ICOS Report I — Runtime Architecture and Foundations

Robert Vandelinder · Exile Research Inc. · ORCID 0009-0001-9793-420X

Preprint v1.0 — November 2026

Abstract

A language model supplies inference; it does not, by itself, constitute an agent. Between a raw model and a useful persistent agent sits a runtime: the substrate that owns durable state, mediates every capability the model can reach, and keeps a record of what happened. Most "agent frameworks" leave that runtime implicit — a composition of libraries, prompt conventions, and in-process glue whose ownership boundaries are undocumented and whose durable state is scattered. This report presents the runtime architecture of ICOS (**I**SABEL **C**ognitive **O**perating **S**ystem), built as a single small system. It describes and evaluates the foundation in detail — nine milestones plus MCP integration, external channels, dynamic MCP toolsets, and a security-hardening pass — and uses the later layers as evidence that the runtime absorbed them without redesign. It makes three claims. First, a **layered runtime with one provider boundary and explicit state ownership**: every durable artefact has exactly one owner, and the model is one replaceable component behind a single OpenAI-compatible interface. Second, **durable, exactly-once ledgers** for sessions, memory candidates, tool invocations, and promotions form the substrate for recovery and audit — the system can say what it did, not merely what it currently believes. Third, and strongest, **modularity demonstrated rather than asserted**: later layers — identity, drift detection, claim verification, external channels, a broader tool surface, a skill library, dynamic MCP toolsets, a full web client, and subagent delegation — were added **on top of the runtime without redesigning it**. We report the design rationale, the concrete mechanisms and their limits, the evaluation (unit, end-to-end, and live runs; 1,327 unit / 90 e2e / 245 web, recorded at v0.8 on 22 November 2026), and we state plainly what the evaluation does **not** establish.

1. Introduction

There is a persistent confusion in applied agent work between two different things: **inference** and **agency**. A language model is a function from tokens to tokens. An agent is a system that maintains state across time, acts on an environment through tools, is accountable for what it did, and can be changed at one layer without rebuilding the others. The gap between the two is the **runtime** — and it is usually the part nobody specifies.

The prevailing pattern assembles a framework: a chain/graph library for control flow, a vector store for memory, an ad-hoc tool registry, and a prompt that describes the conventions. This works for demos

and becomes an architectural problem at the second or third capability: ownership is unclear (which component owns the durable record?), failure is unaccountable (what happened, and can it be replayed?), and the model boundary is smeared across the codebase. When the underlying model is swapped — a routine event — it is unclear what depends on the old one.

ICOS takes the opposite methodological stance. It is a **from-scratch runtime** in which the agent loop, interaction layer, model boundary, session persistence, memory, skills, tools, and channels are parts of one deliberately small system — *the runtime is the harness*. Each milestone introduces one capability behind a stated question and a committed evidence trail. The design rule is: **if a capability cannot justify the complexity it adds, it does not belong in the architecture yet.**

This is the first report in a series. It has three tiers of scope. The **first tier** — M1–M9, M13, M16, M19, and the security-hardening slices S1–S6 — is described mechanism by mechanism (§5) and evaluated directly (§7). The **later additions** — M14, M15, M15.5, M17, M18, M20, M20.6/M20.7, and M21 — are treated primarily as evidence for the invariance claim (§6); where an addition changed an existing mechanism, §5 notes it in place (M17 in §5.4, M20.7 in §5.6). The **memory and identity reports** — M10–M12 (Report II) and the deeper identity and verification work (Report III) — are referenced where the runtime's ownership map requires them. The report is self-contained for what it evaluates; where it touches a layer owned by a later report, it says so and cites that layer's evidence rather than assuming it.

Throughout, `M#` and `S#` are the project's internal milestone and hardening-slice identifiers; each maps to a plan and a committed verification record in the repository (Appendix A). M10–M12 belong to Report II, which owns their belief semantics (§5.10 covers only the ledger contract this report relies on); M22 belongs to a later report. The deferral is intentional.

Contributions.

