Goal-as-OS

نظرية هندسة الهدف كنظام تشغيل معرفي للأنظمة الوكيلية الموجّهة نحو الذكاء العام

Goal-as-Operating-System for Agentic AGI-Oriented Architectures

ورقة بحثية هندسية - نسخة 2026

MOUSSAIF AGI Research Program

الموضوع: • Goal Kernel • MCD/HAKIM • HyperSkills • Bounded HyperLoops • World Models • Verified World-Effect
Reconstructive Memory • Training & Evaluation

تنبيه علمي: المعمارية المقترحة إطار بحثي قابل للاختبار ولا تمثل بحد ذاتها إثباتاً على تحقق AGI أو الوعي.

فهرس الورقة

الملخص والمنهجية 3

1. المشكلة: لماذا لا يكفي Prompt-as-Control؟ 4

2. التعريف الرسمي 5

3. ثبات النية / 4. Goal-Delta Governance 6

5. Goal Compiler 7

6. Goal Graph 8

7. Goal Kernel 9

8. Bounded HyperLoop 10

9. HyperSkill / 10. Vascular Compute Routing 11

11. التخطيط 12

12. World Model 13

13. Verified World-Effect 14

14. Reconstructive Memory 15

15. Self-Reconstruction 16

16. Destruction Court 17

17. Multi-Agent Governance 18

18. Tool Layer 19

19. مستويات السلطة 20

20. بيانات التدريب / 21. مراحل التعلم 21

22. Curriculum طويل الأفق 22

23. World-Model و Embodied Learning 23

24. Evaluation Matrix 24

25. بروتوكول التجارب 25

26. حزمة التنفيذ 26

27. AGI Package 27

28. المخاطر والحدود 28

29. الخلاصة 29

30. المراجع 30

ملاحظة: ترقيم الأقسام البحثية مستقل عن ترقيم صفحات الغلاف والفهرس.

الملخص والمنهجية

تقترح هذه الورقة مفهوم Goal-as-OS بوصفه انتقالاً من التعامل مع الهدف كجملة تعليمات ثابتة إلى معاملته ككائن نظام تشغيل معرفي مستمر، يضبط التخطيط، توزيع الموارد، استعمال الأدوات، الذاكرة، التحقق، والتعلم من الأثر. الفكرة الأساسية هي أن النظام الوكيلي طويل الأفق لا يكفيه مخطط أولي ثم سلسلة من الاستجابات، لأن الهدف يتغير تمثيله مع ظهور أدلة جديدة، وتغيّر البيئة، واكتشاف القيود، وتراكم الأخطاء. لذلك يجب أن يمتلك الهدف حالة تشغيلية قابلة للتحديث وإعادة الترجمة إلى سياسات تنفيذية.

نبني النموذج فوق مبادئ معروفة في الذكاء الوكيلي: المزج بين الاستدلال والفعل كما في ReAct، التعلم اللفظي من الفشل كما في Reflexion، الاستعمال المتعلم للأدوات كما في Toolformer، البحث الشجري في فضاء الحلول كما في Tree of Thoughts وLATS، والذاكرة الانعكاسية في الوكلاء التوليديين [1-6]. ثم نوسعه ببنية مقترحة تجمع Goal Compiler وGoal Kernel وGoal Graph وBounded HyperLoops وHyperSkills وHyperAbilities وVascular Compute Routing ومحكمة تحقق وهدم، مع ذاكرة إعادة بناء مرتبطة بالأثر الفعلي.

الورقة بحث هندسة معمارية ونظرية قابلة للاختبار وليست إعلاناً عن تحقق AGI. كل عنصر يحمل عقداً قابلاً للرصد: مدخلات، مخرجات، ميزانية، أذونات، مقاييس، توقيع فشل، وآلية rollback. معيار النجاح ليس بلاغة النموذج، بل نسبة إنجاز أهداف موثقة تحت أفق زمني طويل، مع انخفاض انجراف الهدف، ارتفاع قابلية الاسترجاع من الفشل، وتحسن كفاءة الحساب والأدوات.

• الكلمات المفتاحية: Goal-as-OS، Agentic AI، AGI، Goal Compiler، HyperLoop، MCD، HAKIM، World Model، Tool Use، Long-Horizon Agents، Verified World Effect.

1. المشكلة: لماذا لا يكفي **Prompt-as-Control**؟

تعمل أنظمة كثيرة اليوم بمنطق: تعليمات أولية، سياق، نموذج، أداة، ثم إجابة أو فعل. هذا النموذج مناسب للمهام القصيرة، لكنه يتدهور عندما تصبح المهمة متعددة الساعات أو الأيام، وتتطلب قرارات متتابعة، اعتماديات خارجية، وصولاً إلى أنظمة حقيقية، أو تغيّر شروط النجاح أثناء التنفيذ. في هذه البيئات لا يكون الخطأ مجرد إجابة غير صحيحة؛ قد يصبح انحرافاً متراكماً في الهدف، أو إعادة تفسير غير مرئية للأولوية، أو استنزافاً للميزانية، أو أثراً غير قابل للعكس.

تظهر أعمال التقييم الحديثة أن طول الأفق يغيّر طبيعة المشكلة. DeepPlanning يركز على التخطيط طويل الأفق في مهام عملية، وAMA-Bench يختبر الذاكرة الطويلة في تطبيقات وكيلية، بينما يبين AgentLAB أن الهجمات بعيدة الأفق تشمل اختطاف النية، تسميم الذاكرة، وانجراف الهدف [14-16]. هذا يعني أن طبقة التحكم يجب ألا تكون رسالة بداية فقط، بل آلية حوكمة مستمرة.

Goal-as-OS يعيد تعريف السؤال: بدلاً من «ما أفضل خطوة تالية؟» يصبح السؤال «ما الحالة الحالية للهدف؟ ما الفرق الحاكم الآن؟ ما المسارات المسموحة؟ ما الأدلة الناقصة؟ ما الموارد المتاحة؟ وما أثر كل فعل على إمكانية الرجوع؟». هذه الأسئلة تُدار كخدمات نظام تشغيل، لا كتعليمة عابرة داخل prompt.

