قاعدة المعرفة كنظام تشغيل للأنظمة الوكيلة المتجهة نحو الذكاء العام

Knowledge Base as an Operating System for AGI-Oriented Agentic Systems

---

ورقة نظرية معمارية - إصدار بحثي موسع

September 2026

HKOS / HyperKnowledge OS

خريطة الورقة

1. المشكلة: لماذا الذكاء النموذجي لا يساوي ذكاء النظام
2. تعريف KB-as-OS
3. الانتقال من RAG إلى Knowledge Resolution
4. Executable Epistemic Graph
5. الحالة المعرفية وعدم اليقين
6. الزمن وصلاحية المعرفة
7. Context Compilation
8. Knowledge Scheduler
9. Knowledge-to-Skill Compilation
10. World-Effect Loop
11. Verification Court
12. Governed Writeback
13. التعلم المستمر
14. التخطيط المقيد بالمعرفة
15. Multi-Agent Knowledge
16. Self-Reconstruction
17. الفرضيات القابلة للتكذيب
18. التصميم التجريبي
19. Ablations
20. Failure Modes
21. حدود النظرية
22. HKOS داخل MOUSSAIF AGI OS
23. برنامج بحث مقترح
24. الخاتمة

الملخص

تجادل هذه الورقة بأن قاعدة المعرفة في الأنظمة الوكيلة المتقدمة ينبغي ألا تعامل بوصفها مخزناً خارجياً للمعلومات أو طبقة استرجاع فحسب، بل باعتبارها Operating Substrate تشارك مباشرة في التحكم في الإدراك والتخطيط والفعل والتحقق والتعلم والاستمرارية الزمنية للوكيل. نسمي هذا التصور KB-as-OS. في هذا التصور تتحول وحدات المعرفة من نصوص قابلة للاسترجاع إلى كيانات تشغيلية typed ومؤرخة ومسنودة بالدليل وقابلة للعنونة، لها dependencies وشروط صلاحية ودرجات ثقة وعلاقات سببية وإجرائية وواجهات مع المهارات والأدوات وعمليات التحقق.

لا تدعي الورقة أن المصطلح يمثل اتجاهاً معيارياً مستقراً في الأدبيات، بل تقدمه كفرضية معمارية تركيبية تتقاطع مع memory-centric agents وcognitive architectures وknowledge graphs وtool-using agents وlifelong learning وplanning. وتقدم نموذجاً رسمياً لـ Executable Epistemic Graph ودورة تشغيل من Knowledge Resolution إلى Action ثم Verification ثم Writeback، مع فرضيات قابلة للتكذيب لقياس أثر تحويل المعرفة إلى substrate تشغيلية.

1. المشكلة: الذكاء النموذجي لا يساوي ذكاء النظام

يمكن لنموذج لغوي قوي أن ينتج استدلالاً ممتازاً في لحظة واحدة، ومع ذلك يفشل كوكيل مستمر. السبب أن الوكيل طويل الأفق يحتاج أكثر من الاستدلال الآني: يحتاج حالة عالم مستقرة، ذاكرة للأحداث، تمثيلاً للشك، معرفة بالإجراءات، شروط صلاحية، provenance، سياسات تحديث، وقدرة على التحقق من أثر أفعاله. لذلك فإن توسيع السياق وحده لا يحل المشكلة؛ فالمشكلة ليست فقط في مقدار ما يدخل النافذة، بل في كيفية إدارة المعرفة عبر الزمن وكيف تتدخل هذه المعرفة في القرار.

```
Agent Intelligence != Single-Pass Model Intelligence
```

أعمال مثل MemGPT تجعل استعارة نظام التشغيل قوية عند إدارة السياق والذاكرة، وCoALA تفصل بين الذاكرة والفعل وعملية اتخاذ القرار، بينما تظهر Generative Agents قيمة الانتقال من التجربة الخام إلى reflection ثم التخطيط. النقلة المقترحة هنا أوسع: المعرفة نفسها تصبح جزءاً من control plane للوكيل.

2. تعريف KB-as-OS

في قاعدة معرفة تقليدية يمكن تمثيل المحتوى على أنه مجموعة من العناصر أو الوثائق. أما في KB-as-OS فكل وحدة معرفة تحمل metadata تشغيلية تسمح للنظام باستخدامها ليس فقط للإجابة عن سؤال بل لضبط السلوك.

```
k_i = <content, type, scope, time, provenance, confidence, relations,
  preconditions, dependencies, permissions, tests, actions, status>
```

بهذا المعنى تصبح الجملة: "الخدمة X تتطلب token صالحاً" أكثر من معلومة وصفية؛ يمكن أن تتحول إلى dependency وprecondition وvalidator وfailure-recovery policy. المعرفة هنا تقيد الفعل، تمكّنه، وتحدد ما ينبغي اختباره قبل التنفيذ وبعده.

