Nick Clark · Published April 25, 2026 · PDF

How to Build an Offline-First Personal AI That Syncs When Connected

You want a personal AI that runs on your own device, keeps working with no network, and reconciles cleanly the moment it reconnects. Most stacks make you choose between local privacy and cloud convenience. This guide walks through an architectural approach that does neither by design, disclosed in U.S. Provisional Application No. 64/070,239. The load-bearing idea is the Agent-Resident Execution Substrate inventive step: the agent, not the model, is the persistent thing, and connectivity becomes an optional lifecycle event rather than a dependency.

What You Are Building

You are building a personal AI that is offline-first: it runs inference on the user's own device, keeps a durable record of what it has done, and when a network appears it exchanges just enough with other devices or services to stay coherent, without ever making connectivity a precondition for basic operation. This is the pattern a developer reaches for when a phone, laptop, vehicle, or embedded device has to remain useful in a tunnel, on a plane, or inside a privacy boundary, and then reconcile the moment it comes back online.

The naive version of this is easy to describe and hard to get right. "Cache some prompts and replay them later" collapses the instant the user edits work on two devices, or the model gets retrained, or you need an audit trail of what left the device. The architecture below, disclosed in U.S. Provisional Application No. 64/070,239, is not a library you install. It is a design you implement. What it gives you is a clean answer to the question "what is the durable identity of this AI, and what is merely a replaceable part," and that answer is what makes offline-first and sync tractable.

Why the Obvious Approaches Fall Short

The common approaches each solve one slice and leave a structural gap.

A hosted inference API is shared across all users and keeps no persistent representation of any one user's body of work; you resupply your context in every request and it is discarded afterward. That is fine online and useless offline. Retrieval-augmented generation adds a local index and injects retrieved fragments at query time, but the base model never internalizes the corpus; it is re-consulted on every query, and its quality rides on retrieval and chunking. On-device inference frameworks run a model locally and route requests to it, but they are request-routing infrastructure: they hold no persistent identity or state, accumulate no history of outcomes, and preserve no continuous entity when you swap the model. User-initiated fine-tuning can specialize a model, but it is a manual, decoupled event, not something that tracks your ongoing work.

Federated learning and cloud-account sync are the usual answers for "many devices." Federated learning coordinates local copies of a shared model by averaging weight updates into a common artifact; cloud-account association synchronizes files or settings under a central account identifier. Both are real and useful, and neither produces a persistent agent identity that operates as one agent across devices. They sync weights or

files; they do not sync a continuous identity-bearing entity with a verifiable history. That missing entity is exactly what an offline-first personal AI needs, because it is the thing that has to survive going offline and reconcile on reconnect.

The Architecture

The central inversion is this: the semantic agent is the persistent execution substrate of the device, and inference endpoints are managed assets subordinate to it. Concretely, the agent holds four persistent fields: a persistent identity field, a cognitive state field, an append-only lineage field, and a governance policy field. Models live separately, in a managed inference tool registry, each endpoint carrying a model artifact, an interface specification, and a governance scope. A dispatcher routes requests to endpoints; a lifecycle controller installs, retrains, replaces, archives, and removes them. The agent's identity does not depend on any specific model artifact and is preserved across replacement, retraining, or removal of any model.

That separation is what makes the AI offline-first. Inference runs entirely on-device against locally resident endpoints, and the spec is explicit that the privacy invariant is operative regardless of network connectivity: on-device inference is not an off-device disclosure event. Nothing about producing an answer requires the network.

The lineage field is the durable spine. It is an append-only sequence of records covering dispatched inference requests and their outcomes, integrity-signal feedback, lifecycle operations, ingestion events, policy updates, counterparty encounters, and more. It is structured so the agent's complete operational history is deterministically reconstructible, and each record is chained to its predecessor under a continuity proof so no prior record can be altered without a detectable break. This is what you reconcile on reconnect: not opaque state, but a verifiable, ordered log.

Personalization lives in a per-user personal corpus model. This is an endpoint whose parameters are fine-tuned against artifacts the user authored, curated, or designated under their own governance policy. The user's body of work is internalized in the weights rather than retrieved at inference time, so it works with no network. Its loop is closed: the user authors an artifact, the artifact is recorded in the lineage field, a corpus assembly step derives an admissible training set from lineage, a parameter-efficient fine-tune (the spec names low-rank adaptation, prefix tuning, prompt tuning, and adapter training) produces an updated artifact, and a governed substitution promotes it in the registry, all while the agent's identity, cognitive state, and lineage are preserved.

Now the sync. Two mechanisms in the disclosure do the connected-when-available work, and they are deliberately different from weight-averaging.

Federation coordinates two or more of the user's substrate devices under a federation policy. Crucially, federation exchanges lineage records and does not require exchange of model artifacts; each device keeps local responsibility for its own models and runs its own lifecycle operations. Sharing lineage lets each device fold outcome signals seen elsewhere into its own routing and its own corpus assembly. The federation layer can maintain a federated agent identity record that verifies, through cross-device attestations, that the federated agents correspond to a single user, so events across devices are treated as originating from one agent identity, preserved across device additions, retirements, and hardware refresh. Because two devices editing offline will eventually disagree, the disclosure includes conflict resolution: last-writer-wins where policy permits, merge of non-overlapping changes, quorum review, or escalation to the user for adjudication, with inputs and outcomes recorded in each device's lineage.

