الركائز التشغيلية المتوازية للأنظمة الوكيلة المتجهة نحو الذكاء العام

Parallel Operating Substrates for AGI-Oriented Agentic Systems

---

ورقة نظرية معمارية - إصدار بحثي موسع

September 2026

A Poly-OS Theory for MOUSSAIF AGI

خريطة الورقة

1. من Knowledge-as-OS إلى Poly-OS
2. تعريف Operating Substrate
3. Constitution-as-OS
4. Goal-as-OS
5. Active Difference
6. Memory-as-OS
7. Context-as-OS
8. Ontology-as-OS
9. World-Model-as-OS
10. Reasoning-as-OS
11. Planning/Search-as-OS
12. Skill-as-OS
13. Ability-as-OS
14. Loop-as-OS
15. Agent-as-OS
16. Federation-as-OS
17. Compute-as-OS
18. Tool-as-OS
19. Action-as-OS
20. Verification-as-OS
21. Authorization/Risk-as-OS
22. Learning/Writeback-as-OS
23. Evaluation-as-OS
24. Observability/Replay-as-OS
25. Deployment-as-OS
26. Continuity-as-OS
27. Identity/Self-Reconstruction
28. Consistency Bus
29. Cognitive Transactions
30. Interrupt Bus
31. AGI Kernel
32. Generality and Open-Endedness
33. فرضيات Poly-OS
34. Ablation Matrix
35. MUG/MUT-AGI
36. MOUSSAIF AGI Poly-OS
37. الخاتمة

الملخص

إذا كانت Knowledge-as-OS تحول المعرفة إلى substrate تشغيلية، فإنها لا تكفي وحدها. هذه الورقة تقترح أن الوكيل المتجه نحو الذكاء العام يحتاج إلى مجموعة من الركائز التشغيلية المتوازية التي تدير أبعاداً مستمرة من وجوده: القواعد العليا، الأهداف، المعرفة، الذاكرة، السياق، الأنطولوجيا، نموذج العالم، الاستدلال، التخطيط، المهارات، الأدوات، الأفعال، التحقق، السلطة، المخاطر، الحوسبة، التعلم، الرصد، النشر والاستمرارية والهوية.

نسمي هذا التصور Poly-OS Agentic Architecture. لا نعني وجود عشرات أنظمة التشغيل البرمجية المنفصلة، بل أن كل بعد يصبح OS-like عندما يملك state وaddressing وscheduling وpolicies وpersistence وinterrupts وinterfaces. وتصبح وحدة الذكاء الحقيقية هي انتظام هذه الحالات وتماسكها عبر الزمن، لا النموذج وحده.

1. من Knowledge-as-OS إلى Poly-OS

وكيل يملك معرفة كاملة بلا هدف لا يعرف ماذا يفعل. وكيل يملك هدفاً ومعرفة بلا ذاكرة يعيد اكتشاف العالم كل مرة. ووكيل يملك ذاكرة بلا نموذج للعالم يعرف ما حدث لكنه لا يستطيع توقع أثر الفعل. وإذا غاب التحقق فسيراكم الخطأ أسرع مما يراكم الخبرة. هذه الملاحظات تقود إلى نتيجة: المعرفة ركيزة مركزية، لكنها واحدة من عدة ركائز لا يمكن للذكاء الوكيلي العام الاستغناء عنها.

```
AGI Substrate != Knowledge Alone
AGI-Oriented Agent = Model + Poly-OS + World Interaction
```

2. تعريف Operating Substrate

نقول إن البعد X يصبح Operating Substrate عندما يملك حالة مستمرة قابلة للعنونة، ويستطيع جدولة ما ينشط منها، وتطبيق سياسات، والحفاظ على persistence، والاستجابة لل interrupts، وتوفير interfaces لبقية النظام.

```
OS(X) = {State, Addressing, Scheduling, Policies, Persistence,
                    Interrupts, Interfaces}
```

هذا التعريف يتيح استعمال لغة أنظمة التشغيل بصورة منضبطة بدل أن تكون استعارة تسويقية فضفاضة.

