Nick Clark · Published May 20, 2026 · PDF

The Architecture, In Full

The preceding articles in The Case make the argument in the negative space — why existing systems cannot be governed at scale, why safety cannot be supervised into existence, why physical-world and enterprise autonomy alike need governance built into the substrate. This is the positive disclosure: the execution substrate itself, in full. A semantic agent operates as the persistent execution authority of a device; inference endpoints, knowledge sources, and adapters are governed managed assets subordinate to it; the agent's identity, cognitive state, and lineage persist across every change to the components beneath; and a personal corpus model trains on the user's own accumulated work from real-world lineage rather than recursive output. Formally an Agent-Resident Execution Substrate with Governed Inference Tool Registry and Lineage-Derived Personal Corpus Model Training — the subject of U.S. Provisional Patent Application Serial No. 64/070,239, composing with ten prior-filed applications across cognition, identity, network, content, perception, and governance.

This is the closing piece of *The Case*. The preceding articles establish, from first principles, why governance, identity, and admissibility cannot be supervised, moderated, or audited into existence after the fact; they must live inside the thing being operated on. What follows is the architecture that does exactly that, disclosed in full:

the agent-resident execution substrate at the center of the portfolio, where the agent is the durable thing and every model, tool, and knowledge source beneath it is a governed managed asset.

1. Introduction

Three architectures dominate the present generation of consumer-grade AI systems. Cloud-hosted large language model services rent inference at a centralized endpoint, with the model trained against aggregated public and licensed data and undifferentiated by user identity. On-device monolithic foundation models, exemplified by Apple Intelligence in its current shipping form, provide local inference but treat the model itself as the system, with task-specific adapters or prompts substituting for any persistent agent layer. Tool-using agent frameworks (the LangChain, AutoGen, OpenAI Assistants, and recently proliferating "Agent OS" ecosystem) provide dispatch and memory orchestration but treat the agent as an ephemeral session or a stateless dispatch function, with no architectural preservation of agent identity across the components beneath.

Each of these architectures fails a different load-bearing test that consumer-grade local AI must pass. Cloud inference cannot keep the user's data on the user's device, cannot scale economically at the per-user level once usage is non-trivial, and structurally subordinates the original equipment manufacturer to whoever owns the cloud. Monolithic on-device foundation models cannot specialize per-user efficiently without manual fine-tuning operations decoupled from authoring activity, cannot safely retrain on user output because recursive self-training collapses the model's output distribution, and cannot preserve any user-facing entity across the model upgrade cycle. Tool-using agent frameworks cannot guarantee the agent the user has built relationship with today is the same agent tomorrow, because the framework imposes no constraint on what changes when the underlying model is replaced, the policy is updated, or the framework itself is upgraded.

This paper describes a fourth architecture. The semantic agent is the durable thing: models, knowledge sources, adapters, and inference endpoints are managed assets the agent governs, and identity, cognitive state, and accumulated lineage persist across every change to the components beneath. The agent owns its tools and trains them from real-world lineage rather than recursive output. The architecture is structurally different from each of the three incumbent approaches, and it is the convergent answer to the question of how to ship local AI that becomes the user's own over time.

Section 2 establishes the three incumbent architectures and their structural failure modes in detail. Section 3 describes the substrate architecture at the primitive level. Section 4 develops the personal corpus model embodiment that operationalizes "AI that becomes yours." Section 5 addresses privacy, federated identity, and hardware-anchored continuity. Section 6 covers the scale-invariant deployment configurations from consumer through industrial. Section 7 audits the prior art systematically and identifies the structural distinctions that defeat anticipation and obviousness rejections. Section 8 enumerates the claim coverage and portfolio composition. Section 9 maps the architecture against the publicly disclosed product directions of Apple, Tesla, and robotic system OEMs to show why each will need either a license or a materially weaker product. Section 10 closes.