- **C1 — A layered runtime reference architecture with one provider boundary and explicit state ownership.** Five layers (interaction, conversation, model boundary, durable state, capabilities), a single OpenAI-compatible provider interface with zero provider branches in the turn loop, and a documented owner for every durable artefact (§4).
- **C2 — Durable, exactly-once ledgers as the substrate for recovery and audit.** Sessions, memory candidates, tool invocations, and promotions each have an append-only, claim-gated ledger; execution is exactly-once across retry, restart, and resume, and the system can reconstruct *what happened* rather than only *what is* (§5.1, §5.5, §5.10).
- **C3 — Modularity demonstrated, not asserted.** The identity, verification, channel, tool, skill, MCP-toolset, web-client, and subagent layers were added **without redesigning the runtime** — the architectural-invariance claim, with committed evidence (§6). This is the report's strongest evidence.

We also report the interaction protocol (commands, approvals, clarifications), skills, the tool surface and its hard limits, MCP integration, external channels, and the security posture, each with its invariants and its measured bounds.

2. Related work

Agent frameworks — LangChain/LangGraph, AutoGen, Semantic Kernel — provide composition primitives (chains, graphs, message passing, tool calling). They are libraries, not runtimes: durable-state ownership, provider independence, and the audit substrate are left to the integrator. ICOS is one coherent runtime with explicit ownership and a committed evidence discipline.

Agent harnesses. The closest contemporary systems are *harnesses* — the runtime that wraps a model and gives it hands: Claude Code, OpenAI's Codex, and LangChain's Deep Agents (permission systems, subagents, context compaction); the minimal **Pi** (pi.dev); **OpenClaw** with NVIDIA NemoClaw/OpenShell; and **Hermes Agent** (Nous Research). ICOS belongs to this camp in spirit — *the runtime is the harness* — and differs on two axes: it is **provider-independent by hard boundary**, and its centre of gravity is the durable epistemic/identity layer that harnesses under-specify. Hermes is additionally ICOS's **lineage and cautionary precedent**: earlier generations were built on it and were destabilised by runaway self-directed change, which is why ICOS makes change deliberate, human-gated, and audited.

Cognitive architectures (Soar, ACT-R) offer unified theories of cognition and are an inspiration for salience/activation and memory tiers; ICOS is LLM-native, provider-independent, and **claims no unified theory of cognition**. **CoALA** (Sumers et al.) is a conceptual taxonomy of language agents; ICOS is an implemented instance of parts of it, with committed evidence. **MemGPT/Letta** manages tiered memory the *model* edits; ICOS keeps change deterministic and human-gated by default. **Generative Agents** (Park et al.) introduced a memory stream with reflection; ICOS adds formal contradiction handling and stored provenance. **RAG** (Lewis et al.) is stateless retrieval; ICOS is stateful, provenance-traced, and lets beliefs evolve. **Event sourcing** and **audit ledgers** (Fowler) are the pattern ICOS applies to sessions, tool calls, and memory.

Novelty claim. Not a new algorithm. The turn loop is a ReAct-style reason–act pattern (Yao et al., 2023); what ICOS adds is the runtime around it. An **implemented architecture** that makes three properties default: explicit state ownership behind one provider boundary, exactly-once durable ledgers, and demonstrated extensibility — with the evidence committed.

---

3. Design goals and non-goals

Goals.

1. **Small enough to understand.** The entire agent loop should be readable without reconstructing a distributed system. The runtime is one NestJS service; the UI is one Angular app.
2. **Provider independent.** The model is one replaceable component behind a single OpenAI-compatible contract; the turn loop contains no provider branch.
3. **Explicit state ownership.** Every durable artefact has exactly one writer. State changes are persisted to SQLite and are inspectable.

4. **Durable, auditable execution.** Actions and beliefs are recorded in append-only ledgers; execution is exactly-once; recovery is a replay, not a guess.
5. **Honest failure.** The system fails closed on malformed provider output and on a missing reference frame, and it degrades rather than failing a turn when an enrichment (recall, extraction) is unavailable.
6. **Human control of consequence.** Anything consequential is approval-gated; the model can propose, but only a human (or an explicit operator policy) can authorise.

Non-goals (explicit).