> **الأطروحة المركزية:** تصبح المعرفة حاسمة حسابياً عندما لا تكون مجرد مادة يقرأها النموذج، بل حالة قابلة للتنفيذ تقيد وتُمكّن وتفسر الفعل وتعيد كتابة نفسها بعد التحقق.

3. من RAG إلى Knowledge Resolution

RAG التقليدي يحل مشكلة إيجاد أدلة ذات صلة بسؤال. لكن الوكيل المستمر يحتاج إلى حل أوسع: ما الحالة المعرفية ذات الصلة بهذا الهدف وهذه اللحظة وهذه الأداة وهذه المخاطرة؟ لذلك نقترح استبدال فكرة retrieval المفردة بـKnowledge Resolution متعدد الأنماط.

```
R(g, s_t) = R_semantic + R_graph + R_causal + R_procedural + R_temporal
                                + R_episodic
```

بعد الاسترجاع الأولي تأتي مرحلة resolution تشمل التحقق من الصلاحية الزمنية، permissions، درجة الثقة، التناقضات، dependencies والتكلفة. والنتيجة ليست بالضرورة نصاً يدخل prompt، بل مجموعة objects يمكن أن تقود التخطيط مباشرة.

4. Executable Epistemic Graph

نقترح تمثيل المعرفة كبيان معرفي-تشغيلي موحد. بعض العقد تمثل حقائق، وبعضها فرضيات، أهدافاً، إجراءات، مهارات، أفعالاً واختبارات. العلاقات لا تكون فقط semantic similarity، بل تدعم وتناقض وتتطلب وتسبب وتمكّن وتبطل وتُختبر بواسطة.

```
G_K = (V, E)
V = V_F ∪ V_H ∪ V_G ∪ V_P ∪ V_S ∪ V_A ∪ V_T
```

ميزة هذا التمثيل أنه يسمح بتوجيه القرار عبر بنية المعرفة نفسها. فإذا تبين أن عقدة معرفية باطلة زمنياً يمكن إبطال الخطط والمهارات التي تعتمد عليها عبر dependency propagation، بدلاً من انتظار أن يقع الخطأ في التنفيذ.

5. الحالة المعرفية وعدم اليقين

الوكيل العام لا يحتاج فقط إلى facts، بل يحتاج إلى معرفة ما مدى ثقتها، وما الذي يناقضها، وما الذي لم يحسم بعد. لذلك نعرّف حالة معرفية تتضمن beliefs وعدم اليقين وحالة التناقض.

```
E_t = (B_t, U_t, C_t)
Belief(k) = f(source_quality, recency, corroboration, contradictions,
test_results)
```

هذه الطبقة تمنع مشكلة شائعة في الذاكرة الوكيلة: تحويل إجابة سابقة للنموذج إلى حقيقة دائمة فقط لأنها كُتبت سابقاً. يجب أن يحمل كل claim مساراً واضحاً من provenance إلى التحقق.

6. الزمن وصلاحية المعرفة

المعرفة الوكيلة ليست ساكنة. حقيقة صحيحة أمس قد تكون خاطئة الآن. ولذلك ينبغي فصل event time وobservation time وwriteback time وcurrent validity. إن لم نفعل ذلك يظهر نوع من الهلوسة الزمنية: استخدام حقيقة منتهية الصلاحية كما لو كانت آنية.

```
k = (statement, t_valid_from, t_valid_to, t_observed, t_written)
```

ويستحسن أن يكون expiration policy جزءاً من metadata. بعض المعارف تبقى سنوات، وبعضها يجب إعادة التحقق منه قبل كل فعل عالي الأثر.

7. Context Compilation

حتى مع قاعدة معرفة مثالية لا يمكن إدخال كل شيء إلى نافذة النموذج. لذلك يجب أن توجد طبقة Compiler تختار تمثيلاً مناسباً للمعرفة بحسب الهدف والميزانية والمخاطر والحالة الحالية.

```
Context_t = Compile(K*, Goal_t, State_t, Budget_t)
```

- Admission: ما الذي يدخل السياق؟
- Compression: بأي مستوى من التجريد؟
- Eviction: ما الذي يخرج؟
- Pinning: ما الذي يجب أن يبقى؟
- Isolation: ما الذي يجب ألا يختلط بسياقات أخرى؟

بهذا يصبح context مورداً محكوماً، لا مجرد مساحة نصية. وهذه هي النقطة التي تربط KB-as-OS مباشرة بفكرة virtual context management.