3. Constitution-as-OS

فوق بقية الركائز توجد قواعد عليا تحدد invariants التي لا يجوز للأنظمة المحلية تجاوزها: حدود السلطة، قواعد التحقق، constraints الأخلاقية أو الأمنية، وقواعد الهوية. Constitution-as-OS لا تختار كل فعل، لكنها تحدد الفضاء الذي يمكن لبقية الOS أن تعمل داخله.

```
Policy_local ⊆ Constitution
```

إذا تعارضت خطة محلية مع invariant عليا، يجب أن يفوز الدستور حتى إن كان المسار المحلي أعلى utility.

4. Goal-as-OS

الأهداف ليست prompts عابرة. في نظام مستمر ينبغي تمثيل الأهداف وحالاتها وdependencies والأولويات والمواعيد النهائية والتعارضات. وظيفة Goal OS هي إبقاء الفرق بين الحالة المرغوبة والحالة المرصودة نشطاً عبر الزمن.

```
Δ_t = DesiredState - ObservedState
G_t = {primary_goals, subgoals, constraints, priorities, deadlines}
```

وبذلك يصبح تحقيق الهدف عملية مراقبة مستمرة لا تنفيذ خطة ثابتة ثم افتراض النجاح.

5. Active Difference

يمكن بناء اختيار الأفعال حول خفض مسافة صريحة بين الواقع والهدف. هذه الفكرة تجعل الوكيل قادراً على إعادة التخطيط بعد تغير البيئة بدلاً من الاستمرار في تنفيذ خطة أصبحت غير ذات صلة.

```
D_t = Distance(State_t, Goal_t)
ActionSelection = argmax_a ExpectedReduction(D_t | a)
```

Memory-as-OS .6

يجب التمييز بين working memory وepisodic memory وsemantic memory وprocedural memory وautobiographical memory. إدارة الذاكرة تشمل write وconsolidation وretrieval وforgetting، ولا يمكن اختزالها في vector search واحد. الذاكرة هنا تحفظ الاستمرارية الزمنية وتوفر مادة للتعلم والمراجعة.

```
Memory_t+1 = Consolidate(Memory_t, Experience_t, Utility, Confidence,
                                                        Novelty)
```

Context-as-OS .7

السؤال الحاسم ليس فقط ماذا يعرف الوكيل، بل ما الجزء الذي يفكر به الآن. عندما تكون المعرفة أكبر بكثير من نافذة السياق نحتاج إلى substrate مستقلة تدير selection وcompression وpinning وeviction. وهذا يجعل context مورداً معرفياً مثل الذاكرة الافتراضية في أنظمة التشغيل.

```
C_t = Select(Goal_t, State_t, Memory_t, Knowledge_t, Budget_t)
```

Ontology-as-OS .8

كل استدلال يحتاج إلى units وأنواع. إذا لم يميز النظام بين fact وhypothesis وgoal وevent وcause وevidence وprocedure فلن يستطيع الحفاظ على قواعد تحديث مختلفة لكل نوع. لذلك تمثل ontology type system للعقل الوكيلي. وفي مشروع MOUSSAIF يمكن ربط هذا مباشرة بمحاور MCD/QMCD/HAKIM ومستويات الوصف.

```
Ontology: RawExperience -> TypedCognitiveObjects
```

World-Model-as-OS .9

قاعدة المعرفة تصف حقائق عن العالم، بينما World Model يحاول توقع dynamics العالم. الفرق مهم: معرفة أن الضغط على زر يرسل طلباً لا تكفي لتقدير احتمال النجاح في الحالة الحالية. World Model يحتاج إلى belief state لأن العالم غالباً partially observable.

```
P(s_(t+1) | s_t, a_t)
b_t = P(s_t | o_(1:t), a_(1:t-1))
```

هذه الطبقة هي التي تسمح بالsimulation والمقارنة بين الأفعال قبل تنفيذها.

Reasoning-as-OS .10

لا يوجد نمط استدلال واحد أمثل لكل مهمة. النظام يحتاج إلى router يختار deduction أو induction أو abduction أو causal reasoning أو analogy أو search أو simulation أو tool-assisted reasoning بحسب المهمة وعدم اليقين والكلفة.

```
ReasoningMode_t = Router(Task, Knowledge, Uncertainty, Risk, Cost)
```

وهذا هو جوهر Cognitive Routing: تخصيص نوع التفكير كما يخصص نظام التشغيل الموارد.