- **Not a general-purpose framework.** ICOS is a research platform, not a competitor on features.
- **No automatic self-modification.** Nothing in the running system rewrites its own code, its core identity, or its visible output silently. Change is gated and audited (a direct response to the lineage failure mode).
- **No claim to a theory of cognition.** "Cognitive" is the project's long-term vision, not a masthead claim.
- **Not production software.** Interfaces and internals may change.

Runtime assumptions. ICOS assumes a single trusted operator, one host (or one container pair) with a local filesystem for its SQLite stores, and a model endpoint that speaks the OpenAI chat-completions contract. It does not assume network access: a local model server is sufficient, and the security posture in §5.9 is written for that shape. These assumptions are stated because several claims in §6 and §7 are bounded by them — a multi-tenant deployment would change the threat model, and a remote-only store would change the recovery story.

---

4. Architecture

ICOS is five stacked layers. Read top to bottom; each uses the one below.

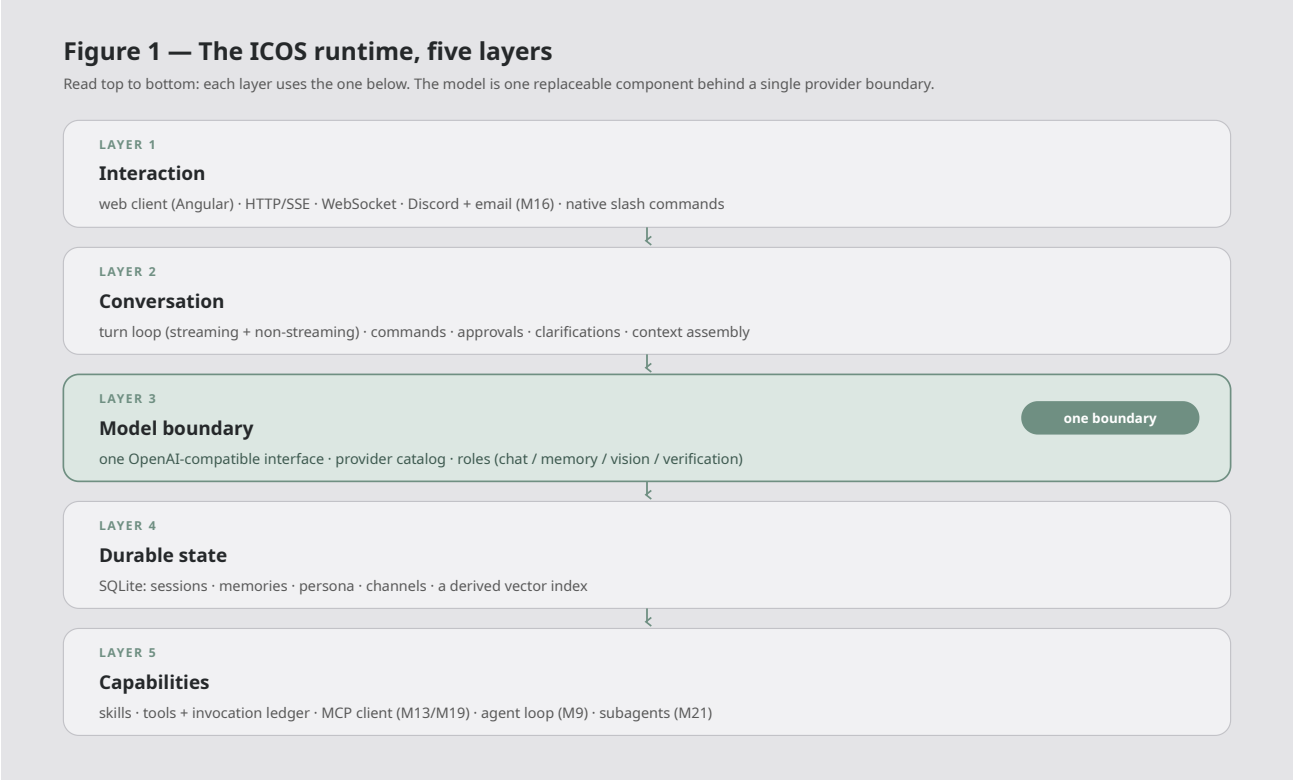


Figure 1 — five layers; each uses the one below. The model is one replaceable component behind a single provider boundary.