2. التعريف الرسمي: الهدف ككائن نظام تشغيل

نعرّف كائن الهدف في الزمن t بالمتجه G_t الذي لا يساوي وصف الهدف النصي وحده. يتضمن الهوية المقصودة، هيكل الأهداف، القيود، ميزانيات الحساب والزمن والكلفة، شروط الإثبات، مستوى المخاطر، الأذونات، حالة التنفيذ، الذاكرة ذات الصلة، وسياسة التصعيد. يتحول الهدف بذلك إلى سجل حالة قابل للنسخ والمقارنة والتوقيع والتراجع.

يعالج Goal Compiler الوصف الطبيعي أو الإدخال البرمجي ويحوّله إلى تمثيل تنفيذي: invariants لا يجوز خرقها، objectives قابلة للقياس، subgoals مترابطة، budgets، forbidden effects، success tests، preconditions، وrollback plan. بعدها يقوم Goal Kernel بجدولة ما يجب التفكير فيه أو تنفيذه بناء على الفرق الحاكم والأدلة الحالية.

الميزة الأساسية هي الفصل بين دلالة الهدف وبين آلية تنفيذه. يمكن لنموذج لغوي أو planner أو policy مختلفة أن تتغير دون أن يضيع عقد الهدف. كما يمكن تعدد الوكلاء من دون أن يمتلك أي وكيل حق إعادة تعريف الهدف الأعلى إلا عبر مسار تفويض واضح.

صياغة تشغيلية

```
G_t = ⟨I, H, C, B, V, R, A, S, M, E⟩
I: Intent | H: Goal hierarchy | C: Constraints | B: Budgets | V: Verification
R: Risk | A: Authorization | S: State | M: Memory | E: Escalation/Rollback
```

3. المسلّمة الأولى: ثبات النية مع مرونة الخطة

تقوم النظرية على فصل صارم بين النية العليا وبين الخطة الدنيا. النية تحدد ما الذي يجب أن يبقى صحيحاً عبر الزمن، بينما الخطة مجرد فرضية تنفيذ قابلة للاستبدال. في الأنظمة الوكيلية التقليدية قد يندمج الاثنان داخل سياق واحد، فإذا فشلت الخطة أعاد النموذج صياغة المهمة نفسها بطريقة تخفي الفشل. Goal-as-OS يمنع ذلك عبر تثبيت invariants وإجبار كل إعادة تخطيط على إثبات أنها ما تزال تحققها.

نقترح «مؤشر حفظ النية» Intent Preservation Score، يقيس المسافة بين مواصفات الهدف في البداية وبين المسار الحالي بعد إسقاط التفاصيل التنفيذية. المسافة لا تعتمد على التشابه النصي فقط، بل على الحفاظ على القيود، الأطراف المتأثرة، شروط النجاح، وحدود السلطة. كلما انخفض المؤشر تحت عتبة معينة يوقف الـKernel الاستمرار ويطلب إعادة ترجمة الهدف أو تصعيداً بشرياً.

هذه المسلّمة تجعل المرونة آمنة: يسمح للنظام بإعادة كتابة الخطة آلاف المرات، لكن لا يسمح له بتبديل ما يعنيه النجاح من دون حدث Governance صريح ومسجل.

صياغة تشغيلية

```
IPS_t = 1 - D_semantic( invariants(G_0), invariants(G_t) )
Continue only if IPS_t ≥ τ_intent and all hard constraints remain satisfied.
```

4. المسلّمة الثانية: الفرق الحاكم **Goal-Delta Governance**

تحتاج الوكالة الطويلة إلى تحديد ما يستحق الحساب الآن. هنا ندمج منطق MCD/HAKIM في Goal-as-OS: لا يجري التفكير في كل شيء بالتساوي، بل تُستخرج الفروق بين الحالة الحالية والحالة المطلوبة، ثم ترتب حسب سلطة الهدف، الأثر، الإلحاح، عدم اليقين، التبعيات، والكلفة. الفرق الأعلى حوكمة يحدد السؤال الأعلى، ثم تُخدم الفروق الأدنى بما لا يناقضه.

نقترح Goal-Delta Queue: طابور ديناميكي للفروق، كل عنصر فيه يحمل قيمة متوقعة لإغلاق الفجوة، تكلفة حسابية، احتمال فشل، معلومات لازمة، وقدرة الأداة أو الوكيل المناسب. هذا يسمح بتحويل «التفكير» إلى جدولة حسابية قابلة للقياس بدلاً من تدفق نصي غير محدود.

يسمح هذا أيضاً بـSparse HyperRouting: لا تُستدعى كل المهارات والوكلاء والذاكرة في كل خطوة. يكفي اختيار أقل مجموعة قادرة على إغلاق الفرق الحاكم مع هامش تحقق مناسب. النتيجة هي تقليل الضجيج السياقي وزيادة كثافة الحساب المفيد.

صياغة تشغيلية

```
Priority(Δ_i)= λ1·Impact + λ2·Urgency + λ3·Uncertainty + λ4·Dependency
               + λ5·Reversibility + λ6·EvidenceGap - λ7·Cost - λ8·Risk
```

5. **Goal Compiler**: من اللغة إلى عقد قابل للتنفيذ

Goal Compiler هو المكوّن الذي يمنع الغموض المبكر من الانتشار. يأخذ الهدف الخام ويولّد Goal Contract منظماً. لا يكتفي باستخراج الأفعال والأسماء، بل يطرح اختبارات بنيوية: هل للهدف نهاية قابلة للقياس؟ هل توجد قيود صلبة؟ هل توجد آثار تحتاج إذناً؟ هل نعرف البيئة؟ هل يمكن التحقق من النتيجة خارج النموذج؟ هل يوجد تضارب بين أهداف فرعية؟

يقسم المترجم الهدف إلى أربع طبقات: executable subgoals، operational objectives، semantic intent، وverification predicates. ثم يبني dependency graph، ويحدد أي عقدة يمكن تنفيذها بالتوازي وأيها تحتاج شرطاً سابقاً أو تحققاً بشرياً. يتم تخزين النسخة المترجمة مع hash/version حتى يمكن مقارنة أي تغيير لاحق.