Planning/Search-as-OS .11

التخطيط يحول الهدف إلى مسار أفعال، لكن في البيئات المفتوحة يجب أن يكون التخطيط search قابل للإصلاح. Mission DAG أفضل من خطة خطية عندما توجد مهام متوازية وdependencies. أعمال مثل LATS تدعم قيمة دمج reasoning وacting وplanning مع search وfeedback.

```
MissionDAG = (Tasks, Dependencies)
PlanRepair <- WorldChange ∨ VerificationFailure ∨ NewEvidence
```

Skill-as-OS .12

Skill ليست فقرة تعليمات، بل executable contract. ينبغي أن تعرف inputs وpreconditions وprocedure وtools وoutputs وexpected effects وvalidators وrecovery. بهذا تصبح skills شبيهة بـsystem calls معرفية قابلة لإعادة الاستعمال.

```
S_i = <InputType, Preconditions, Procedure, Tools, OutputType, Effects,
                        Validators, FailureRecovery>
```

Ability-as-OS .13

القدرة أعلى من skill واحدة. مثلاً إجراء تحقيق علمي يتطلب البحث والتحقق وتوليد الفرضيات وتصميم التجارب والتحليل والاستشهاد والكتابة. يمكن إذن تمثيل ability كتركيب graph من skills مع شروط انتقال بينها.

```
Ability = Compose(S_1, S_2, ..., S_n)
```

Loop-as-OS .14

كثير من القدرات لا تنتهي بخطوة واحدة، بل تحتاج loop تستمر حتى تحقق invariant أو تصل إلى الهدف أو تنفد الميزانية. لذلك تمثل HyperLoops الركيزة التي تعطي النظام الاستمرارية التشغيلية عبر دورات الملاحظة والاستدلال والفعل والتحقق.

```
Observe -> Reason -> Act -> Verify -> Repeat
Stop iff InvariantSatisfied ∨ GoalReached ∨ BudgetExceeded
```

Agent-as-OS .15

عندما تصبح المهمة أكبر من thread معرفية واحدة يمكن توزيع العمل على وكلاء ذوي أدوار وسياقات وصلاحيات مختلفة. لكن زيادة عدد الوكلاء لا تضمن تحسن الأداء؛ نحتاج إلى orchestration يوازن capability والاعتمادية والكلفة والتداخل.

```
Assign(Task_i, Agent_j) = argmax_j [Capability × Reliability / Cost]
```

Federation-as-OS .16

يمكن أن تتكون المعمارية من kernels أو نماذج متعددة: نموذج سريع للمهام البسيطة، reasoning model للمسائل الثقيلة، vision model للإدراك، verifier مستقل، وربما solver رمزي. Federation OS تختار kernel المناسب وتنسق نقل الحالة بينها.

```
K* = argmax_K ExpectedQuality(K, Task) / Cost(K, Task)
subject to latency, privacy, tool and context constraints
```

Compute-as-OS .17

الحوسبة مورد محدود. لا ينبغي استعمال أغلى نموذج وأطول context لكل خطوة. نقترح Compute Router يرفع مستوى الحوسبة كلما ارتفعت difficulty أو risk أو uncertainty أو value. هذا يقابل مفهوم Sparse/Vascular Compute: إرسال الموارد إلى موضع الحاجة.

```
ComputeAllocation = f(difficulty, risk, uncertainty, value, budget)
```

Tool-as-OS .18

Tool use لا يساوي function calling. يجب أن يمتلك النظام registry لقدرات الأدوات وschemas والصلاحيات والكلفة والlatency والآثار الجانبية ودرجة reversibility وأنماط الفشل. الأدوات تصبح جزءاً من البيئة التشغيلية التي تحتاج discovery وselection وgovernance.

Action-as-OS .19

Tool call حدث برمجي، أما action فهي انتقال في العالم. لذلك يجب فصل execution عن completion. نجاح API لا يكفي؛ يلزم readback أو evidence يثبت أن العالم تغير كما كان متوقعاً. تصميم Agent-Computer Interface نفسه يمكن أن يغير أداء الوكيل جذرياً، كما توضح أعمال SWE-agent وبيئات الحاسوب المفتوحة.

```
a_t: World_t -> World_(t+1)
Execution != Completion
Completion = VerifiedWorldEffect
```

