Nick Clark · Published April 25, 2026 · PDF

OpenAI AgentKit and the Assistants/Responses API vs agent-carried, hardware-anchored identity with governed tool lifecycle

OpenAI AgentKit and the Assistants and Responses APIs give developers hosted building blocks for agents: tool definitions, hosted threads, server-side conversation state, and managed retrieval. The hard problem underneath is where an agent's identity, its accumulated history, and its authority over the tools it calls actually live. This article compares that model against the Agent-Resident Execution Substrate, disclosed in U.S. Provisional Application No. 64/070,239, in which the agent persists on the device as a continuity-proven, hardware-anchored identity that owns its inference tools as governed assets.

What OpenAI AgentKit and the Assistants/Responses API (agent building blocks: tools, hosted state, threads) Does

OpenAI AgentKit and the associated Assistants and Responses APIs give developers a well-designed set of building blocks for constructing agents on top of hosted models. A developer defines an assistant or a response configuration with instructions, a model selection, and a set of tools. Tools include function calling, where the model emits a structured call the developer's code fulfills, alongside hosted capabilities such as file search over uploaded documents and code execution. Conversation state is managed

server-side: threads (in the Assistants API) and stored responses (in the Responses API) let the platform retain message history and tool results so a developer does not have to resend the entire context on every turn.

These are genuinely useful abstractions, and OpenAI executes them well. Hosted state removes a large class of plumbing from the developer's plate. Tool definitions are clean and declarative. File search provides competent retrieval without the developer standing up a vector store. The platform handles model hosting, scaling, and versioning, so a small team can ship a capable agent quickly. For a wide range of applications, that is exactly the right tradeoff, and nothing in the comparison below suggests otherwise.

The architectural point to be precise about is where the durable parts of an agent reside. In this model, the assistant configuration, the thread or stored-response history, the uploaded files, and the tool definitions live in OpenAI's service. The agent is, in effect, a configuration plus a conversation record held on the provider's infrastructure and addressed by identifiers. Tools are things the model is told about and invokes; they are not assets the agent owns, versions, or governs. This is a reasonable and deliberate design for a hosted developer platform. It is also the axis on which the disclosed invention differs.

The Architectural Axis

The axis is the locus and durability of agent identity, together with the agent's relationship to its tools.

In the hosted-building-block model, identity is a configuration handle. An assistant ID or a chain of response IDs names a stateful object on the server, and the durable history of what the agent did lives in that provider-held thread. The tools an agent uses are declarations passed into a request; the model decides when to call them, and the developer's code or the platform fulfills them. There is no notion, at this layer, of the

agent cryptographically proving that it is the same continuous entity across time, of the agent carrying a tamper-evident record of its own operational history, or of the agent holding governed authority over the lifecycle of the tools it invokes. Those are simply not the problems these APIs set out to solve.

The disclosed invention addresses exactly those properties. Framed as a difference rather than a defect: the hosted model optimizes for fast construction of capable agents on shared infrastructure, while the substrate optimizes for a durable, verifiable, self-governing agent whose identity and history are properties of the agent itself rather than of a hosted record. The two answer different questions about what an agent fundamentally is.

How the Disclosed Approach Differs

The Agent-Resident Execution Substrate, disclosed in U.S. Provisional Application No. 64/070,239, treats the semantic agent as a persistent execution substrate of the computing device rather than as a configuration on a hosted service. The spec describes the agent as comprising a persistent identity field, a cognitive state field, a lineage field, and a governance policy field, maintained continuously across the lifetime of the device.

Three mechanisms distinguish it on this axis, all traceable to the specification.

First, identity is continuity-proven and can be hardware-anchored. The spec describes an identity field carrying continuity metadata and, in an embodiment, a continuity hash chained over the sequence of substrate events recorded in the lineage field, verifiable by reproducing the chain from any prior reference value, with a continuity break detectable as a discrepancy that triggers escalation under policy. In a further embodiment the identity field is cryptographically bound to a hardware security element such as a secure enclave, trusted platform module, or embedded secure

element, with private key material that is not extractable from the hardware element, and the binding is verifiable at any subsequent substrate event. This is a materially different claim than addressing an agent by a server-side identifier.

Second, tools are owned managed assets subject to a governed lifecycle, not declarations the model is handed. The spec describes a managed inference tool registry in which each managed inference endpoint comprises a model artifact, an interface specification, and an associated governance scope, and a tool lifecycle controller that performs installation, retraining, replacement, archival, and removal, each operation evaluated against the governance policy field and recorded in the lineage field. The spec sets out an explicit tool lifecycle state machine (uninstalled, installed-inactive, active, retraining, updated-active, archived, removed) with governed transitions including rollback on failed validation. The agent is described as the authority over these operations, so the tools are subordinate assets it governs rather than external services it merely calls.