8. Knowledge Scheduler

كما يقرر نظام التشغيل أي process يحصل على المعالج، يمكن لـKnowledge Scheduler أن يقرر أي بنى معرفية يجب تفعيلها الآن. ليس الأكثر تشابهاً نصياً هو دائماً الأكثر أهمية؛ قد تتقدم معرفة منخفضة التشابه لأنها تحمل تحذيراً أمنياً أو dependency حاسمة.

```
Activation(k) = α Rel(k,g) + β Rel(k,s) + γ Urgency(k) + δ Risk(k) + ε
Dependency(k)
```

ويؤدي scheduler دوراً شبيهاً بالانتباه المؤسسي للوكيل: يمنع المعرفة المهمة من أن تغرق داخل كم ضخم من الوثائق.

9. Knowledge-to-Skill Compilation

الاختبار الأقوى لقيمة المعرفة هو قدرتها على أن تتحول إلى إجراء يمكن إعادة استعماله. لذلك نقترح خط أنابيب يحول المعرفة التصريحية إلى مرشح procedural ثم skill ثم skill موثقة.

```
Declarative Knowledge -> Procedural Candidate -> Skill -> Validated
Skill
```

كل skill يجب أن تحتوي GoalClass وPreconditions وProcedure وTools وExpectedEffects وTests وFailureModes. أعمال مثل Voyager تدعم قيمة مكتبة skills المتراكمة، لكن هنا نربط كل skill صراحةً بمعرفة مصدرية واختبارات وصلاحية.

10. World-Effect Loop

الوكيل لا ينجح عندما يرسل استدعاء tool فقط. النجاح يتطلب أن تكون الحالة المرصودة بعد التنفيذ متوافقة مع الحالة المطلوبة. لذلك يجب أن تكون write action متبوعة بـreadback مستقل كلما أمكن.

```
Intent -> Authorization -> Preconditions -> Action -> Confirmation ->
Readback -> Classification -> Writeback
ActionSuccess != GoalSuccess
Success = similarity(ExpectedWorldState, ObservedWorldState) >= τ
```

بهذا نمنع execution hallucination، أي اعتبار العملية التقنية الناجحة دليلاً على تحقق الهدف الواقعي.

11. Verification Court

قاعدة معرفة تستطيع تحديث نفسها بلا حوكمة يمكن أن تسمم نفسها بسرعة. لذلك نقترح محكمة تحقق معرفية تمر عبرها candidate knowledge قبل أن تصبح canonical. يمكن للمحكمة أن تفحص provenance والتناقضات والصلاحية الزمنية والاستقلال بين المصادر ونتيجة اختبارات الواقع.

```
CandidateKnowledge -> VerificationCourt -> {Accept, Reject, Quarantine}
```

عند عدم اليقين، quarantine أفضل من الترقية أو الحذف. هذا يسمح بتراكم فرضيات غير محسومة من دون مساواتها بالحقائق.

12. Governed Writeback

ليس كل trace جديراً بالحفظ. ينبغي أن يكون writeback قراراً اقتصادياً ومعرفياً. نحتاج إلى utility وnovelty وconfidence مقابل كلفة التخزين ومخاطر التسميم.

```
Write(k) iff Utility(k) × Confidence(k) × Novelty(k) > Cost(storage + risk)
```

ثم يوضع العنصر في طبقة مناسبة: validated، episodic، temporary أو canonical. ويسمح ذلك بذاكرة ذات درجات، بدلاً من مستودع يتضخم بلا تمييز.

13. من الذاكرة إلى التعلم المستمر

Memory تسجل الماضي، أما knowledge فتقدم تفسيراً منظماً للماضي والحاضر والقواعد والإجراءات. يمكن إذن النظر إلى الذاكرة بوصفها مورداً داخل substrate معرفية أوسع، مع إبقائها طبقة وظيفية مستقلة.

```
Memory = recorded past
Knowledge = structured interpretation of past, present, rules, procedures, models
```

ويتحقق التعلم النظامي عندما يغير الفعل الحالي ما سيعرفه النظام وما سيسترجعه وما سيختاره في المهمة اللاحقة، حتى إن لم تتغير أوزان النموذج الأساسي.