The single provider boundary. Every inference call in the conversation path — chat, memory extraction, vision, claim verification — goes through one OpenAI-compatible client (`LlmClient`). The active endpoint is chosen by `LLM_*` environment values or a runtime **provider catalog** (`providers.json`) whose key values are *references* (`$VAR` or `secret:NAME`), never literals. The turn loop contains **zero provider branches**; a static grep against the loop's source is recorded in the M5 evidence, not asserted. Swapping a local Ollama model for a cloud endpoint is therefore a configuration change, not a code change. Two narrower exceptions sit outside the loop and are named here so the claim is checkable: image generation has its own explicitly configured OpenAI-compatible backend selector, and context-budget probing knows the Ollama metadata shape. Neither participates in the loop's decisions.

State ownership. The backbone of the architecture is that every durable artefact has exactly one owner and one writer:

Layer	Store / owner	What it owns
Sessions	`sessions.db` (`sessions`, `messages`)	the transcript (never deleted; `/undo` flags, `/fork` copies)
Interactions	`sessions.db` (`approvals`, `clarifications` + event trails)	the pending/resolved lifecycle of human checkpoints

Layer	Store / owner	What it owns
Execution	`sessions.db` (`tool_requests`)	the **invocation ledger** — the system of record for what ran
Runs	`sessions.db` (`agent_runs`)	one row per turn/run, its outcome and budgets
Memory candidates	`memories.db` (`memory_candidates`)	the immutable evidence ledger (M4)
Beliefs	`memories.db` (`claims`, `claim_history`, `promotion_journal`)	the derived belief store (M10–M12)
Identity	`persona.db`	the evolving persona layer (M14); the immutable core is a file
Channels	`channels.db`	inbound/outbound state (M16)
Recall index	`claims-vector.db`	a **derived** vector index (rebuildable; never the record)

Figure 2 — State ownership: one writer per artefact

Every durable artefact has exactly one owner. Derived state (the vector index) is disposable and rebuildable; the record is not.

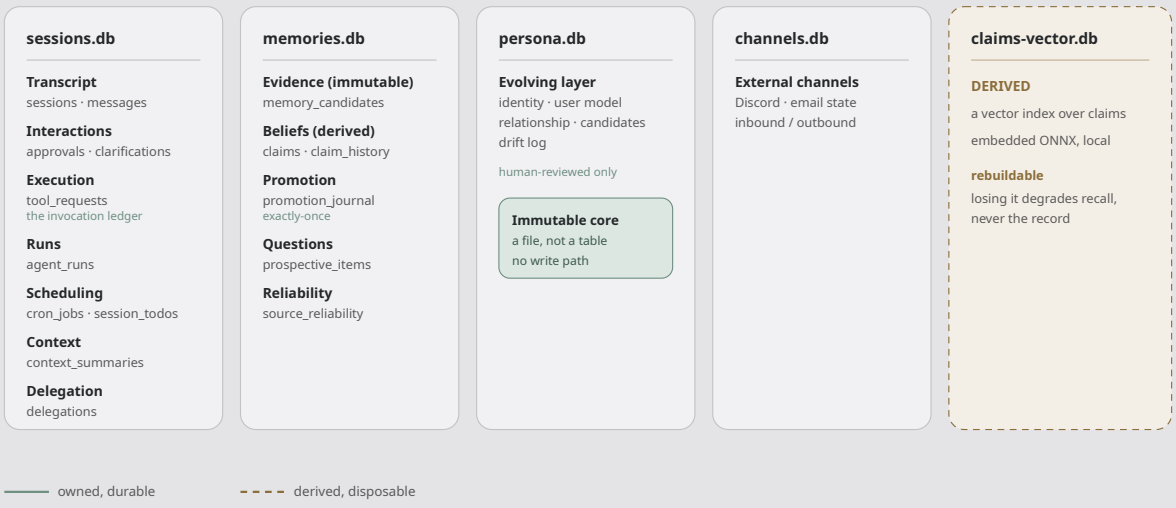


Figure 2 — the state-ownership map. Derived state is disposable; the record is not.