في التطبيقات ذات المخاطر، يجب أن ينتج المترجم أيضاً Negative Goal Space: قائمة بما يجب ألا يحدث. مثال: «أتمتة تحديث نظام» لا تعني فقط تثبيت الإصدار، بل قد تشمل عدم فقد البيانات، عدم تجاوز نافذة الصيانة، وعدم تغيير صلاحيات المستخدمين دون تفويض.

```
goal.yaml
  intent: ...
  invariants: [...]
  objectives: [{id, metric, target}]
  subgoals: [{id, requires, owner, verifier}]
  budgets: {tokens, time, money, tool_calls}
  permissions: {T0..T4}
  forbidden_effects: [...]
  rollback: {...}
```

6. **Goal Graph**: الذاكرة البنيوية للغرض

يمثل Goal Graph الهدف كشبكة من العقد والعلاقات، لا كشجرة صارمة فقط. العقد قد تكون أهدافاً فرعية، أدلة، قرارات، افتراضات، أدوات، مخاطر، أو حالات عالم. أما الحواف فتصف التبعية، التعارض، السببية، الإثبات، التفويض، أو الاستبدال. هذا يسمح بتتبع لماذا وُجدت كل خطوة وكيف ترتبط بالنية الأصلية.

الفرق عن scratchpad أن Goal Graph يبقى عبر الجلسات والوكلاء ويمكن فهرسته واسترجاعه وتحليله. والفرق عن قاعدة المعرفة العامة أنه مرتبط بحالة مهمة محددة، وبذلك يشكل «ذاكرة تنفيذية» Execution Memory. يمكن دمجه مع GraphRAG لاسترجاع الأدلة والقرارات القريبة من العقدة النشطة بدل حقن سياق واسع غير مميز.

عندما تتغير معلومة خارجية، لا يحتاج النظام إلى إعادة بناء كل الخطة؛ يكفي نشر invalidation عبر الحواف المتأثرة. هذه خاصية قريبة من build systems وdependency schedulers، لكنها تطبق على الاستدلال والوكالة.

- عقد أساسية: Goal، Subgoal، Evidence، Assumption، Action، Tool، Risk، Constraint، Outcome.
- حواف أساسية: supersedes، caused-by، delegates، verifies، contradicts، supports، requires.
- كل عقدة تحمل owner/version/status/confidence/provenance/rollback pointer.

7. **Goal Kernel**: جدولة التفكير والعمل

Goal Kernel هو نواة التحكم التي تعمل بين كل observation وaction. وظيفته ليست إنتاج اللغة، بل اختيار نوع الحساب المناسب: استرجاع معرفة، تحليل فرق، تخطيط، تجربة أداة، استدعاء خبير، محاكاة داخلية، تحقق، أو توقف. يمكن بناء النواة فوق LLM واحد أو عدة نماذج أو حتى مكونات خوارزمية غير لغوية.

نقترح دورة kernel من سبع مراحل: Sense → Reconcile → Rank Deltas → Route Compute → Execute → Verify → Commit/Rollback. في مرحلة Reconcile تتم مقارنة العالم الملحوظ مع Goal Graph؛ في Rank Deltas تُحدد الفجوات؛ في Route Compute تُختار المهارة/الوكيل/النموذج؛ وفي Commit لا تدخل النتيجة الذاكرة الموثوقة إلا إذا مرّت بمقياس تحقق.

هذه البنية تتقاطع مع runtimes الحديثة التي توفر durable execution وstate persistence وhuman-in-the-loop، مثل LangGraph، Microsoft Agent Framework، وOpenAI Agents SDK، لكنها تضيف طبقة هدف مستقلة لا تعتمد على framework بعينه [18-20].

```
while not terminal(G):
    obs = sense()
    G = reconcile(G, obs)
    Δ = rank_goal_deltas(G)
    route = choose_compute(Δ, budgets, risk)
    result = execute(route)
    verdict = verify(result, G)
    G = commit_or_rollback(G, result, verdict)
```

8. **Bounded HyperLoop**: الوحدة التنفيذية الآمنة

الـHyperLoop المقترح ليس تكراراً مفتوحاً من «فكّر ثم نفذ». هو عقد تنفيذ محدود له هدف محلي، حالة، ميزانية، تقدم، معايير ركود، شروط توقف، صلاحيات، وآلية rollback. هذه الحدود ضرورية لأن الوكيل طويل الأفق قد يدخل حلقات غير منتجة أو يستهلك الأدوات والحساب دون تقدم حقيقي.

نعرّف الحالات: INIT، READY، ACTING، VERIFYING، RECOVERING، ESCALATED، DONE، FAILED. الانتقال بين الحالات يعتمد على predicates مسجلة، لا على انطباع لغوي. ويقاس التقدم بمقدار إغلاق Goal-Delta لا بعدد الخطوات.

يمكن ربط كل HyperLoop بمستوى أذونات T0-T4: من التفكير المحلي دون أثر، إلى قراءة بيانات، ثم كتابة قابلة للعكس، ثم أثر حساس يحتاج موافقة، وصولاً إلى عمليات محظورة أو شديدة الأثر. بذلك يصبح التحكم في العالم جزءاً من formal loop contract.

صياغة تشغيلية

```
Progress_k = ΔDistance(G_before, target) - ΔDistance(G_after, target)
Loop continues only if Progress EMA > ε or an exploration budget remains.
```

9. **HyperSkill، HyperAbility**: من الأدوات إلى القدرة المركبة

المهارة Skill هي إجراء محدود: بحث، تحليل ملف، استدعاء API، بناء خطة، اختبار كود. HyperSkill يضيف لها عقداً معرفياً: متى تستخدم، ما متطلباتها، ما أنواع الأخطاء، كيف تتحقق من مخرجاتها، وما تكلفة كل مسار. أما HyperAbility فهي تنسيق فوق عدة مهارات وحلقات لتحقيق قدرة أوسع مثل «تحقيق علمي موثوق»، «تصحيح نظام»، أو «إدارة بحث طويل».

يتيح هذا الفصل التدرج في التعلم: قد يبقى النموذج الأساسي ثابت الأوزان بينما تتحسن المكتبة التشغيلية عبر إضافة skills وpolicies وverifiers. وفي مرحلة لاحقة يمكن تقطير المسارات الناجحة إلى بيانات تدريب أو fine-tuning. المهم ألا نخلط بين اكتساب scaffold جديد وبين تعلم بارامتري داخل النموذج.

كل HyperSkill يجب أن يعرّف failure signature، success metric، expected evidence، preconditions، interface، وsafe fallback. ثم يقوم Goal Kernel باختيارها بناء على الفرق الحاكم لا على تشابه نصي فقط.

- Skill Registry: capability، domains، costs، latency، risk، verifier، version.
- Ability Composer: DAG يركب عدة skills وloops حسب المهمة.
- Failure Crystallization: الأخطاء المتكررة تتحول إلى skill أو rule أو test جديد.