2. The Three Incumbent Architectures and Their Failure Modes

2.1 Cloud-hosted inference

Cloud-hosted large language model services (OpenAI's hosted GPT family, Anthropic's hosted Claude family, Google's hosted Gemini family, and the various enterprise wrappers thereof) provide inference at a network endpoint operated by the model provider. The model parameters are not on the user's device. The model is trained against aggregate public and licensed datasets and is not specialized per user. Per-query inference cost is borne by the provider and recovered through subscription pricing or

per-call usage fees. Personalization, where offered, takes the form of retrieval-augmented prompt context (Mem.ai, ChatGPT memory, Claude Projects, NotebookLM) or system-prompt configuration (custom GPTs).

This architecture fails three load-bearing tests for consumer-grade local AI. First, the user's data must traverse the network and be processed under the provider's terms; regulatory environments increasingly disfavor this disposition, and consumer privacy expectations have moved against it. Second, per-query cost does not scale to the use cases that make local AI valuable (continuous background inference, fine-grained personalization, multimodal sensor fusion) without unacceptable economics or hard rate-limiting that erodes the product story. Third, an original equipment manufacturer shipping a consumer product on top of cloud inference is structurally a reseller of someone else's substrate; the OEM does not own the customer relationship at the inference layer, and the model provider can revoke, reprice, or compete at any time.

Cloud inference will remain dominant for frontier-scale generative tasks for the foreseeable future. It is not, however, the architecture that the next generation of consumer-grade local AI products will be built on. Apple has signaled this with Apple Intelligence and Private Cloud Compute, Tesla with on-vehicle autonomy, and Meta with on-headset Quest models, while the robot OEM landscape signals it implicitly by physical necessity.

2.2 Monolithic on-device foundation models

The current shipping form of Apple Intelligence is representative of the second architecture. A small or medium foundation model runs locally on the device, and per-task behavior is configured by parameter-efficient adapters loaded per task. The model has no persistent agent identity; it is the system. When the foundation model is updated, the user's experience updates with it; there is no continuous entity that preserves what the user has accumulated through the previous version.

This architecture solves the data residency and latency problems of cloud inference but introduces three new failure modes. First, the model cannot efficiently specialize per user. Adapter swap is task-scoped, not user-scoped; large-scale fine-tuning at the per-user level is impractical for a single monolithic model serving multiple tasks. Second, the model cannot safely retrain on its own output. Self-rewarding language model approaches and reinforcement-learning-from-AI-feedback approaches are now known to collapse the model's output distribution after a small number of iterations on its own generations, because the training signal does not contain new information about the world. Third, the model cannot preserve identity across upgrades, because identity is not architecturally separable from the model. When the model is replaced, there is no agent to preserve.

Monolithic on-device foundation models will remain a useful component of local AI products, but they cannot be the architecture; they are inference primitives. The architecture above them, the part that owns identity, preserves continuity across upgrades, and operates per user, is the substrate. That layer does not exist in current monolithic shipping products.

2.3 Tool-using agent frameworks

The third incumbent architecture is the tool-using agent framework. LangChain, AutoGen, the OpenAI Assistants and GPTs platforms, Anthropic's tool-use SDK, the Rivet agent runtime, the buildermethods.com Cursor workflow, the PwC enterprise consulting practice marketed as "Agent OS," and the Sierra "Agent OS 2.0" customer-service SaaS all sit in this category. Each provides some combination of model orchestration, tool dispatch, conversational memory, and prompt template management.

These frameworks treat the agent as either an ephemeral session that exists for the duration of a conversation, or a stateless dispatch function that runs in response to inputs. None of them claim or enforce that the agent persists across sessions in a manner that accumulates lineage; none impose an invariant that lifecycle operations on

the underlying components do not modify the agent's identity or cognitive state; none condition routing on the agent's persistent internal cognitive state; none own the trained tools they invoke as managed assets subject to governed lifecycle operations.

The frameworks have product-market fit because tool dispatch and orchestration are real engineering problems. But they are infrastructure for running agents, not architectures for what agents are. When the model under them updates, the agent does not survive; a new agent runs in its place, with the same configuration but no continuity to anything the previous agent accumulated. This is the gap the substrate fills.