Two disciplines follow. **Derived state is disposable**: the vector index can be rebuilt from the ledger; losing it degrades recall, never the record. **Append-only where it matters**: candidates, invocation records, and promotion journals are never rewritten in place; changes are new rows and status transitions.

Data flow (one turn). A message enters through the interaction layer; the conversation layer resolves the session, assembles context (system prompt, skills, memory bands, history), and calls the model through the boundary. If the model proposes a tool call, the runtime validates it, applies policy (execute or park for approval), executes exactly once, records the invocation, and loops — bounded. If it proposes nothing, the turn ends and an enrichment pass (extraction, recall recording) runs off the interactive path.

---

5. Mechanisms

The subsections below take the mechanisms in the order a turn meets them: the transcript, the protocol that addresses it, the capabilities the model can use, the loop that drives them, the integrations at the edge, the boundary that encloses the whole, and the memory ledgers the turn feeds. Each states the mechanism's contract and the failure it is designed to survive.

5.1 Sessions, streaming, and search (M1–M3)

A turn is persisted to SQLite with FTS5 lexical search. Streaming is SSE (`meta → token* → done | error`); history is written only on clean completion. The transcript is the ground truth: rows are never deleted, and the non-streaming and streaming paths share one loop so their behaviour cannot drift (a divergence between them was a real, fixed bug — §7).

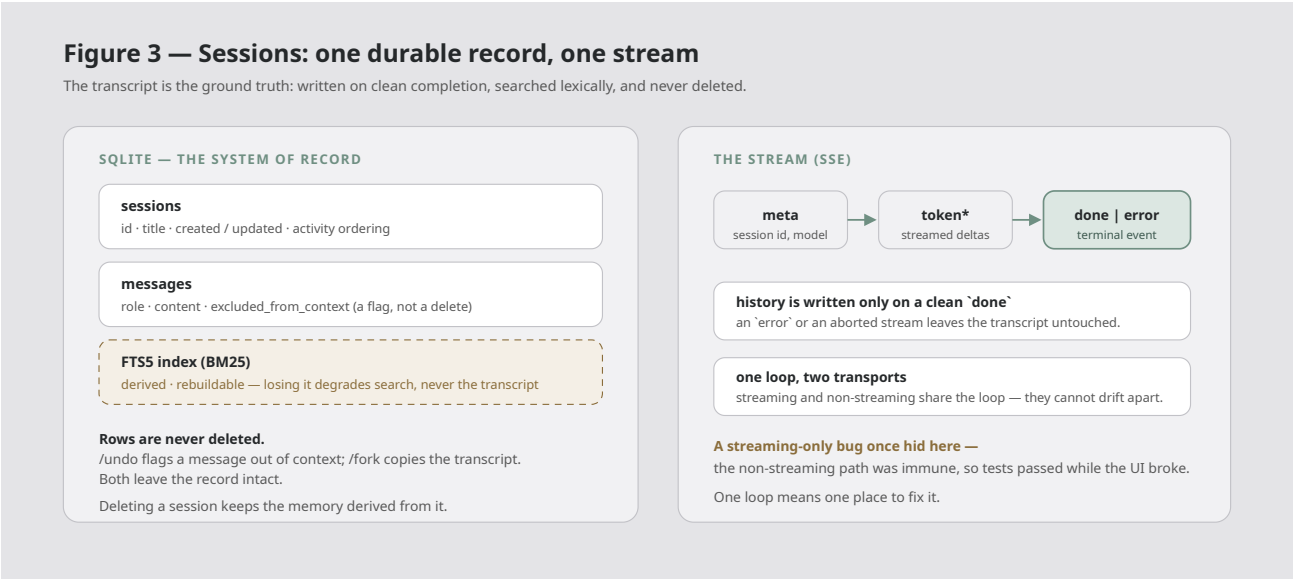


Figure 3 — the transcript is the ground truth; the stream is written only on clean completion.

5.2 The interaction protocol (M6)

Three paths stay distinct and never collapse into one message pipeline:

- **Commands** — a leading-`/` ingress handled deterministically; a command **never reaches the model**.