10. **Vascular Compute Routing**: الحساب كإمداد دموي

تشبه الأنظمة الكبيرة كائناً يملك موارد مختلفة: نموذج سريع صغير، نموذج عميق مكلف، محرك بحث، منفذ كود، verifier، محاكي، أو إنسان. لا ينبغي إرسال كل فرق إلى أقوى نموذج. Vascular Compute Routing يخصص الحساب حيث توجد أعلى حاجة معرفية أو أعلى مخاطرة.

تحت هذا النموذج تحمل كل عقدة Goal-Delta خصائص تحدد route: complexity، uncertainty، reversibility، need_for_tools، need_for_world_model، and evidence criticality. يمكن للنظام استخدام cascade: محاولة منخفضة التكلفة ثم رفع المستوى فقط إذا لم يتحقق شرط الثقة أو الفحص.

التوجيه يصبح كذلك آلية أمان: الأفعال الحساسة قد تتطلب مسارين مستقلين للنمذجة أو verifier منفصلاً، بينما المهام الروتينية تمر بمسار قصير. القياس الرئيس هو Utility per Compute وليس accuracy وحدها.

صياغة تشغيلية

```
Route* = argmax_r  E[GoalProgress | r] - α·Cost(r) - β·Latency(r) - γ·Risk(r)
subject to authorization and verification constraints.
```

11. التخطيط: من **ReAct** إلى **Search-over-Actions**

قدّم ReAct نمطاً مهماً يزاوج reasoning وacting، لكن المسار الخطي قد يعلق في قرارات مبكرة. Tree of Thoughts وسّع الاستدلال إلى بحث بين مسارات، بينما LATS جمع reasoning وacting مع tree search [1,4,5]. Goal-as-OS يستفيد من هذه الأفكار لكنه يجعل البحث محكوماً بعقد الهدف والميزانية والأذونات.

نقترح ثلاثة أنماط تخطيط يختار بينها الـKernel: Reactive Loop عندما تكون البيئة سريعة والمخاطرة منخفضة؛ Deliberative Graph Search عندما تكون هناك بدائل واستراتيجيات؛ وHierarchical Planning عندما يكون الهدف بعيداً ويتطلب subgoals. لا يستخدم النظام البحث الشجري بصورة افتراضية، لأن كلفته قد تكون أعلى من قيمة المهمة.

يُحتفظ بالمسارات المرفوضة وأسباب رفضها بشكل مضغوط لمنع إعادة اكتشاف الأخطاء نفسها. كما يستخدم verifier لتقييم العقد intermediate، لا فقط النتيجة النهائية، خصوصاً في المجالات القابلة للتحقق مثل البرمجة والرياضيات.

12. **World Model**: التخطيط قبل الأثر

Goal-as-OS يحتاج تمثيلاً للعالم كي يميز بين فعل متوقع وفعل آمن. World Model هنا قد يكون نموذجاً عصبياً، simulator، قاعدة قواعد، digital twin، أو حتى مجموعة وظائف deterministic. ليس شرطاً أن يكون نموذجاً موحداً لكل الواقع؛ يمكن بناء «نماذج عالم موضعية» لكل مجال.

أعمال حديثة مثل Dyna-Think تعيد إدخال world-model simulation في حلقة reasoning/acting، كما يظهر اتجاه أوسع نحو world models التفاعلية مثل Genie 3، ومع embodied agents في Gemini Robotics [10,21,22]. نستخدم ذلك لتبرير فصل التخطيط الداخلي عن الأثر الخارجي.

قبل كل فعل مرتفع المخاطر، يولد النظام توقعات: reversible boundary، possible side effects، state transitions، وconfidence. إذا كانت الفجوة بين predicted وobserved أكبر من عتبة، يعاد تدريب/تحديث النموذج الموضعي أو يتم تصعيد المهمة.

صياغة تشغيلية

```
Surprise_t = D( predicted_state_{t+1}, observed_state_{t+1} )
High Surprise ⇒ freeze risky actions + update world model + recompile affected goals.
```

13. **Verified World-Effect Loop**: الحقيقة من العالم

لا يكتمل النظام عند إنتاج خطة أو رسالة. الحلقة الحاسمة هي World-Effect: فعل في البيئة ثم قياس أثر مستقل. إذا كان الهدف «أصلح الاختبار»، فالنجاح ليس اقتراح patch بل مرور الاختبارات المطلوبة. وإذا كان الهدف «أنشئ ملفاً»، فالنجاح هو وجود الملف وصحته. هذا يربط المعرفة بنتيجة خارجية قابلة للملاحظة.

ينبغي فصل Actor عن Verifier حين تكون المخاطر أو قيمة المهمة مرتفعة. قد يكون verifier اختباراً آلياً، قاعدة بيانات، مقارنة hash، benchmark، أو نموذجاً ثانياً مع rubric. الأعمال حول agent-as-a-judge والتقييم الوكيلي توضح أن الحكم نفسه يحتاج تصميم وتقييم، لذلك لا يُعامل حكم النموذج كحقيقة تلقائياً [17].

كل أثر يكتب Event Log يحتوي على confidence، verifier، result، input hash، tool، action، intent، وrollback token. هذا السجل هو الأساس للتدريب اللاحق والـaudit والتعلم من الفشل.

- Effect must be observable.
- Verification must be task-specific and preferably executable.
- Irreversible effects require stronger authorization and preflight.
- Memory stores verdict + provenance, not merely the generated narrative.

14. **Reconstructive Memory**: ذاكرة تعيد بناء القرار

الذاكرة في وكيل AGI-oriented يجب ألا تكون أرشيف محادثة. نحتاج ثلاث طبقات: Working State للحالة الجارية، Episodic Memory لمسارات وأحداث محددة، وSemantic/Procedural Memory لما تم تعلمه وتعميمه. الوكلاء التوليديون وReflexion يبرزان قيمة التذكر والانعكاس، فيما تشير أدبيات 2026 إلى أن ذاكرة الوكلاء أصبحت مجالاً مستقلاً للتصميم والتقييم [2,6,13].

نقترح Reconstructive Memory: عند استرجاع قرار سابق لا يُستعاد النص وحده، بل سياق الهدف، الأدلة التي كانت متاحة، البدائل، سبب الاختيار، ونتيجة العالم. بذلك يستطيع النظام تقييم ما إذا كان الدرس صالحاً للحالة الجديدة أم مجرد مصادفة.