3. The Substrate Architecture

3.1 The persistent semantic agent

The substrate's keystone primitive is a semantic agent instantiated on the device at first power-on and maintained as a persistent computational entity across the lifetime of the device. The agent comprises four persistent fields. The identity field carries one or more identifier values and continuity metadata sufficient to verify the agent's continuity across substrate restart events; in hardware-equipped devices, the identity field is cryptographically bound to a secure enclave, trusted platform module, or other hardware security element such that the identity is not transferable to a device lacking access to the bound hardware. The cognitive state field carries persistent cognitive domain values: affective state, normative integrity, execution confidence, capability awareness, forecasting state, and any other domain values declared in the agent's cognitive schema. The lineage field is an append-only sequence of records encoding the agent's complete operational history. The governance policy field carries the cryptographically signed policy objects governing the agent's admissible operations.

The architecture enforces a structural invariant: no operation performed by any subordinate component of the substrate modifies the agent's identity field, cognitive state field, or lineage field, except by appending to the lineage field under a continuity

proof. Tool lifecycle operations, governance policy updates, substrate-runtime updates, knowledge ingestion events, and federation events are all governed events that write to the lineage field as deterministic records but do not modify the agent's identity or cognitive state. The agent's behavioral continuity is therefore independent of every component beneath it.

3.2 The managed inference tool registry

The agent owns a tool registry, a local data structure recording the managed inference endpoints currently installed within the substrate. Each managed inference endpoint comprises a model artifact, an interface specification, and an associated governance scope. Endpoints are heterogeneous in type: general-purpose language models, task-specific fine-tuned language models, image classifiers, speech recognition models, embedding models, retrieval models, vision models, and personal corpus models. Adapter-based variants in which a base model is shared across multiple endpoints and per-endpoint adapter weights specialize the base model are supported as a particular memory-efficient configuration.

The registry is subject to a set of governed lifecycle operations (install, retrain, replace, archive, remove, quiesce, resume), each evaluated against the governance policy, executed by a dedicated tool lifecycle controller, and recorded in the lineage field. Atomic substitution semantics ensure that an interrupted retraining or replacement does not leave the registry in an inconsistent state. Each lifecycle operation records the policy object version under which it was admitted, such that the admissibility of any prior operation is reproducible from the lineage.

3.3 The agent-to-tool dispatcher

A dispatcher routes inference requests from the semantic agent, and from host applications calling through the application interface, to managed inference endpoints in the registry. The routing decision is conditioned on the agent's current cognitive state, the input characteristics of the request, the policy objects applicable to the

request, and the per-endpoint capability declaration. Candidate endpoints whose declared confidence, normative compliance, or capability fall below the applicable thresholds derived from the cognitive state are excluded from the routing decision; among the remaining candidates, the dispatcher prefers endpoints with higher historical outcome quality recorded in the per-endpoint sub-lineage. Concurrent multi-endpoint dispatch with output aggregation, and serial multi-stage pipelines composing specialized endpoints, are supported as alternative configurations.

Cognitive-state conditioning of dispatch is one of the architecture's load-bearing novelties. None of the incumbent gateway frameworks route based on the agent's persistent internal state, because none of them have a persistent internal state to condition on, which makes the substrate's dispatcher structurally different.

3.4 Lineage as training signal

For each inference request, the lineage controller appends a lineage record comprising the input descriptor, the endpoint identifier, the output descriptor, a timestamp, and a downstream-outcome reference. The downstream-outcome reference points to one or more subsequent events evaluating the inference output: user acceptance, user revision, downstream execution success, downstream execution failure, and integrity-signal feedback from coupled cognitive domain fields. Integrity-signal feedback comprises evaluations of the inference output against the normative integrity domain and other cognitive domains, including detection of deviation from policy-declared normative thresholds, detection of inconsistency with prior agent outputs, and detection of structural mismatch between the inference output and the user's accumulated structural conventions.