Verification-as-OS .20

التحقق يجب ألا يكون critique اختيارياً في نهاية السلسلة. ينبغي أن يملك قدرة اعتراض claims والخطط والأفعال وعمليات writeback. يمكن أن توجد عدة verifiers: دليلية، منطقية، برمجية، أمنية، وعالمية. وعند اختلافها نحتاج إلى Epistemic Court تفصل بحسب evidence وauthority وindependence.

```
Verifier(x) -> {pass, fail, uncertain}
Court(x, Evidence, Policies) -> Decision + Confidence
```

Authorization/Risk-as-OS .21

الذكاء الوكيلي من دون نموذج سلطة واضح خطر بنيوياً. يجب التفريق بين read وpropose وsimulate وwrite وexecute وirreversible execute. كما يجب ربط درجة التحقق المطلوبة بمخاطر الفعل وعدم قابليته للعكس.

```
ActionAllowed iff Permission ∧ Preconditions ∧ Policy
Risk(a) = Impact(a) × Irreversibility(a) × Uncertainty(a)
```

Learning/Writeback-as-OS .22

التعلم في الوكيل لا يتطلب دائماً fine-tuning. يمكن للنظام أن يتحسن عبر تحديث الذاكرة والمعرفة والمهارات والسياسات ونموذج العالم. الشرط هو أن يكون writeback محكوماً بالتحقق، وأن تظل lineage واضحة حتى يمكن التراجع عن ترقية خاطئة.

```
Learning = MemoryUpdate + KnowledgeUpdate + SkillUpdate + PolicyUpdate +
WorldModelUpdate
```

Evaluation-as-OS .23

الوكيل الذي لا يقيس نفسه لا يعرف أنه تحسن. Evaluation OS تحول النجاح والموثوقية والكلفة والاسترجاع والتعافي إلى telemetry مستمرة، لا benchmark موسمي فقط. هذا مهم خصوصاً عندما يتعلم النظام من الخبرة ويغير بنيته بين مهمة وأخرى.

```
Performance_t = f(success, reliability, cost, recovery, transfer,
safety)
```

Observability/Replay-as-OS .24

نحتاج إلى trace سببي يربط الهدف بالمعرفة بالقرار بالفعل بالأثر. وعند الفشل يجب تحويل الحالة إلى replay قابل لإعادة الإنتاج. هكذا يصبح تاريخ الأخطاء نفسه benchmark نامياً واختبار regression يمنع عودة الفشل بعد ترقيات مستقبلية.

```
Goal -> Knowledge -> Decision -> Action -> Effect
Failure_i -> ReproducibleScenario_i -> RegressionSuite
```

Deployment-as-OS .25

إذا أنشأ الوكيل skill جديدة أو عدل policy، يجب ألا تنتقل مباشرة إلى production. نحتاج إلى sandbox ثم tests ثم canary ثم promotion، مع rollback عند الفشل. هذا هو معنى HyperDeployment على مستوى القدرات الوكيلة.

```
Candidate -> Sandbox -> Tests -> Canary -> Promotion
Failure -> Rollback
```

Continuity-as-OS .26

Uptime ليس identity. بعد restart ينبغي استعادة أهداف المهمة وحالة العالم والافتراضات والمعرفة والذاكرة والصلاحيات والإصدارات. Continuity OS تقيس مقدار ما فقد وما استعاد وتمنع الوكيل من الظهور كنسخة جديدة عديمة الالتزامات كل مرة.

```
ContinuityScore = Similarity(ReconstructedState, ExpectedState)
```

Identity / Self-Reconstruction-as-OS .27

الهوية الوظيفية للوكيل يمكن تمثيلها كمجموعة constitution وأهداف وقدرات والتزامات وتاريخ وسلطة وعلاقات عالمية. إذا ضاعت المحادثة لا ينبغي أن تضيع هذه الذات التشغيلية. من هنا تأتي Self-Reconstruction Memory وContinuity Vault.

```
I = {constitution, goals, capabilities, commitments, history, authority,
active_world_relations}
```

Consistency Bus .28

التحدي الأكبر في Poly-OS ليس وجود الركائز بل اتساقها. قد تقول الذاكرة إن credential صالح بينما يقول world observation إنه منتهي. يجب أن ينتشر التحديث إلى knowledge ثم الخطة والمهارات التي تعتمد عليها. لذلك نحتاج إلى bus للحالة والتبعيات.

```
Observation -> WorldModel -> KnowledgeInvalidation -> PlanRepair ->
                                SkillRecheck
```

هذه الخاصية - propagation المنظم للتغير - هي ما يفصل system architecture عن prompt chain.

