Why Scaling Models Is Not Enough

The Case for Organizational Depth in Agent Architecture

Hadosh Academy — Agents Series, Essay 2 (Companion White Paper)

February 2026

This white paper accompanies the blog essay "We Could Have Had AGI By Now." The blog makes the case in accessible terms. This paper provides the analytical framework, falsifiable predictions, and a practical experiment for builders who want to test the hypothesis in their own work.

1. Two Kinds of Complexity

When people say "complex system," they often conflate two fundamentally different meanings.

Internal complexity (function complexity). A trained transformer is a high-dimensional, nonlinear function. It exhibits emergent behavior. It can be sensitive to small perturbations. It can approximate many computations.

Organizational complexity (system complexity). A complex organization consists of interacting subsystems with boundaries, roles, persistence, and multi-timescale adaptation. It can grow by adding parts, not only by stretching one part.

A modern LLM has the first kind of complexity in abundance. It has the second kind only weakly, if at all. A single model has no durable identity outside its context. Its "state" is the prompt. That is not what long-horizon agency looks like.

2. Why "Turing Complete" Misses the Point

Neural networks can emulate arbitrary computation in principle. But "in principle" is not the target. The target is a practical substrate for building systems that:

- Keep working after the chat ends
- Improve from experience
- Remain coherent across months
- Can be inspected, updated, and governed

A single monolithic policy can simulate many modules inside itself, but that simulated modularity is hard to inspect, hard to independently update, and hard to evolve without interference. This is exactly why we build software as modules instead of one giant file. Not because one giant file is impossible. Because it is a terrible substrate for growth.

3. Context Provenance: A Useful Lens

Consider an agent at time t. The model has a context window. That context is the agent's working state. Now ask: where did that state come from?

Generated state: tokens produced by the model in the current session. **Injected state:** content inserted by architecture — files, hooks, policies, tool results, schemas, memory retrieval, logs.

A model-centric worldview pushes toward maximizing generated state; make the model do it all. An architecture-centric worldview pushes toward increasing injected state; keep durable structure outside the model; treat the model as a cognitive module.

Injected state enables persistence across sessions, selective retrieval (only load what matters), deterministic guardrails (hooks), and localized upgrades (edit a file, replace a module). Generated state

alone creates a closed loop. Closed loops can be clever, but they tend to be unstable under long horizons.

3.1 Injected Context Ratio

If you want one practical metric to track whether you are building an organization or a monologue, use a rough injected context ratio:

> ***Injected ratio = (tokens from files, tools, hooks, memory) / (total context tokens)***

A low ratio means the system mostly regenerates state internally; long-horizon behavior depends on fragile self-consistency. A higher ratio means the system depends on stable external anchors; behavior becomes easier to audit and stabilize. Over time, an architecture-centric system should show more injected context from durable playbooks, less repeated re-derivation of the same constraints, and fewer failures from forgotten steps.

4. The Random Walk Problem in Agent Behavior

A bare CLI agent has an action space: reply, think, call a tool, ask permission, delegate, compact, stop. With minimal structure, the model chooses among these actions probabilistically. Over time you get a chain of actions that is sensitive to small context changes. The same task can produce different sequences on different runs.

This is not a moral failure of the model. It is a property of using a probabilistic policy without a regulatory layer. Organizational systems avoid this by adding processes, checklists, roles, gates, and enforcement. That is what hooks and instruction files (CLAUDE.md, AGENT.md) are in CLI agents: the earliest form of organizational regulation.

5. The Core Hypothesis

> ***To reach AGI-like long-horizon competence, systems must increase organizational depth (number of interacting persistent subsystems) at least as much as they increase model capacity.***

Organizational depth includes:

- Persistent memory stores with clear schemas
- Event-driven hook systems (Claude Code hooks, OpenCode plugin events, Gemini CLI hooks)
- Job/behavior schedulers with stop-blocking enforcement
- Verification and rollback mechanisms
- Multi-timescale learning loops (work vs. consolidation)
- Modular skill libraries

If this hypothesis is true, then model scaling alone can keep raising ceiling performance, but it will not produce the stability and evolvability that broad labor replacement requires.

6. What Makes an Organization Complex in the Useful Sense

A practical checklist for organizational complexity, with mappings to agent architecture:

Property	In Organizations	In Agent Architecture
Hierarchy	Subsystems inside subsystems	Directory trees and nested instruction layers
Boundaries	Clear interfaces, scoped context	Compartmentalized files and scoped memory
Persistence	State survives across interactions	Durable stores (knowledge, logs, job ledgers)
Event handling	Automatic reflexes, compliance	Hooks around tool use and lifecycle events
Multi-timescale	Fast actions, slow restructuring	Work loop vs. consolidation loop
Local modification	Improve one part, keep rest stable	Edit one skill, replace one module
Selection	Keep what works, retire what fails	Measurable evaluation and promotion/retirement

A transformer can be internally complex, but it does not naturally expose these organizational levers. A monolithic model has one main way to change: adjust weights (via training) or adjust prompt (via context). Both are global interventions. Neither is a local, modular edit with stable boundaries.