The lineage-derived training signal is structurally distinct from training signals derived from model self-output. Training corpora derived from a model's own prior outputs, such as reinforcement-learning-from-AI-feedback and self-rewarding language model approaches, are known to exhibit distributional collapse over successive training iterations when the signal is dominated by recursive model output. The lineage signal in

the disclosed substrate comprises downstream-outcome data reflecting real-world acceptance, revision, execution success or failure, and integrity feedback, which together encode information not generated by the model under training. The model collapse failure mode does not apply because the signal does not come from the model.

3.5 Identity preservation across tool lifecycle events

The architecture's keystone invariant is that the agent's identity, cognitive state, and lineage are preserved across every lifecycle operation applied to subordinate components. The continuity guarantee is enforced through structural separation between the agent's persistent state and the state of subordinate components, through cryptographic continuity proofs over the lineage field, or through a combination thereof. Lifecycle operations write to the tool registry, to staging areas, and to the lineage field as appends; they do not modify the agent's identity, cognitive state, or any prior lineage record. The lineage field is append-only under a continuity proof, with each record carrying a cryptographic reference to its predecessor.

The continuity guarantee extends across substrate-runtime updates. The substrate runtime itself (the agent runtime, the dispatcher, the tool lifecycle controller, the governance enforcement subsystem, the resource governance subsystem, and related substrate-internal components) is updatable through governed substrate-runtime update operations, with the agent's identity preserved across the update through a continuity attestation cryptographically chained over the sequence of substrate-runtime versions. The agent is the same agent before and after a runtime upgrade.

4. The Personal Corpus Model

The substrate's most visible embodiment is the personal corpus model: a managed inference endpoint whose parameters are trained, fine-tuned, or adapter-modified against a training corpus derived from artifacts authored, curated, or designated by the user under the user's governance policy. The personal corpus model is configured to

perform inference reflective of the user's domain knowledge, terminology, structural conventions, and prior outputs, without retrieval over the body of user artifacts at inference time. The user's accumulated work is internalized in the parameter values rather than consulted as external context.

The closed-loop architecture works as follows. The user authors an artifact through user-facing applications operable on the substrate device, including text editors, code editors, content management interfaces, and recording interfaces. Each authored artifact is registered in the lineage field under the user's governance policy. The corpus assembly module periodically derives an incremental training corpus by selecting artifacts admissible under the corpus policy, filtering for modality compatibility with the personal corpus model, and applying any declared redaction or anonymization rules. The fine-tuning module applies a parameter-efficient fine-tuning operation (low-rank adaptation, prefix tuning, prompt tuning, continuous adapter training, or any other parameter-efficient method) to produce an updated model artifact. The governed substitution module performs a lifecycle replacement in which the updated artifact is promoted to the active state in the tool registry. Subsequent inference requests from host applications operated by the user are routed by the dispatcher to the updated personal corpus model, and the outputs of the model assist the user's subsequent authoring activity. The artifacts produced thereunder enter the lineage field to feed the next iteration of the loop.

The personal corpus model differs structurally from retrieval-augmented generation. Retrieval-augmented generation systems store information for lookup at inference time; the model has no internalized representation of the user's corpus. The accuracy of retrieval-augmented systems depends on retrieval quality, chunk-boundary effects, and the base model's ability to integrate retrieved context. The personal corpus model has no such failure modes because the user's accumulated work is in the weights, not in an index.

The personal corpus model also differs structurally from user-initiated fine-tuning services. Such services require manual corpus curation, manual initiation of training operations, and manual deployment of resulting artifacts. The substrate's personal corpus model is agent-curated, automatically retrained under policy, and substituted into the active registry under governed lifecycle operations, so the agent manages the training pipeline rather than the user.

The personal corpus model is one of a plurality of corpus models that may be co-resident in the tool registry. A first corpus model may be fine-tuned against the user's authored prose in a professional scope; a second against the user's authored source code in a project-specific scope; a third against the user's designated professional publications; a fourth against the user's media library. The dispatcher selects among corpus models based on input modality, task category, and the active-scope indicator carried in the cognitive state field.