Third, history is a tamper-evident, agent-carried lineage. The spec describes the lineage field as an append-only sequence of records, each verifiable against its predecessor under a continuity proof, structured so the complete operational history is deterministically reconstructible and no prior record can be modified or deleted without producing a detectable continuity break. Critically, the spec ties this lineage to a distinct capability: because the substrate persists across model replacement, the agent's identity, cognitive state, and lineage are preserved even when every managed inference endpoint in the registry is replaced or the substrate runtime itself is updated, under a continuity attestation linking identity across versions.

The practical consequence, per the spec, is that the agent is decoupled from any specific model. The model artifacts loaded at any moment constitute the agent's currently available capabilities, but the agent's identity does not depend on any of them and survives their replacement. That is the inverse of a model where the durable object is the hosted thread and the model is the fixed backend.

Where They Fit Together

These are complementary more than mutually exclusive, and the honest framing is composition rather than displacement.

The substrate does not require that all inference run locally. The spec describes a cloud-burst forwarding subsystem that selectively forwards inference requests to a remote inference endpoint operated by a network-accessible service when local endpoints lack capability or capacity, governed by a cloud-burst policy object specifying admissible remote endpoints, disclosure scopes, encryption requirements, and cost limits, with each forwarding event recorded as a disclosure event in the lineage field. A hosted model behind AgentKit or the Responses API is exactly the kind of high-capability remote endpoint a substrate deployment would forward to when its local, bounded-size models are insufficient.

Read that way, the two occupy different layers. OpenAI's building blocks are an excellent way to build and host a capable agent quickly and to reach a frontier model. The substrate is an architecture for a durable, governed, device-resident agent that owns its identity and history and can draw on hosted inference as one governed tool among several. A team could reasonably use hosted APIs today for the model tier and adopt substrate-style identity, lineage, and tool governance for the properties those APIs do not target. The comparison is about which layer holds the agent's identity, not about which model answers a prompt.

Boundary Conditions

Honesty about the disclosed side matters as much as fairness about the competitor.

The substrate is described in a provisional application. It is an architectural disclosure with named embodiments and incorporated-by-reference building blocks; it is early-stage and its claims trace to the specification, not to a shipped, benchmarked product. No performance numbers are asserted here for the disclosed approach, because none

are claimed in a way that would be responsible to cite as measured fact. The device-local model the substrate depends on is explicitly bounded: the spec frames model artifacts as sized to fit local memory, storage, and compute, and repeatedly relies on cloud-burst forwarding precisely because local endpoints will sometimes lack capability. Hardware-anchored identity depends on the presence of a suitable hardware security element and is described as available "where supported," not universally.

On the other side, the hosted building blocks are mature, widely used, and continuously evolving. Nothing here should be read as asserting that OpenAI's platform has a defect. It targets developer velocity and access to strong hosted models, and it succeeds at that. The properties the substrate emphasizes are simply outside that platform's stated scope, and platform capabilities change over time; readers should verify current features against OpenAI's own documentation rather than treating this snapshot as authoritative.

Disclosure Scope

The mechanisms attributed to the disclosed approach in this article are those set out in U.S. Provisional Application No. 64/070,239 and are described at the level of the specification's embodiments; they define the scope of what is disclosed there and nothing broader. The references to OpenAI AgentKit and the Assistants and Responses APIs, and to the hosted-agent market generally, are provided as external context to locate the disclosure among familiar approaches; they are not characterizations made by or claims of the filing, and they should be independently verified against OpenAI's current documentation. Nothing in this comparison asserts that OpenAI's products are defective, infringing, or deficient. The comparison is confined to a single architectural axis, identity, lineage, and tool governance, and describes a structural difference in where those properties reside, not a judgment about the quality or suitability of either approach for any particular use.

Agent-Resident Execution Substrate (/agent-resident-execution-substrate)

All 40 steps → (/inventive-steps)

Persistent execution environment carried by the agent, not the host — identity, state, and lineage across power cycles, devices, and upgrades.

Provisional application

PRIMARY TECHNICAL DISCLOSURE

- Agent-Resident Execution Substrate, Articles (/articles/agent-resident-execution-substrate)