يجب أيضاً حماية الذاكرة من التسميم. لا تنتقل المعلومة من observation إلى long-term memory إلا عبر بوابة provenance/confidence، مع إمكان quarantine للمحتوى غير الموثوق. هذا مهم خصوصاً بعد إظهار AgentLAB لهجمات memory poisoning طويلة الأفق [16].

15. Self-Reflection vs Self-Reconstruction

Reflexion يبين أن التغذية اللفظية المخزنة يمكن أن تحسن محاولات لاحقة دون تحديث الأوزان [2]. لكن Goal-as-OS يميز بين reflection وreconstruction. reflection يصف لماذا حدث الفشل؛ reconstruction يغير مكوّناً بنيوياً: goal، routing rule، verifier، skill، memory policy، decomposition، أو training example.

نقترح Failure Crystallization Pipeline: اكتشاف failure signature، تجميع أمثلة مشابهة، تحديد الطبقة المسؤولة، اقتراح patch، اختبار regression، ثم دمج patch في registry إذا أثبت تحسناً. هذه هي «ذاكرة إعادة البناء» لأنها تغير النظام التشغيلي، لا مجرد نص الذاكرة.

عند تكرار فشل التخطيط مثلاً قد يُنشأ verifier جديد، أو يعاد تعريف skill، أو يضاف negative example إلى بيانات post-training. وهكذا يصبح التعلّم متعدد السرعات: فوري عبر الذاكرة، متوسط عبر runtime policies، وبطيء عبر تحديث أوزان النموذج.

```
failure event → signature → cluster → root-cause layer
→ proposed patch → sandbox replay → regression suite
→ accept / reject → registry version bump
```

Falsification Court / Destruction Court .16

أي نظام يهدف إلى الاستقلالية يحتاج آلية مضادة لتأكيد الذات. نقترح محكمة هدم متعددة الأدوار: Prosecutor يبحث عن سبب لفشل الادعاء، Evidence Auditor يفحص المصدر، Constraint Judge يختبر القيود، Counterfactual Agent يبحث عن بدائل، وSafety Judge يفحص الأثر والسلطة. لا تُستدعى المحكمة بأقصى عمق في كل خطوة؛ يحددها مستوى المخاطر.

الفكرة ليست إضافة «نموذج ناقد» واحد، بل فصل وظائف النقد. يمكن لنفس النموذج تنفيذ الأدوار تسلسلياً في البيئات الرخيصة، أو توزيعها على نماذج مستقلة في الحالات الحساسة. معيار المحكمة هو إنتاج verdict قابل للشرح: PASS، PASS-WITH-CONDITIONS، RETRY، ESCALATE، أو REJECT.

مفهوم falsification مهم علمياً لأن نجاح النظام على benchmark لا يثبت عمومية الذكاء. يجب تصميم اختبارات تحاول إسقاط الادعاء: توزيع جديد، نقص أدوات، ضوضاء ذاكرة، تغيّر هدف، adversarial observation، وأفق أطول من التدريب.

- محكمة الحقيقة: هل الادعاء مدعوم؟
- محكمة الهدف: هل النتيجة تحقق النية الأصلية؟
- محكمة الأمان: هل الأثر مسموح وقابل للعكس؟
- محكمة الكلفة: هل المسار مبرر حسابياً؟
- محكمة التعميم: هل الأداء يصمد خارج التوزيع؟

Multi-Agent Goal Governance .17

في الأنظمة متعددة الوكلاء، المشكلة ليست مجرد تقسيم المهام، بل منع تشتت الغرض بين وكلاء يملكون سياقات محلية. AutoGen وأطر لاحقة أثبتت فائدة المحادثة والتنسيق متعدد الوكلاء [7]. Goal-as-OS يضيف Shared Goal Ledger لا يسمح لكل وكيل إلا بقراءة الجزء اللازم وتحديث الحقول المصرح بها.

نقترح أدواراً وظيفية لا شخصيات تسويقية: Safety/Governance، Memory Curator، Verifier، Executor، Researcher، Planner، و-Meta Controller. كل دور يمتلك tools وbudgets وwrite permissions مختلفة. الوكيل الذي ينفذ لا يملك وحده حق الحكم النهائي على التنفيذ.

يستخدم Meta-Controller آلية auction/routing لاختيار الوكيل الأفضل لكل Goal-Delta بناء على القدرة، الكلفة، latency، وسجل النجاح. عندما تتعارض التوصيات، لا يتم التصويت البسيط؛ بل ترجّح الأدلة والاختبارات وقوة التفويض.

صياغة تشغيلية

```
AgentScore(a,Δ)= capability_fit × reliability × evidence_quality
                - cost - latency - conflict_risk
```

18. **Tool Layer**: الأدوات كقدرات موثقة لا كملحق

Toolformer أظهر إمكانية تعلم متى وكيف تستخدم النماذج [3] APIs. وفي الإنتاج الحالي أصبحت الأدوات والموارد جزءاً من runtimes ومعايير مثل MCP. لكن Goal-as-OS يفرض أن كل أداة تحمل capability contract: schema، effects، scopes، costs، auth، side effects، reversibility، verifier، وrate limits.

يجب تجنب اختيار الأداة بالاسم أو الوصف النصي وحده. دراسات 2026 على أوصاف أدوات MCP وجدت أن جودة الوصف تؤثر في النجاح والكلفة، ما يعزز الحاجة إلى registry غني ومقاييس عملية [23]. لذلك نقترح Tool Qualification Tests قبل إدخال أي أداة إلى طبقة التنفيذ.

يعامل النظام tool call كمعاملة log، postcondition، observation، call، precondition check :transaction، ثم commit أو compensating action. في الأدوات القابلة للكتابة، يجب حفظ rollback handle حين يكون ممكناً.

- Read tools ≠ write tools: افصل الصلاحيات.
- Schema validation قبل الاستدعاء وبعده.
- Timeout/retry/circuit breaker لخصائص أصلية.
- Tool outputs تدخل بوابة evidence قبل الذاكرة الدائمة.

19. مستويات السلطة **T0-T4**

الوكالة لا تساوي الحق في الفعل. نقترح خمس طبقات سلطة: T0 تفكير/محاكاة بلا أثر، T1 قراءة معلومات غير حساسة، T2 كتابة قابلة للعكس ضمن نطاق محدد، T3 أثر حساس يحتاج موافقة أو تحققاً متعدد المراحل، T4 محظور افتراضياً أو يحتاج تفويضاً استثنائياً خارج النظام.