5. Privacy, Federated Identity, and Continuity

5.1 Privacy as architectural invariant

The substrate enforces a privacy invariant: lineage records, model artifacts, training corpora, personal corpus model parameters, scope-local context store contents, and counterparty identity records are not transmitted off the substrate device except under an explicit disclosure policy object within the governance policy field. The disclosure policy identifies a recipient, a scope of permitted disclosure, an authorization attestation, a retention requirement, and a revocation mechanism. Each off-device disclosure event is recorded in the lineage field as a deterministic disclosure event, comprising the policy identifier, the recipient, the categories of disclosed material, an enumeration of specific records or artifacts disclosed in summary form, the authorization attestation, and a timestamp. Disclosure lineage is verifiable by any party with access to the lineage field, permitting the user or a regulatory authority to audit the complete record of off-device disclosures originating from the substrate.

Enforcement mechanisms include a substrate-runtime egress filter intercepting outbound network traffic, per-component isolation preventing subordinate components from initiating off-device transmissions absent dispatch through the egress filter, signed disclosure-policy preconditions verified before any encryption key required for off-device transmission is released, and hardware-anchored attestation that the substrate runtime mediating the disclosure has not been tampered with. The privacy invariant is operative regardless of network connectivity; inference operations performed entirely on the substrate device do not constitute off-device disclosure.

5.2 Federated unified identity across devices

A user typically operates a population of devices: phone, tablet, laptop, desktop, head-mounted display, vehicle, and household appliance. The substrate's federation layer maintains a federated agent identity record corresponding to the agent identity field of two or more federated substrate devices, verifying through cross-device attestations that the federated agents correspond to a single user identity. The federation layer treats inference requests, lineage records, scope mutations, counterparty encounters, and policy events across the federated devices as originating from a single agent identity.

Federation is structurally distinct from federated learning. Federated learning aggregates weight updates across devices to produce a shared model; it operates at the model layer and does not produce a federated agent identity. Federation as disclosed herein operates at the agent layer: each device retains its own local model artifacts and performs its own lifecycle operations, while the federation layer exchanges lineage records and governed model updates under federation policy. The federated agent identity is preserved across device additions, device retirements, and device hardware refresh within the user's device population.

Federation is also structurally distinct from cloud-account-based device association. Cloud-account synchronization associates devices with an account identifier and synchronizes files or settings; it does not unify a persistent agent identity verified

through cross-device attestations. The disclosed federation produces agent identity continuity that spans devices and is independent of any specific model weights or cloud account.

5.3 Hardware-anchored identity binding

Where the substrate device provides a hardware security element (secure enclave, trusted platform module, hardware security module, or embedded secure element), the persistent identity field is cryptographically bound to it through a key-derivation operation incorporating values derived from the hardware element. The agent's identity is not transferable to a device lacking access to the bound hardware element except under a governed migration operation in which both the originating and destination hardware elements attest to the migration, the destination derives a new binding under a migration policy object, and the migration event is recorded in the lineage field of both devices. The substrate quarantines agent execution if the hardware element fails attestation or is detected as compromised.

5.4 Counterparty identity records

Beyond the user's own identity, the substrate maintains counterparty identity records corresponding to entities encountered by the agent: other human users, peer agents on other substrate devices, devices interacting with the substrate, and organizational entities transacting with the substrate. Each counterparty identity record comprises a counterparty identifier, a counterparty scope object specifying admissible interactions, an encounter history within the lineage field, a persistence designation (ephemeral, persistent, or promoted-from-ephemeral), and one or more attestation references verifying the counterparty's claimed identity. Promotion of an ephemeral encounter to a persistent counterparty is governed by a promotion policy object that may condition promotion on explicit user authorization, interaction frequency or duration thresholds, or policy-declared event categories.

Counterparty handling scales to substantial encountered populations through bounded retention of ephemeral records, summarization of long-tail counterparty histories, and federation of counterparty records across substrate devices. The architecture explicitly contemplates deployment configurations in which the substrate encounters thousands or millions of counterparties over its operational lifetime: household robots that meet visitors, vehicles that interact with other vehicles and pedestrians, and enterprise systems that transact with clients, vendors, and employees.