SECONDARY TECHNICAL

- Persistent Semantic Agent (/articles/agent-resident-execution-substrate/persistent-semantic-agent)
- Managed Inference Tool Registry (/articles/agent-resident-execution-substrate/managed-inference-tool-registry)
- Agent-to-Tool Dispatcher (/articles/agent-resident-execution-substrate/agent-to-tool-dispatcher)
- Lineage-Derived Training Signal (/articles/agent-resident-execution-substrate/lineage-derived-training-signal)
- Identity Preservation Across Upgrades (/articles/agent-resident-execution-substrate/identity-preservation-across-upgrades)
- Cognitive State-Conditioned Dispatch (/articles/agent-resident-execution-substrate/cognitive-state-conditioned-dispatch)
- Governed Tool Lifecycle (/articles/agent-resident-execution-substrate/governed-tool-lifecycle)
- Continuity-Proof Lineage (/articles/agent-resident-execution-substrate/continuity-proof-lineage)
- Substrate Runtime Continuity (/articles/agent-resident-execution-substrate/substrate-runtime-continuity)
- Personal Corpus Model Training (/articles/agent-resident-execution-substrate/personal-corpus-model-training)
- Heterogeneous Inference Endpoints (/articles/agent-resident-execution-substrate/heterogeneous-inference-endpoints)
- Atomic Lifecycle Substitution (/articles/agent-resident-execution-substrate/atomic-lifecycle-substitution)
- Integrity Signal Feedback (/articles/agent-resident-execution-substrate/integrity-signal-feedback)
- Hardware-Bound Identity (/articles/agent-resident-execution-substrate/hardware-bound-identity)

- Cognitive State Append-Only Invariant (/articles/agent-resident-execution-substrate/cognitive-state-append-only-invariant)
- Counterparty Identity Records (/articles/agent-resident-execution-substrate/counterparty-identity-records)
- Privacy Egress-Controlled Disclosure (/articles/agent-resident-execution-substrate/privacy-egress-controlled-disclosure)
- Federated Cross-Device Agent Identity (/articles/agent-resident-execution-substrate/federated-cross-device-agent-identity)

APPLICATIONS · GENERAL

- Personal AI Agents That Survive Device Loss: One Continuous Identity and a Private Corpus Across Every Device (/articles/agent-resident-execution-substrate/personal-cross-device-agents)
- Enterprise Agent Fleets: Stable Agent Identity and Governed Tool Access Across Model Upgrades and Infrastructure Migration (/articles/agent-resident-execution-substrate/enterprise-agent-fleets)
- Audit-Grade Agent Identity for Regulated Finance and Healthcare: Continuity-Proof Lineage Across the Agent Lifecycle (/articles/agent-resident-execution-substrate/regulated-industry-agents)
- Edge and On-Device Agents: Hardware-Bound Identity Across Heterogeneous Inference Endpoints (/articles/agent-resident-execution-substrate/edge-and-on-device-agents)
- Agent-to-Agent Commerce With Counterparty Identity Records and Egress-Controlled Disclosure (/articles/agent-resident-execution-substrate/agent-to-agent-commerce)
- Governed Tool Lifecycles for Managed Inference-Provider Ecosystems: A Substrate Approach to Owning, Routing, and Retiring AI Tools (/articles/agent-resident-execution-substrate/managed-tool-ecosystems)
- Proving Unbroken Continuity in Long-Lived Autonomous Systems Across Substrate Migration and Atomic Model Substitution (/articles/agent-resident-execution-substrate/long-lived-autonomous-systems)

APPLICATIONS · SPECIFIC

- LangGraph Platform (LangChain) vs an agent-resident execution substrate: orchestration-graph state versus a portable, hardware-anchored agent runtime (/articles/agent-resident-execution-substrate/langgraph-platform)
- **OpenAI AgentKit and the Assistants/Responses API vs agent-carried, hardware-anchored identity with governed tool lifecycle (/articles/agent-resident-execution-substrate/openai-agentkit)**
- Microsoft Copilot Studio vs an agent-resident execution substrate: platform-hosted agent authoring versus portable, device-resident agent identity and continuity (/articles/agent-resident-execution-substrate/microsoft-copilot-studio)

- Google Vertex AI Agent Engine (managed runtime for deploying and scaling agents, with sessions/memory) vs an agent-carried, continuity-proofed identity substrate (/articles/agent-resident-execution-substrate/google-vertex-agent-engine)
- AWS Bedrock AgentCore (runtime, memory, identity, and gateway services for deploying agents at scale) vs an agent-resident execution substrate: where does the agent identity actually live? (/articles/agent-resident-execution-substrate/aws-bedrock-agentcore)
- Letta (formerly MemGPT) vs an append-only cognitive-state substrate: what a memory-management framework does not provide (/articles/agent-resident-execution-substrate/letta-memgpt)
- Cognition's Devin, an autonomous AI software-engineering agent vs a portable, continuity-proofed agent-resident runtime (/articles/agent-resident-execution-substrate/cognition-devin)
- Cloudflare Agents (Durable Objects) vs an agent-resident execution substrate: portable hardware-bound identity and continuity-proof lineage (/articles/agent-resident-execution-substrate/cloudflare-agents)

Agent-Resident Execution Substrate overview → (/agent-resident-execution-substrate)