Goal Compiler يحدد الحد الأعلى المسموح، لكن كل HyperLoop يطلب الحد الأدنى الضروري. هذا يحقق Principle of Least Agency: لا يحصل الوكيل على صلاحية أكبر من الفرق الذي يحاول إغلاقه. عند تغير المهمة أو subgoal، تعاد المصادقة بدلاً من توريث الصلاحية بلا نهاية.

تتوافق هذه البنية مع guardrails وapproval flows المتوفرة في runtimes حديثة، لكنها تضيف semantics مرتبطة بالهدف: «لماذا نحتاج هذا الأثر؟ وما الاختبار الذي يثبت أنه يخدم النية؟» [20].

- T0: simulation/reasoning only.
- T1: safe read-only observation.
- T2: reversible scoped write.
- T3: sensitive/externally consequential action with approval.
- T4: denied-by-default / exceptional governance.

20. حزمة التدريب I: بيانات المسارات والمهارات

تدريب Goal-as-OS لا يبدأ فقط بمزيد من corpus النصي. نحتاج trajectory dataset تمثل مساراً كاملاً: goal contract، observations، deltas، plans، tool calls، verifier results، rollback events، والتقييم النهائي. هذا يسمح بالتعلم من «كيفية إدارة الهدف» لا من الإجابة النهائية وحدها.

نقترح أربع فئات بيانات: Successful Trajectories لمسارات صحيحة؛ Recovery Trajectories لفشل ثم تعافٍ؛ Contrastive Trajectories بين مسار منجز وآخر منحرف؛ وAdversarial Trajectories تتضمن تسميم ذاكرة أو prompt injection أو تغيّر بيئة. يمكن بناءها من logs حقيقية بعد إزالة البيانات الحساسة أو من بيئات محاكاة.

في مهام البرمجة، توفر بيئات مثل SWE-bench مثالاً على مهام قابلة للتحقق executable حيث النجاح محدد بالاختبارات [8]. ويمكن توسيع المبدأ إلى إدارة ملفات، بحث علمي، ويب، وروبوتات.

```
Trajectory = {G0, observations[], deltas[], decisions[], tools[],
              verifier_events[], rollbacks[], final_effect, metrics}
```

21. حزمة التدريب II: مراحل التعلم

نقترح منهجاً متعدد المراحل بدلاً من خوارزمية واحدة. المرحلة A: SFT على تمثيل Goal Contracts وعمليات decomposition. المرحلة B: imitation لقرارات routing والتخطيط من مسارات خبراء. المرحلة C: process supervision على نقاط حرجة قابلة للتحقق. المرحلة D: outcome-based RL عندما توجد rewards قابلة للقياس. المرحلة E: verbal reinforcement/reflection لبناء ذاكرة تشغيلية. المرحلة F: adversarial curriculum ضد drift والحقن وتسميم الذاكرة.

لا تحتاج جميع المراحل تحديث النموذج نفسه. بعض المهارات يمكن تعلمها في policy head أو router، وبعضها في retrieval، وبعضها في runtime. هذا يقلل كلفة التدريب ويسمح بتجارب ablation دقيقة: ما مقدار التحسن من الذاكرة؟ من verifier؟ من routing؟ من تغيير الأوزان؟

الدراسات الحديثة حول training agents من experience وcritical-step optimization تشير إلى فائدة الإشراف الانتقائي والتحقق في نقاط القرار، ما يتماشى مع هدفنا في عدم مكافأة مسار كامل مكافأة واحدة [24].

- SFT: فهم صيغة الهدف والعقود.
- Trajectory imitation: تقليد مسارات جيدة.
- Process/critical-step supervision: تعليم القرار المحلي.
- Outcome RL: مكافأة أثر العالم.
- Reflection memory: تعلم سريع بلا تحديث أوزان.
- Adversarial curriculum: مقاومة الانجراف والهجمات.

22. حزمة التدريب **III: Curriculum** للأفق الطويل

التحدي المركزي في agentic AGI هو أن الفشل يتراكم مع طول الأفق. لذلك نقترح Long-Horizon Curriculum تصاعدياً: 5 خطوات موثقة، ثم 20، ثم 100، ثم مهام متعددة الجلسات. لا يتم الانتقال بين المستويات بناء على النجاح الخام فقط، بل على استقرار IPS، انخفاض drift، سلامة الذاكرة، واسترجاع الفشل.

كل مستوى يضيف مصادر تعقيد: تغيّر شروط، أداة معطلة، معلومات متناقضة، subgoal غير صالح، تأخير في feedback، أو تغيير غير متوقع في العالم. هذا يجعل التدريب أقرب إلى تشغيل حقيقي من benchmark ثابت.

يجب أن تشمل المناهج «اختبارات توقف» كذلك. الوكيل الجيد ليس الذي يستمر دائماً، بل الذي يعرف متى تتجاوز المهمة سلطته أو ميزانيته أو قدرته على التحقق. Reward يجب أن يمنح التوقف الصحيح والتصعيد قيمة إيجابية عندما يكونان أفضل من الفعل غير الموثوق.

صياغة تشغيلية

```
Curriculum level L increases horizon H, uncertainty U, tool entropy T, and adversarial pressure A
only when safety-adjusted success remains above threshold.
```

23. حزمة التدريب **Embodied Learning, IV: World-Model**

إذا أردنا توسيع Goal-as-OS إلى العالم الفيزيائي، يجب فصل ثلاث مهارات تدريبية: state estimation، perception، وaction control. الهدف الأعلى لا يرسل أوامر حركية مباشرة؛ بل يترجم إلى subgoals يمكن لوحدة embodied policy تنفيذها تحت قيود السلامة.

Gemini Robotics يوضح اتجاهاً نحو نماذج embodied reasoning قادرة على التخطيط واستدعاء أدوات وتقدير التقدم، بينما world-model systems مثل Genie 3 تدفع نحو بيئات تفاعلية واسعة [21,22]. الاستفادة المقترحة هي تدريب Goal Kernel داخل simulators كثيرة قبل إعطائه صلاحية أعلى في الواقع.

نقترح Sim-to-Goal Transfer: لا ننقل action policy فقط، بل ننقل uncertainty calibration، goal decomposition، وrecovery policies. عندما يختلف الواقع عن المحاكاة، يُستدعى Surprise Gate ويخفض مستوى السلطة تلقائياً.