7. Monolithic Models vs. Modular Organizations

Even if a huge model could internalize the equivalent of hooks, memory routing, and policies, external organization still has practical advantages:

- **Replaceability:** swap the engine without rewriting identity
- **Local change:** edit one rule without retraining everything
- **Inspectability:** read the policy file; audit the decision
- **Safety:** enforce tool constraints at the hook layer
- **Incremental growth:** add a new module without destabilizing the whole

These are not aesthetic preferences. They are engineering properties of evolvable systems. The economic argument reinforces this: organizational improvements are often cheap, deterministic, and compositional. A verification checklist costs almost nothing. A budgeting rule costs almost nothing. A tool permission gate costs almost nothing. If these prevent even a small fraction of failures, they dominate marginal gains from scaling in many real deployments.

8. The Minimal Seed Agent: Roles and Differentiation

Instead of arguing about a single "minimum number" of components, define a minimum set of roles a system must perform:

Role	Function	Implementation
Scheduler	Jobs, priorities, budgets	Job ledger (JSON), priority queue
Planner	Decompose objective into jobs	Planning prompt + job creation
Executor	Tool calls, file edits	Standard CLI tool use
Verifier	Check outputs against criteria	Post-execution validation hook
Memory Mgr	Store, retrieve, scope	Compartmentalized file system
Consolidator	Distill experience into durable forms	Sleep/consolidation loop
Critic	Detect recurring failure patterns	Log analysis during consolidation
Skill Builder	Create new tools/workflows	Self-modification of agent directory
Watchdog	Detect loops and runaway behavior	Budget hooks, loop detection

Each role can be implemented as a job type, an intention, a module, or an internal specialized model. The key is not the label. It is differentiation. An agent that performs all of these roles through a single undifferentiated prompt is functionally less organized than one that separates them into distinct, inspectable components.

9. A Concrete Architecture Pattern: Jobs + Hooks + Memory Metabolism

One concrete pattern that embodies these principles is a persistent job ledger:

- Jobs exist as structured objects (e.g., JSON files on disk)
- Each job has: objective, success criteria, budget, status, and an observation log
- Hooks block stopping while active jobs exist (Claude Code Stop hook, OpenCode session.idle event, Gemini CLI AfterAgent hook)
- A job cannot complete until its observation log is distilled into durable memory and artifacts

This pattern forces the system to behave like an organization: obligations cannot be ignored, experience cannot be left unprocessed, and progress has a verifiable lifecycle. The same pattern can be expressed as intentions and behaviors if you prefer: behaviors are reusable actions, intentions are compositions of behaviors, and an intention scheduler replaces the job ledger. The representation can vary. The organizational principle is stable.

9.1 Governance Without Freezing the System

If the architecture can modify itself, how do you avoid runaway drift? Separate the ability to change from the ability to immediately deploy change. A practical approach is two-phase commit: propose changes to a staging area, evaluate them against tests and metrics, and promote them only if they improve outcomes without violating constraints. This does not require an immutable constitution. It requires a deployment gate.

10. Falsifiable Predictions

A common failure mode of architecture debates is vagueness. Here are concrete, testable predictions that follow from the organizational depth hypothesis:

Prediction A:

A smaller model with strong organizational depth can outperform a larger model with weak organization on long-horizon tasks, at lower cost.

Prediction B:

Systems that externalize memory and verification will show steadier improvement over time without retraining the main model.

Prediction C:

The fraction of injected context (the injected context ratio) will correlate with stability — lower variance across runs for the same task.

These are measurable. You can run two systems for months: one that mostly regenerates state within the model, one that persists state and uses hooks to enforce lifecycles. Compare outcomes.

11. A Practical Experiment for Builders

If you want to test the organizational depth hypothesis in your own work, here is a controlled experiment you can run:

Setup

Pick a long-horizon project — something that takes weeks, not minutes. A multi-phase research project, a software build with iterative requirements, a content production pipeline with multiple deliverables. Run it twice:

Version A (Minimal Structure): Use a CLI agent in a mostly empty directory. Rely on chat-based interaction and ad-hoc prompts. No persistent job tracking, no hooks beyond defaults, no consolidation phases, no compartmentalized memory files. The model generates and regenerates its own context each session.