6. Scale-Invariant Deployment

The substrate architecture is scale-invariant. The defining property of "local" in this architecture is "local to a single operational entity" rather than "small": one user, one household, one team, one factory, one fleet, or one organizational tenant. The same primitive set runs across personal computing devices, multi-user workstations, enterprise servers, industrial and operational technology installations, embedded controllers, robotic devices, vehicular computing devices, household appliances, wearable devices, and federated multi-device configurations.

In a personal computing configuration, the substrate is instantiated on consumer-class hardware (desktop, laptop, miniature desktop, tablet, or high-end mobile) with the user's governance policy as the dominant policy authority. In a workstation configuration, multiple authorized users each maintain independent agent identities and personal corpus models under a shared workstation policy. In an enterprise server configuration, the substrate operates under organizational governance with stricter constraints on ingestion sources, training corpus composition, and inference logging. In an industrial or operational technology configuration, the substrate operates local to a factory production line, refinery, distribution center, warehouse, power facility, treatment plant, mining operation, hospital, laboratory, or agricultural operation, with operational policy as the user policy and counterparty records corresponding to operators, suppliers, customers, regulatory inspectors, and equipment endpoints.

In robotic and vehicular configurations, the substrate integrates with physical-world perception and actuation primitives (published separately in U.S. Provisional Patent Application Serial No. 64/049,409) through multimodal input adapters and effector output adapters. Sensor inputs from perception adapters become inference requests dispatched through the agent-to-tool dispatcher; inference outputs become governed side effects supplied to actuation primitives under the substrate's governance policy field. Counterparty records correspond to humans, peer devices, vehicles, pedestrians, and other entities encountered during operation. Resource governance budgets are configured to respect thermal, power, real-time-response, and safety-critical constraints.

In federated multi-device configurations, two or more substrate devices operate under a federation policy declared at the user, household, or organizational scope, with lineage exchange, optional model artifact propagation, and federated agent identity maintained across the device population.

7. Prior Art Audit

We audit the principal prior art systematically. None of the references combines the architectural elements of the disclosed substrate, and the structural distinctions that defeat anticipation and obviousness rejections are the same distinctions that determine product outcome in the local-AI category.

Ollama, llama.cpp, LM Studio, and similar local model registries provide tool registries and lifecycle utilities but have no agent, no persistent identity, no governance policy field, no preservation invariant, and no cognitive-state-conditioned dispatch. They are infrastructure for running local models, not architectures for what runs them.

LangChain, AutoGen, OpenAI Assistants, Anthropic tool use, Rivet, and the broader tool-using agent ecosystem provide dispatch and memory but no persistent agent identity in the architectural sense, no tool lifecycle controller as a first-

class subsystem, and no invariant prohibiting modification of agent state by lifecycle operations. Agents in these frameworks are sessions or stateless dispatch functions.

Apple Intelligence ships a foundation model with task-specific adapters. The model is the system; there is no agent atop the model. Adapters are configurations of the same model, not subordinate inference endpoints under an agent. The identity-preservation invariant does not apply where there is no separable agent identity to preserve.

NVIDIA Voyager is the closest published research neighbor. Voyager runs an agent in Minecraft that accumulates a code-skill library through execution feedback. Voyager's skills are GPT-generated Python code, not trained model artifacts. There is no tool lifecycle, no managed inference endpoint as a model-artifact-bearing object, and no preservation invariant across model substitution because there is no model substitution in Voyager.

DSPy compiles large language model modules. It is a programming abstraction over inference calls, not an agent runtime.

Toolformer trains a model to call tools. The tools are static; no tool lifecycle is involved.

Self-Rewarding Language Models, Constitutional AI, and reinforcement-learning-from-AI-feedback approaches are model-internal self-improvement mechanisms that operate on a single monolithic model. They use recursive model output as training signal, which is known to cause distributional collapse, and have no tool registry, no agent layer, and no preservation invariant.

Federated learning frameworks such as Flower, OpenFL, and FedML aggregate weight updates across devices. They have no agent identity, no federated agent identity record, and no architectural concept of "the same agent operating on a different substrate."