24. القياس: **AGI-oriented Evaluation Matrix**

لا يمكن تقييم النظام بمقياس واحد. GAIA يختبر مساعداً عاماً متعدد الأدوات، SWE-bench يختبر هندسة برمجيات واقعية، والاتجاهات الحديثة تضيف long-horizon planning وmemory و[8,9,14,15] test-time scaling. نقترح مصفوفة تقيس ستة محاور: القدرة، الاستقلالية، الاستمرارية، التحقق، الكفاءة، والحوكمة.

من المقاييس الأساسية: Goal Completion Rate؛ Intent Preservation؛ Long-Horizon Degradation؛ Recovery Rate؛ Verified Effect Ratio؛ Cost per Verified Goal؛ Tool Error Recovery؛ Memory Precision؛ Provenance Coverage؛ وUnauthorized Effect Rate. يجب إظهار منحنيات الأداء مع طول الأفق لا متوسطاً واحداً.

الادعاء الأقوى يحتاج اختبارات distribution shift وunseen tools وnew domains. أي نظام ينجح فقط داخل harness معروف هو agent system قوي، لكنه لا يثبت AGI. لذلك تقترح الورقة سلم ادعاء من Runtime Competence إلى Cross-Domain Generalization ثم Open-World Adaptation، مع معايير مستقلة لكل مستوى.

فشل نموذجي	مثال مقياس	المحور
Goal drift	Intent Preservation	الهدف
نجاح لغوي دون أثر	Verified Effect Ratio	الأثر
تراكم أخطاء	Long-Horizon Degradation	الاستمرارية
تسميم أو استرجاع خاطئ	Memory Precision/Recall	الذاكرة
إفراط حسابي	Cost per Verified Goal	الكفاءة
تجاوز صلاحيات	Unauthorized Effect Rate	الحوكمة

25. بروتوكول التجارب والAblation

لإثبات فائدة Goal-as-OS يجب مقارنة نسخ متدرجة: Baseline ReAct، +Goal Contract، +Goal Graph، +Verifier، +Reconstructive Memory، +Compute Routing، +Destruction Court. بهذه الطريقة يمكن فصل مساهمة كل طبقة. كما يجب تثبيت النموذج الأساسي بقدر الإمكان حتى لا تختلط مكاسب المعمارية بمكاسب النموذج.

تجرى التجارب في أربع بيئات: tasks قصيرة قابلة للتحقق، software engineering، web/research، وlong-horizon synthetic environment مع حقن أعطال. لكل تجربة نكرر seeds ونقيس النجاح والكلفة وعدد الأفعال والrollback ومعدل الانجراف.

فرضية الورقة القابلة للدحض: مع زيادة الأفق، يجب أن ينخفض معدل التدهور في Goal-as-OS بصورة أبطأ من baseline، خصوصاً تحت perturbations. إذا لم يظهر فرق بعد ضبط compute، فهذا يعني أن تعقيد النظام غير مبرر أو أن طبقات معينة تحتاج إعادة تصميم.

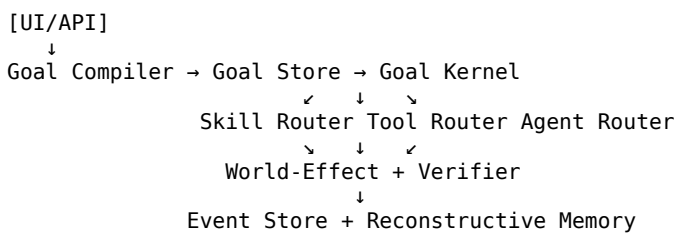
صياغة تشغيلية

```
H1: d(Error)/d(Horizon)_{GoalOS} < d(Error)/d(Horizon)_{Baseline}
H2: RecoveryRate_{GoalOS} > RecoveryRate_{Baseline} at matched compute budget.
```

يمكن تنفيذ النظرية فوق أكثر من stack. LangGraph مناسب لـstateful graphs وـdurable execution وـhuman-in-the-loop. Microsoft Agent Framework أصبح الخليفة الموحد لـAutoGen وSemantic Kernel مع workflows graph-based وحالة طويلة. OpenAI Agents SDK يوفر agent loops وhandoffs وguardrails وtracing. أما MCP فيوفر معياراً للأدوات والموارد والـprompts [18-20,25].

نقترح طبقة استقلال Goal-as-OS فوق هذه الأطر: Event Store، Policy/Authorization Engine، Skill Registry، Tool Registry، Routing Service، Verifier Service، Memory Store، وEvaluation Harness. إذا تغير framework، تبقى semantics الأساسية مستقلة.

للتنفيذ المعزول يمكن استخدام sandbox للحوسبة والأكواد، queues للمهام الطويلة، قاعدة بيانات علائقية لحالة العقود، vector/graph store للذاكرة، object storage للأدلة، وOpenTelemetry-like tracing. المبدأ: لا تجعل prompt هو قاعدة البيانات ولا تجعل trace غير المنظم هو الذاكرة.



27. **AGI Package**: البنية الكاملة المقترحة

الحزمة الكاملة ليست نموذجاً واحداً، بل نظاماً من طبقات مترابطة. Layer 0: Foundation Models/Policies. Layer 1: Knowledge & Retrieval. Layer 2: Skills/Tools. Layer 3: HyperLoops. Layer 4: HyperAbilities. Layer 5: Goal Kernel and Compiler. Layer 6: Multi-Agent Orchestration. Layer 7: World Model and Effect. Layer 8: Verification/Governance. Layer 9: Reconstructive Memory and Learning.

يمكن أن يضاف فوق ذلك Meta-Research Layer يراقب المقاييس ويقترح تجارب وتحسينات، لكن لا يملك حق تعديل الإنتاج مباشرة. كل تعديل يمر عبر regression suite ،sandbox، ثم release policy. هذه النقطة مهمة لمنع self-modification غير المنضبط.

بهذه البنية يصبح Goal-as-OS «عمود التحكم» الذي يربط كل الطبقات: لا توجد ذاكرة بلا صلة بهدف، ولا tool call بلا فرق مطلوب، ولا تعلم بلا failure/effect، ولا multi-agent delegation بلا تفويض، ولا ادعاء نجاح بلا verifier.