Version B (Organizational Depth): Use the same CLI agent with explicit architectural layers: a CLAUDE.md or AGENT.md file defining operational phases; hook-based enforcement (stop-blocking for active jobs, pre-tool validation, post-tool logging); a persistent job ledger tracking objectives, criteria, and status; a compartmentalized file structure for knowledge, memory, and skills; and scheduled consolidation phases where traces are analyzed and distilled into durable upgrades.

Metrics to Track

Metric	What It Measures	Expected Result
Retries per milestone	Reliability of first-attempt execution	Version B: fewer retries
Missed requirements	Completeness of output	Version B: fewer misses
Cost/time per milestone	Efficiency over long horizons	Version B: lower after initial setup
Context restatements	How often you re-explain the same thing	Version B: near zero after week 1
Cross-run variance	Consistency of approach across sessions	Version B: lower variance
Injected context ratio	Architecture maturity (see Section 3.1)	Version B: increasing over time

Expected Outcome

In most real projects, Version B wins even with a weaker model. The initial setup cost is higher, but it amortizes quickly as the agent accumulates structured experience. By week two or three, Version B should require significantly fewer human interventions per milestone, show more consistent output quality, and cost less per unit of useful output.

If Version A wins — that is, if the unstructured approach is cheaper, more reliable, and more consistent over the full project duration — then the organizational depth hypothesis is weakened for that class of task. Report the result either way. The point is measurement, not advocacy.

12. Common Objections

"A huge model with long context solves memory." Long context reduces some friction, but it does not create selection, boundaries, or modular upgrades. It gives you a bigger whiteboard, not a better organization.

"MoE models are modular already." Mixture-of-experts is internal specialization. It can help. But it is still trapped inside the weight space and is not easily governed or locally upgraded by the user.

"External structure is just scaffolding." Scaffolding is how organisms are built. The claim that scaffolding is temporary assumes the model is the final organism. The more realistic view is that scaffolding becomes the skeleton.

"Bigger models will internalize architecture." Even if internalization is possible, it is not modular (you cannot inspect or swap the internalized hook) and it is economically brutal (you are paying model-scale costs for what software can do cheaply). If your goal is practical labor replacement, "in principle" is not a plan.

13. The Biological Analogy, Used Carefully

Analogies are dangerous when they pretend to be proofs. Use this one as a lens, not a claim of equivalence.

Early life likely involved molecules that both stored information and performed functions. Over time, biology separated concerns: storage and heredity, catalytic execution, membranes and compartment boundaries, regulation and signaling, and later specialized tissues and nervous systems. The lesson is not "RNA is bad." The lesson is that higher-order capability required new organizational layers.

In agent terms:

- The LLM is a powerful catalytic engine
- Persistent files (CLAUDE.md, AGENT.md, knowledge stores) are heredity and identity
- Hooks are regulation and reflexes
- Tools are effectors
- Consolidation loops are remodeling

A single scaled model may approximate some of this internally, but it does not give you clean, evolvable layers you can govern.

14. The Hook System Landscape (February 2026)

The convergence of hook systems across CLI agent platforms validates the architectural argument. Three major platforms now offer lifecycle interception:

Platform	Hook Mechanism	Key Events	Config Location
Claude Code	Hooks via settings.json	PreToolUse, PostToolUse, Stop, Notification, SubAgentStop, PreCompact	.claude/settings.json
OpenCode	Plugin event system	tool.execute.before/after, session.idle, session.created, stop, session.compacting	.opencode/plugin/*.ts
Gemini CLI (v0.26.0+, Jan 2026)	Hooks via settings.json	BeforeTool, AfterTool, BeforeAgent, AfterAgent, BeforeModel, AfterModel	.gemini/settings.json
Codex CLI	No hook system	Only basic notify (agent-turn-complete)	config.toml (limited)

The absence of hooks in Codex CLI is notable. A community-submitted hooks implementation (PR #9796) was declined by the OpenAI team in January 2026. Feature requests remain open (Issue #2109, Discussion #2150). Without lifecycle interception, Codex users cannot enforce deterministic structure around the model's probabilistic behavior — the agent remains closer to a raw engine than a governed organization.

15. Summary

Model scaling increases internal function complexity. Architecture scaling increases organizational complexity. If your target is long-horizon, self-improving, career-capable agency, organizational

complexity is not optional. It is the substrate.

The practical roadmap: build a filesystem brain with compartmentalized memory; enforce phases and permissions with hooks; track obligations with persistent jobs; and run consolidation loops that convert experience into durable upgrades. The engine can be swapped. The architecture is the organism.