14. التخطيط المقيد بالمعرفة

التخطيط الحر قد يولد مسارات مقنعة لغوياً لكنها غير قابلة للتنفيذ. في KB-as-OS يمكن تحويل التخطيط إلى search مقيد بالمعرفة والpermissions والpreconditions.

```
P* = argmax_P [GoalAchievement(P) - Risk(P) - Cost(P)]
subject to Dependencies(K) ∧ Permissions(K) ∧ Preconditions(K)
```

هذا يسمح بإسقاط المسارات غير الصالحة مبكراً، ويجعل planner يستفيد من failure knowledge وليس فقط من تعليمات المهمة.

15. Multi-Agent Knowledge

عند تعدد الوكلاء لا يكفي shared vector store. يجب فصل المعرفة العالمية عن معرفة الوكيل الفردية ومعرفة المهمة والمعرفة الخاصة والمعرفة المؤقتة. كما يجب تحديد من يملك حق ترقية المعرفة إلى المستوى العالمي.

```
K = {K_global, K_agent_i, K_mission, K_private, K_ephemeral}
```

هذا مهم لمنع agent محلي من نشر استنتاج غير موثوق إلى كل النظام، ولإتاحة اختلاف epistemic perspectives عندما تكون المسألة غير محسومة.

16. Self-Reconstruction

الوكيل طويل العمر يجب أن يستطيع العودة بعد انقطاع runtime. لذلك لا تكفي المحادثة الجارية كمخزن للهوية. نحتاج إلى حالة تشغيلية قابلة لإعادة البناء تضم الهوية والأهداف والمهام النشطة والمعرفة والذاكرة والقدرات والpermissions وحالة العالم.

```
Self_t = {identity, goals, active_missions, knowledge, memory,
          capabilities, permissions, world_state}
Self_(t+1) ≈ Reconstruct(Artifacts_t)
```

هنا تصبح KB-as-OS جزءاً من استمرار الهوية الوظيفية للوكيل، لا مجرد آلية لإحضار السياق.

17. فرضيات قابلة للتكذيب

1. **H1 - Long-Horizon Continuity:** سيقل معدل إعادة اكتشاف المعرفة والأخطاء المتكررة مقارنة بـRAG عادي.

2. **H2 - Contradiction Resistance:** provenance + temporal validity + contradiction graph ستقلل بقاء claims المتعارضة.
3. **H3 - Procedural Transfer:** Knowledge-to-Skill compilation سيرفع transfer على مهام تشترك في البنية.
4. **H4 - Recovery:** failure knowledge وreadback وreplay سترفع معدل recovery بعد أخطاء التنفيذ.
5. **H5 - Context Efficiency:** context compiler سيحقق أداء أعلى عند token budget ثابت من top-k retrieval.
6. **H6 - Epistemic Drift:** ungoverned writeback سيؤدي إلى تدهور أسرع من verification-gated writeback.

18. التصميم التجريبي

نقترح مقارنة أربع عائلات من الأنظمة على missions طويلة متعددة المراحل، مع تغييرات في البيئة وأخطاء أدوات وحقائق متقادمة وإعادة تشغيل runtime.

```
A0 = LLM
A1 = LLM + RAG
A2 = LLM + Memory + Tools
A3 = KB-as-OS
```

- Task success
- Repeated-error rate
- Knowledge reuse
- False-writeback rate
- Contradiction survival time
- Recovery rate
- Context cost
- Verified world-effect success
- Cross-task transfer

19. Ablations

حتى تكون النظرية علمية يجب حذف كل مكون بصورة مستقلة: temporal model، provenance، procedural retrieval، world readback، skill compilation، writeback court، contradiction graph، وcontext compiler. إذا لم يتغير الأداء بعد حذف مكون، تصبح دعوى ضرورته أضعف.

20. Failure Modes

نمط الفشل	الوصف	آلية الحد منه
Epistemic poisoning	حفظ claim خاطئة وإعادة استعمالها	provenance + court + quarantine
Procedural poisoning	تحويل نجاح مصادف إلى skill عامة	replay tests + scoped validity
Stale truth	استخدام معرفة انتهت صلاحيتها	temporal validity + refresh
Context tyranny	هيمنة معرفة قديمة على كل قرار	scheduler + diversity + expiration
Ontology lock-in	إجبار الواقع على ontology خاطئة	schema evolution + competing models
Self-confirmation loop	اعتبار مخرجات الوكيل أدلة مستقلة	source lineage + independence checks
Execution hallucination	اعتبار tool success = world success	readback + external verification

21. حدود النظرية

KB-as-OS لا تحل وحدها general reasoning أو perception أو exploration أو goal generation أو value alignment أو causal discovery أو creativity. كما أنها ليست بديلاً عن النموذج الأساسي. بل هي طبقة تجعل الذكاء الناتج عن النموذج أكثر استمرارية وربطاً بالعالم.

```
Capability = Model × KnowledgeSubstrate × Runtime ×
EnvironmentInteraction
```

إذا اقترب أي عامل من الصفر يمكن أن ينهار أداء النظام الكلي، ولذلك لا ينبغي تقديم KB-as-OS بوصفها تفسيراً وحيداً للAGI.

