وتعلم ما يصلح وما لا يصلح.

الخلاصة

MICAI هو دمج الفرق والسؤال والوكالة والأثر. MCD يرى ما تغير، QMCD يسأل لماذا يهم، HAKIM-MCD يشغل الدورة، HyperKB OS يحفظ الذاكرة، وneoAGI/MICAGI ينشرها كمنظومة وكلاء.

عقد التنفيذ التقني

لكي لا يبقى محور "الخريطة النهائية لبناء MICAI" وصفا نظريا، يعامل داخل MICAI كعقد تشغيل: inputs واضحة، transformation محدد، outputs قابلة للتخزين، metric للحكم، failure mode معروف، وrepair path عند الفشل. يربط هذا العقد بين النظرية والكود: كل مفهوم يتحول إلى schema، وكل schema إلى خدمة، وكل خدمة إلى سجل أثر. عند بناء prototype يكتب لهذا المحور test صغير: مدخل نموذجي، نتيجة متوقعة، اعتراض محتمل، ومعيار قبول. بهذه الطريقة تتراكم النظريات كحزمة modules قابلة للدمج لا كمقالات منفصلة.