Cognitive Transactions .29

قبل mission عالية المخاطر يمكن أخذ snapshot للحالة المعرفية والتشغيلية. إذا وقع corruption يمكن rollback للحالة الداخلية، لكن أفعال العالم قد تكون irreversible، لذلك يجب تسجيل reversibility مسبقاً. Cognitive transactions أوسع من database transactions لأنها تشمل العالم الخارجي.

```
                Snapshot(A_t) -> Execute -> Verify
              If internal_corruption: Rollback(A_t)
If irreversible_world_effect: Compensate rather than rollback
```

Interrupt Bus .30

كل substrate يمكن أن تصدر interrupt: Goal OS عند تغير الأولوية، Knowledge OS عند تناقض جديد، World OS عند حدث خارجي، Verification OS عند خرق Authorization OS، invariant عند سحب الصلاحية وCompute OS عند نفاد الميزانية. runtime العام يحتاج interrupt bus يقرر متى يوقف المسار الحالي ويعيد التخطيط.

31. AGI Kernel

يمكن تعريف kernel أدنى لا يحتوي الذكاء كله بل ينظم عمل modules الذكية. يتضمن scheduler وstate bus وevent bus وpolicy engine وregistry وcheckpointing. بهذا يصبح من الممكن تبديل models مع الحفاظ على runtime architecture.

```
Kernel = Scheduler + StateBus + EventBus + PolicyEngine + Registry +
Checkpointing
```

هذه النقطة مهمة لأنها تفصل Model Intelligence عن System Intelligence. النموذج مورد مركزي، لكنه لا يساوي النظام.

32. Generality وOpen-Endedness

نقترح أن generality لا تقاس بعدد المهام المحفوظة، بل بقدرة النظام على نقل المعرفة والمهارات إلى أوضاع جديدة، والتكيف، وتركيب القدرات، والتعافي من الفشل. أما open-endedness فتظهر عندما يستطيع النظام اكتشاف نقص، خلق subgoal، اكتساب معرفة، تطوير skill، اختبارها ثم إضافتها إلى repertoire.

```
Generality = Transfer × Adaptation × Composition × Recovery
```

هذا يسمح بظهور meta-learning على مستوى النظام حتى عندما تبقى أوزان النموذج ثابتة.

```
Weights_(t+1) = Weights_t
but SystemPolicy_(t+1) != SystemPolicy_t
```

33. فرضيات Poly-OS القابلة للتكذيب

1. حذف Goal OS سيزيد task drift وفقد الأولويات طويلة الأفق.
2. حذف Memory OS سيزيد التكرار وفقد الاستمرارية بعد restart.
3. حذف World Model سيزيد planning brittleness في البيئات المتغيرة.
4. حذف Verification OS سيزيد false-success وcontaminated writeback.

5. حذف Skill OS سيخفض transfer رغم بقاء المعرفة التصريحية.
6. حذف Continuity OS سيؤدي إلى انهيار المهام الممتدة عبر جلسات متعددة.
7. حذف Consistency Bus سيزيد عدد الحالات المتعارضة بين الذاكرة والعالم والخطة.
8. تفعيل Compute Routing سيخفض الكلفة عند ثبات الأداء مقارنة باستعمال أعلى model دائماً.

34. Ablation Matrix

الركيزة المحذوفة	الفشل المتوقع	المقياس الرئيسي
Goal OS	Task drift	Goal persistence
Knowledge OS	Rediscovery / factual inconsistency	Knowledge reuse
Memory OS	Repeated mistakes	Repeated-error rate
World Model	Brittle plans	Plan repair success
Skill OS	Low procedural transfer	Cross-task transfer
Verification OS	False completion	Verified success gap
Writeback Governor	Epistemic contamination	False-write rate
Continuity OS	Restart collapse	State recovery score
Compute Router	High cost / latency	Quality-per-cost

35. MUG / MUT-AGI كإطار اختبار

يمكن تحويل Poly-OS من مخطط معماري إلى برنامج تجريبي داخل MUG/MUT-AGI. يصبح السؤال ليس "هل تبدو المعمارية جيدة؟" بل: أي substrates ضرورية؟ أيها متكررة وظيفياً؟ متى تظهر synergy؟ ومتى تصبح الكلفة أكبر من الفائدة؟

```
MUT: Does multi-substrate governance improve robust generalization?
MUG: Which substrates are necessary, sufficient, synergistic, or
redundant?
```