22. HKOS داخل MOUSSAIF AGI OS

داخل المعمارية المقترحة يمكن تعريف HKOS - HyperKnowledge Operating System بوصفه طبقة تجمع Epistemic Graph وOperational Graph وMemory Graph وEvidence Ledger وContext Compiler وWriteback Governor. ليس HyperKB كتاباً ضخماً، بل شبكة من objects قابلة للتنفيذ والاختبار والتحديث.

```
HKOS = EpistemicGraph + OperationalGraph + MemoryGraph + EvidenceLedger
                + ContextCompiler + WritebackGovernor
```

وتتصل HKOS مباشرة بمحاور MCD/QMCD للأنطولوجيا ومستويات الوصف، HyperSkills للإجراءات، HyperLoops للاستمرارية، World-Effect Loop للفعل، Falsification Court للتحقق وSelf-Reconstruction Memory للاستعادة.

23. برنامج بحث مقترح

1. إنشاء schema موحد لعناصر المعرفة التشغيلية مع type system واضح.
2. بناء benchmark للتناقضات والصلاحية الزمنية وknowledge refresh.
3. اختبار Knowledge-to-Skill compilation على عائلات مهام جديدة.
4. إنشاء long-horizon benchmark يتضمن restarts وأخطاء أدوات وتغيرات خارجية.
5. قياس أثر governed writeback على الانجراف المعرفي عبر آلاف الدورات.
6. اختبار multi-agent promotion policies ومنع contamination بين الوكلاء.
7. فصل مساهمة النموذج الأساسي عن مساهمة الOS عبر نموذج ثابت ومعماريات مختلفة.

24. الخاتمة

الخطوة الحاسمة في الانتقال من assistant إلى وكيل مستمر ليست فقط زيادة parameters أو context. إنها نقل المعرفة من "معلومات متاحة للنموذج" إلى "حالة منظمة تحكم الوكيل". عندئذ تصبح المعرفة قادرة على توجيه الاسترجاع والتخطيط والفعل والتحقق والتعلم وإعادة البناء. بهذا المعنى KB-as-OS مرشحة لأن تكون إحدى substrates الأساسية للأنظمة الوكيلة المتجهة نحو الذكاء العام، بشرط أن تعمل داخل منظومة أوسع من الأهداف والذاكرة ونماذج العالم والمهارات والفعل والتحقق والحوكمة.

```
Know -> Act -> Observe -> Verify -> Rewrite
```

المراجع الأساسية

1. Packer, C. et al. (2023). MemGPT: Towards LLMs as Operating Systems. arXiv:2310.08560.
2. Sumers, T. R. et al. (2023). Cognitive Architectures for Language Agents (CoALA). arXiv:2309.02427.
3. Park, J. S. et al. (2023). Generative Agents: Interactive Simulacra of Human Behavior. arXiv:2304.03442.
4. Shinn, N. et al. (2023). Reflexion: Language Agents with Verbal Reinforcement Learning. arXiv:2303.11366.
5. Yao, S. et al. (2022). ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629.
6. Schick, T. et al. (2023). Toolformer: Language Models Can Teach Themselves to Use Tools. arXiv:2302.04761.
7. Wang, G. et al. (2023). Voyager: An Open-Ended Embodied Agent with Large Language Models. arXiv:2305.16291.
8. Wang, L. et al. (2023). A Survey on Large Language Model based Autonomous Agents. arXiv.
9. Shinn, N. et al. / related search-agent literature (2023-2024): tree search, reflection, environment feedback and long-horizon planning.
10. Microsoft Research (2024). From Local to Global: A Graph RAG Approach to Query-Focused Summarization.

هذه الورقة إطار نظري معماري وبرنامج بحث. المصطلحات المقترحة مثل KB-as-OS وHKOS وExecutable Epistemic Graph تستخدم هنا كصياغة تركيبية قابلة للاختبار، لا كادعاء بوجود إجماع اصطلاحي في الأدبيات.