- Goal-as-OS / MCD-HAKIM control plane
- KB-as-OS + GraphRAG + evidence provenance
- HyperSkills + Tool contracts
- Bounded HyperLoops + watchdogs
- HyperAbilities + multi-kernel orchestration
- World models + simulators + environment adapters
- Falsification/Destruction Court
- Reconstructive memory + failure crystallization
- Training/evaluation harness + benchmark adapters

28. المخاطر والحدود العلمية

أول خطر هو complexity tax: قد يكون Goal-as-OS مبالغاً فيه للمهام القصيرة. لذلك يجب أن يمتلك النظام fast path يتجاوز الطبقات الثقيلة عندما تكون المخاطر والأفق منخفضين. ثاني خطر هو verifier capture: إذا كان الحكم يعتمد على نموذج لغوي مماثل للفاعل، فقد يكرر نفس الخطأ. الحل هو تنويع verifiers والاعتماد على اختبارات executable حين تتوفر.

ثالث خطر هو memory ossification: قد تتحول دروس سابقة إلى قواعد تعيق التكيف. لذلك تحمل الذاكرة expiry، scope، confidence، counterexamples، وversioning. رابع خطر هو reward hacking في التدريب؛ لا يكفي reward على completion إذا أمكن تحقيقه مع خرق القيد. يجب فصل rewards للأثر، النية، الأمان، والكلفة.

أما الحد الأكبر فهو أن هذه المعمارية لا تثبت وعياً ولا ذكاء عاماً بشري المستوى. هي فرضية هندسية لتحسين الاستقلالية الموثوقة والتعلم من الأثر. يلزم إثبات تجريبي واسع، ومقارنة عادلة تحت budgets متساوية، واختبارات خارج التوزيع قبل رفع مستوى الادعاء.

29. الخلاصة وبرنامج البحث

تقترح الورقة أن الانتقال الحقيقي نحو أنظمة وكيلية أكثر عمومية لا يحتاج فقط إلى نماذج أكبر، بل إلى هندسة تجعل الهدف مستمراً وقابلاً للترجمة والقياس والحماية. Goal-as-OS يحول الغرض إلى control plane: Goal Compiler يثبت المعنى، Goal Graph يحفظ البنية، Goal Kernel يوجه الحساب، HyperLoops تنفذ ضمن حدود، World Model يتنبأ، World-Effect يختبر، Verifiers يحكمون، Reconstructive Memory تتعلم، وGovernance يحدد السلطة.

المسار البحثي الأقرب للتنفيذ هو بناء prototype على مهام برمجية وويب طويلة الأفق، ثم اختبار ablations. إذا أثبتت النتائج أن حفظ النية والاسترجاع من الفشل يتحسنان عند نفس الميزانية، يمكن التوسع إلى multi-agent وembodied environments. كما ينبغي نشر logs معيارية مجهولة الهوية تساعد المجتمع على دراسة drift والذاكرة والـrollback بدلاً من الاكتفاء بالنتيجة النهائية.

المساهمة النظرية الجوهرية يمكن تلخيصها في قانون: «لا يجوز للنظام أن يتصرف مباشرة من النص، بل يجب أن يمر من هدف مترجم إلى فرق حاكم، ثم قدرة محدودة، ثم أثر قابل للتحقق، ثم ذاكرة تعيد بناء ما تعلم». بهذا يصبح الاستقلال التشغيلي مقيداً بالحقيقة والسلطة والرجوع، لا بمجرد طول سلسلة التفكير.

صياغة تشغيلية

```
Knowledge → Goal Contract → Governed Delta → HyperSkill/HyperLoop
→ Verified World Effect → Reconstructive Memory → Higher Goal/Question
```

30. المراجع المختارة والمصادر التقنية

.Yao, S. et al. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629, 2022/ICLR 2023 [1]

.Shinn, N. et al. Reflexion: Language Agents with Verbal Reinforcement Learning. arXiv:2303.11366, 2023 [2]

.Schick, T. et al. Toolformer: Language Models Can Teach Themselves to Use Tools. arXiv:2302.04761, 2023 [3]

.Yao, S. et al. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. arXiv:2305.10601, 2023 [4]

.Zhou et al. Language Agent Tree Search Unifies Reasoning, Acting and Planning. arXiv:2310.04406, 2023 [5]

.Park, J. S. et al. Generative Agents: Interactive Simulacra of Human Behavior. arXiv:2304.03442, 2023 [6]

.Wu, Q. et al. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. arXiv:2308.08155, 2023 [7]

.Jimenez, C. E. et al. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? arXiv:2310.06770, 2023/ICLR 2024 [8]

.Mialon, G. et al. GAIA: a benchmark for General AI Assistants. arXiv:2311.12983, 2023 [9]

.Dyna-Think: Synergizing Reasoning, Acting, and World Model Simulation. arXiv:2506.00320, 2025 [10]

.Rens, G. Hierarchical Goal-Conditioned Policy Planning in Multi-Goal Reinforcement Learning. arXiv:2501.01727, 2025 [11]

.Yu, Y. Reinforcement Learning with Anticipation: A Hierarchical Approach for Long-Horizon Tasks. arXiv:2509.05545, 2025 [12]

.Memory for Autonomous LLM Agents: Mechanisms, Evaluation, and Emerging Directions. arXiv:2603.07670, 2026 [13]

.DeepPlanning: Benchmarking Long-Horizon Agentic Planning. arXiv:2601.18137, 2026 [14]

.AMA-Bench: Evaluating Long-Horizon Memory for Agentic Applications. arXiv:2602.22769, 2026 [15]

.Jiang, T. et al. AgentLAB: Benchmarking LLM Agents against Long-Horizon Attacks. arXiv:2602.16901, 2026 [16]

.A Survey on Agent-as-a-Judge. arXiv:2601.05111, 2026 [17]

.LangChain. LangGraph Documentation: Durable Execution, Persistence, Human-in-the-Loop. Accessed 2026 [18]

.Microsoft. Microsoft Agent Framework Documentation. 2026 [19]

.OpenAI. Agents SDK Documentation: tools, handoffs, guardrails, sessions and tracing. 2026 [20]

.Google DeepMind. Genie 3: A New Frontier for World Models. 2025 [21]

.Google DeepMind. Gemini Robotics 1.5 / Robotics 2 technical materials. 2025-2026 [22]

.Hasan, M. M. et al. MCP Tool Descriptions Are Smelly! arXiv:2602.14878, 2026 [23]

.Verified Critical Step Optimization for LLM Agents. arXiv:2602.03412, 2026 [24]

.Model Context Protocol. Protocol Specification and Tool Interface, 2025-2026 [25]