Mem.ai, Rewind.ai, Limitless, ChatGPT memory, Claude Projects, NotebookLM, custom GPTs are memory and retrieval layers. They do not modify model parameters; they do not have persistent agent identity at the architectural level; they do not have tool lifecycle; they do not have a preservation invariant.

The "Agent OS" product category, including buildermethods.com/agent-os (a Cursor workflow template), the PwC consulting practice marketed as Agent OS, the Sierra.ai customer-service SaaS marketed as Agent OS 2.0, and the various academic and developer-framework usages of the term, comprises heterogeneous products and services that share a marketing label without sharing a structural architecture. None of them ship a persistent agent identity that survives model substitution under a governed tool registry with a privacy invariant.

The disclosed architecture combines (i) persistent agent identity with structural separation from subordinate component state, (ii) tool registry as managed-asset class subject to governed lifecycle operations, (iii) cognitive-state-conditioned dispatch, (iv) identity-preservation invariant across all lifecycle and substrate-runtime events, (v) lineage-derived training signal sourced from real-world outcomes rather than recursive model output, and (vi) personal corpus model as parameter-resident user knowledge. No prior reference combines these elements, and the architectural invariant in particular is unmotivated by any combination of references.

8. Claim Coverage and Portfolio Composition

The architecture is the subject of U.S. Provisional Patent Application Serial No. 64/070,239, filed May 20, 2026, titled "Agent-Resident Execution Substrate with Governed Inference Tool Registry and Lineage-Derived Personal Corpus Model Training." The disclosure includes three independent claims and thirty-six dependent claims spanning the architectural primitives described above. The independent claims cover (i) the substrate-architecture system, (ii) a method for autonomous tool improvement on the system, and (iii) the personal corpus model embodiment.

The substrate composes with a portfolio of ten prior-filed applications covering related primitives. The semantic agent operating as the persistent execution substrate may be implemented as a persistent executable object as described in U.S. Application 19/538,221, and may be structured as a semantic agent object as described in U.S. Application 19/452,651. The cognitive state field may comprise persistent cognitive domain fields and a cross-domain coherence engine as described in U.S. Application 19/647,395. The governance policy field may comprise cryptographically signed policy objects as described in U.S. Application 19/561,229. The persistent identity field may comprise dynamic agent hashes, dynamic device hashes, continuity-based biological identifiers, or hardware-anchored identifiers as described in U.S. Application 19/388,580. The substrate further integrates the cognition-native semantic execution platform of U.S. Application 19/230,933, the adaptive network framework of U.S. Application 19/326,036, and the cognition-compatible network substrate and memory-native protocol stack of U.S. Application 19/366,760 for inter-substrate coordination. Physical-world perception and actuation primitives suitable for robotic and vehicular deployments are described in U.S. Provisional Application 64/049,409. Compensation routing primitives applicable to publisher and creator economies on top of the substrate are described in U.S. Provisional Application 63/808,372 and the related PCT International Application PCT/US26/28630.

Each filing claims a primitive; the substrate claims the integration architecture that lets them compose into a deployable product on a device. A competitor wishing to ship a consumer-grade local AI product without licensing the substrate must either (i) avoid the substrate's architectural elements, accepting a materially worse product, or (ii) build their own substitutes for each of the underlying primitives the substrate composes, a multi-patent design-around exercise across ten separate inventive concepts.

9. Relevance to OEM Local-AI Roadmaps

The substrate architecture maps directly to the publicly disclosed product directions of several major original equipment manufacturers building local AI.

Apple. Apple Intelligence today ships a monolithic on-device foundation model with task-specific adapters and Private Cloud Compute as an off-device burst path. The next architectural step that consumer expectation will demand is the substrate: a persistent on-device agent that learns the user across sessions, survives foundation-model upgrades, integrates with third-party applications under governed admission policy, and federates across the user's iPhone, iPad, Mac, Apple Watch, and Apple Vision Pro under a single identity. Hardware-anchored identity binding maps directly to Apple's Secure Enclave. The application interface and host application admission policy map to Apple's app distribution model. The federated unified user identity maps to Apple's existing Apple ID account model extended to the agent layer. Apple's stated commitment to on-device privacy maps to the substrate's privacy invariant as architectural property.