Cloud-burst forwarding handles the case where local endpoints lack capability or capacity. Under a cloud-burst policy, a request may be forwarded to a remote endpoint, but only after an admissibility test: a capability test (does any local endpoint satisfy the request), a capacity test (can local compute meet the latency budget), a disclosure test (are the inputs admissible for off-device disclosure), and a cost test. Forwarded

payloads are treated as off-device disclosure events, evaluated against the disclosure policy, and recorded in lineage. For offline-first behavior specifically, the disclosure describes a deferred forwarding mode: requests are queued for forwarding on a subsequent connectivity event while the agent operates in a degraded mode and returns partial or surrogate responses in the meantime. That is your "syncs when connected" path for anything that genuinely needs the cloud.

Two supporting pieces make this trustworthy. The persistent identity field can be cryptographically bound to a hardware security element (secure enclave, TPM, HSM, or embedded secure element), so the agent's identity is anchored to the device and not freely transferable except under a governed migration operation. And the privacy invariant holds that lineage records, model artifacts, corpora, corpus-model parameters, and counterparty records are not transmitted off-device except under an explicit disclosure policy, enforceable by an egress filter, per-component isolation, or key-release preconditions, with every disclosure and every denial logged.

How to Approach the Build

A workable order of implementation, following the disclosure:

1. **Define the agent as the persistent entity, not the model.** Stand up the four persistent fields first: identity, cognitive state, an append-only lineage log, and a governance policy store. Everything else is subordinate. Resist the urge to make "the model" your top-level object; that is the mistake the spec is designed to avoid.
2. **Make lineage append-only and chained from day one.** Each record references its predecessor so the whole history is reconstructible and tamper-evident. You will lean on this for routing, for corpus assembly, and for reconciliation, so retrofitting it later is painful. An illustrative record shape (not an API you install):

```
LineageRecord {
  prev_hash, timestamp, scope_id,
  kind: dispatch | outcome | lifecycle | ingestion |
        policy | encounter | disclosure | federation,
  payload_descriptor
}
```

This sketch is illustrative and faithful to the disclosed fields; you design the concrete encoding.

3. **Build the tool registry and a dispatcher.** Register each endpoint with a model artifact, an interface spec, a governance scope, and a capability declaration (modality, task category, resource envelope). Route requests by matching input modality and task category to capability declarations, then bias toward endpoints with better historical outcomes recorded in lineage.
4. **Add the lifecycle controller with staged, atomic substitution.** Retrain and substitute in a staging area, promote only on successful policy validation, and roll back on failure with the cause recorded. This is what lets you swap or retrain models without disturbing agent identity.
5. **Implement the personal corpus model loop.** Wire authoring to lineage, lineage to corpus assembly (filtered by an admissibility policy and optionally by scope), corpus to a parameter-efficient fine-tune, and the result back through governed substitution. Schedule retraining for idle or power-surplus windows so foreground inference is never blocked.
6. **Layer scopes over one identity.** The disclosure partitions the agent into named scopes (professional, personal, project, household) each with its own corpus policy, tool subset, and lineage partition, under one persistent identity. Carry a scope identifier on every lineage record so per-scope views are just filters over the unified log.

7. **Add federation as lineage exchange.** Exchange lineage records, not weights, under a federation policy. Implement conflict resolution explicitly (last-writer-wins, merge, quorum, or user escalation) because offline edits on two devices will collide. Log every federation and conflict outcome.
8. **Add cloud-burst last, behind the four-part admissibility test.** Gate any forwarding on capability, capacity, disclosure, and cost, and implement the deferred mode so offline requests queue and drain on reconnect while you serve degraded responses locally.
9. **Enforce the privacy invariant at the egress boundary.** Treat federation, ingestion, cloud-burst, and encounters as the only paths that can leave the device, and route each through disclosure-policy evaluation with a logged allow or deny.

What This Does Not Give You

This is an architecture, not a drop-in library. There is no package to install, no SDK, and nothing here "just works" out of the box; you implement each subsystem yourself and make your own choices for encodings, storage, crypto primitives, and scheduling. The disclosure describes the design, not a benchmarked or productized system, and it states no latency, accuracy, model-size, or throughput numbers, so do not expect performance guarantees from it and do not infer any.

It also does not remove the hard parts. You still need a real fine-tuning pipeline, real on-device model artifacts sized to your device envelope, and a real conflict-resolution policy that fits your app. Federation exchanges lineage, so the value of cross-device sync depends on how well your outcome signals and admissibility rules are designed. And this pattern is aimed at personal and edge substrates with a bounded local compute envelope; if your use case is a shared multi-tenant cloud service with no per-user persistence requirement, most of this structure is overhead you do not need.

Disclosure Scope

The approach described here is disclosed in U.S. Provisional Application No. 64/070,239, an agent-resident execution substrate with a governed inference tool registry and lineage-derived personal corpus model training. This guide is educational: it explains the disclosed architecture so a developer can build their own implementation. It is not a warranty, a specification of a shipping product, or an offer of software, and nothing in it should be read as a performance claim or a guarantee of fitness for any purpose. Claims about how the approach works are drawn from that filing; where you extend beyond it, those are your design decisions, not statements of the disclosure.

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)
- Personal-Model Personalization: A User's Own Corpus-Internalized Model on the Agent-Resident Execution Substrate (/articles/agent-resident-execution-substrate/personal-model-personalization)
- On-Device Agent Identity for Robots and Autonomous Vehicles: An Auditable Substrate for Embodied Physical-World Agents (/articles/agent-resident-execution-substrate/embodied-physical-world-agents)

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)
- Ollama alternative: from local model runner to a governed agent-resident substrate (/articles/agent-resident-execution-substrate/ollama)

- Apple Intelligence (on-device foundation models, Private Cloud Compute) vs a persistent agent-resident substrate: who owns identity, lineage, and the model? (/articles/agent-resident-execution-substrate/apple-intelligence)

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