MOUSSAIF AGI Poly-OS .36

داخل مشروعنا يمكن رسم الخريطة الإغلاقية التالية: Constitution/Meta-OS في الأعلى، Goal Compass للحالة القصدية، HKOS للمعرفة، MCD/QMCD للأنطولوجيا، Self-Reconstruction Memory للزمن والهوية، Context Map للسياق، World Model للبيئة، Cognitive Routing للاستدلال، Mission DAG للتخطيط، HyperSkills/HyperAbilities للقدرة، HyperLoops للاستمرارية، HyperAgents للتوزيع، World-Effect Loop للفعل، Falsification Court للتحقق، authority tiers للسلطة، Vascular Compute للحوسبة، governed writeback للتعلم، Evaluation gates للقياس، telemetry/replay للرصد، HyperDeployment للنشر وContinuity Vault للاستعادة.

```
                                Constitution
                                     ↓
                Goal -> Knowledge/Ontology <-> Memory <-> WorldModel
                                     ↓
        Context -> Reasoning -> Planning -> Skills/Abilities -> Loops/Agents
                                     ↓
        Tools -> WorldEffect -> Verification -> Writeback -> Evaluation ->
                                SelfReconstruction
```

ويحيط بهذه السلسلة Authorization وObservability وCompute Routing وDeployment وContinuity.
بهذه الصياغة لا تعود المحاور أسماء متجاورة، بل حالات تشغيلية مترابطة ضمن نظرية واحدة.

37. الخاتمة

Knowledge-as-OS مهمة لأنها تحول المعرفة من content إلى control substrate، لكن الاتجاه نحو الذكاء الوكيلي العام يتطلب تحولاً موازياً في بقية الأبعاد. الهدف يصبح persistent operating state، الذاكرة managed temporal state، السياق scheduled cognitive resource، العالم continuously updated model، المهارات executable reusable capability، الأفعال verified world transitions، التحقق gate إلزامياً، التعلم governed writeback، والهوية reconstructible continuity.

```
General Agentic Intelligence emerges from coordinated state management
    across knowledge, goals, memory, world, capabilities, action,
                    verification and continuity.
```

وعليه فإن الوصف الأدق للمعمارية المقترحة هو MOUSSAIF AGI Poly-OS: نظام تكون فيه HKOS هي Epistemic OS، وGoal/Compass هي Intentional OS، وSelf-Reconstruction Memory هي Temporal/Identity OS، وHyperSkills/Abilities/Loops هي Capability OS، وWorld-Effect + Courts هما Action/Truth OS، وHyperRouting/Vascular Compute هو Compute OS. القيمة العلمية لهذا التصور تتوقف على ablations وbenchmarks وقدرته على تقديم تنبؤات فشل محددة قابلة للتكذيب.

المراجع الأساسية

1. Sumers, T. R. et al. (2023). Cognitive Architectures for Language Agents (CoALA). arXiv:2309.02427.
2. Packer, C. et al. (2023). MemGPT: Towards LLMs as Operating Systems. arXiv:2310.08560.
3. Yao, S. et al. (2022). ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629.
4. Shinn, N. et al. (2023). Reflexion. arXiv:2303.11366.
5. Wang, G. et al. (2023). Voyager. arXiv:2305.16291.
6. Zhou, A. et al. (2023). Language Agent Tree Search Unifies Reasoning, Acting, and Planning in Language Models. arXiv:2310.04406.
7. Yang, J. et al. (2024). SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. arXiv:2405.15793.
8. Xie, T. et al. (2024). OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments.
9. Liu, X. et al. (2023). AgentBench: Evaluating LLMs as Agents. arXiv:2308.03688.
10. Park, J. S. et al. (2023). Generative Agents. arXiv:2304.03442.

Poly-OS مفهوم معماري مقترح في هذه الورقة لدمج الركائز الوكيلة في نموذج قابل للاختبار. المقصود بـOS-like وجود state وpolicies وscheduling وpersistence وinterfaces، وليس الادعاء بأن كل ركيزة نظام تشغيل برمجياً مستقلاً.