Tesla and automotive OEMs. An in-vehicle agent must preserve identity across software updates across the lifetime of the vehicle, must train tools (driving models, voice assistants, scene understanding models) from accumulated driving lineage rather than recursive model output, must encounter and maintain records of repeat counterparties (frequently-encountered vehicles, pedestrians at the driver's home and work, household members), and must respect real-time, thermal, and power constraints. The substrate's claims include vehicular deployment configurations, sensor-coupled scope triggers, multimodal perception and actuation adapters, counterparty identity records at population scale, and resource governance budgets configured for the deployment context. The architectural fit is direct.

Robot OEMs. Optimus, Figure, Apptronik, 1X, and the broader humanoid and service-robot ecosystem face a stronger version of the automotive challenge: an embodied agent that encounters hundreds or thousands of humans over its operational lifetime, must

maintain governed counterparty records, must integrate sensor input and physical actuation under safety-critical resource constraints, must update its trained tools as it accumulates operational lineage, and must federate across multi-robot fleets under operator policy. The substrate provides each of these as a named architectural primitive.

Meta and head-mounted display OEMs. Quest, Vision Pro, and the broader extended-reality device category face the multi-tier hardware unification challenge: a single user's agent operating across phone, headset, and desktop with continuous identity. The substrate's federation layer and cross-tier identity coordination address this directly.

Enterprise and industrial OEMs. Factory floor systems, hospital systems, distribution-center systems, and the broader operational-technology landscape need governed local AI that operates within the trust boundary of a single operational entity, respects regulatory data-residency requirements, and integrates with the operation's existing data infrastructure under policy. The substrate's industrial configuration claim and operational-policy framing address this directly.

In each case, the architectural elements the OEM needs to ship are the architectural elements the substrate claims. The patent does not prevent the OEM from shipping local AI; it makes the architecture under which the OEM must ship subject to licensing or design-around.

10. Conclusion

Adaptive Query's substrate architecture is the convergent answer to the question every original equipment manufacturer shipping consumer-grade local AI is asking: how does the product become the user's own over time, preserve continuity across model upgrades, and respect privacy as an architectural property rather than a marketing claim? The architecture is the layer above the foundation models, above the

orchestration frameworks, and above the cloud burst paths: the layer that owns identity, owns the tools, owns the lineage, and survives every change in the components beneath.

The architecture is now patent-pending under USPTO Provisional Application Serial No. 64/070,239 and composes with a portfolio of ten prior-filed applications covering the primitives the substrate integrates. The disclosure is published openly here to support technical-audience review, prior-art establishment, and serious partnership discussion. Specific implementation choices, performance optimizations, and several architectural enhancements are retained as trade secret or are the subject of separate continuing filings.

For organizations building, evaluating, or partnering on local AI infrastructure: the architecture is available to license. The portfolio is available to evaluate. The technical conversation is open. We invite engagement from sophisticated technical and business audiences at the original equipment manufacturer, automotive, robotics, enterprise infrastructure, and consumer device levels.

Adaptive Query is a persistent execution substrate where local, governed, and personalized AI lives, where your agent, your accumulated work, and your data stay with you across every model upgrade and every device.

Read alongside the rest of *The Case*, this is where the argument stops being an argument. The preceding articles establish what governance at scale requires; this one is the filing that supplies it.

Legal Notice

implementation, or distribution requires a separate written license agreement. The disclosure is provided for technical-audience review, prior-art establishment, and partnership engagement purposes.

No Warranty. The disclosure is provided "as is" without warranty of any kind. Performance projections and product fit characterizations are mechanism-grounded analyses, not empirical guarantees. Implementation outcomes depend on engineering choices not disclosed herein.

Contact. Licensing, partnership, and serious technical inquiries: nick@qu3ry.net.