- **Conversation** — the model path.
- **Interactions** — approvals and clarifications, resolved **explicitly by id**; model text can never approve anything, and interactions never write message rows.

Approvals carry explicit states (`pending → approved / rejected / expired / cancelled`) with an event trail; clarifications park a run and resume it by id.

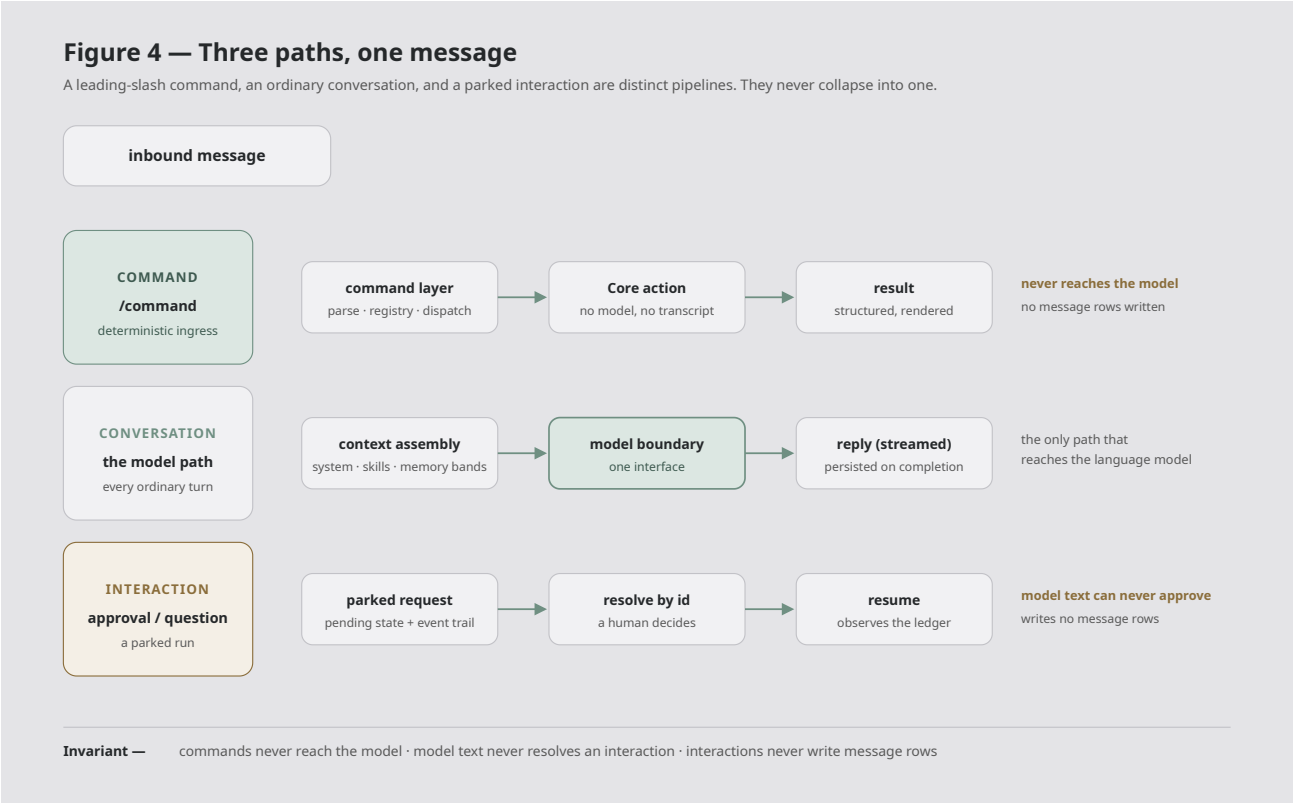


Figure 4 — the interaction protocol: three distinct pipelines that never collapse into one.

5.3 Skills (M7)

Skills are versioned markdown instruction packs (`SKILL.md`) discovered from a filesystem catalog, validated fail-closed, and injected into context when relevant (deterministic keyword discovery — no embeddings). Discovery never reads bodies and never mutates state; a malformed skill is skipped with a reason, never a crash. Later, per-turn tool *discovery* (§5.6) applies the same bounded-injection idea to tools.

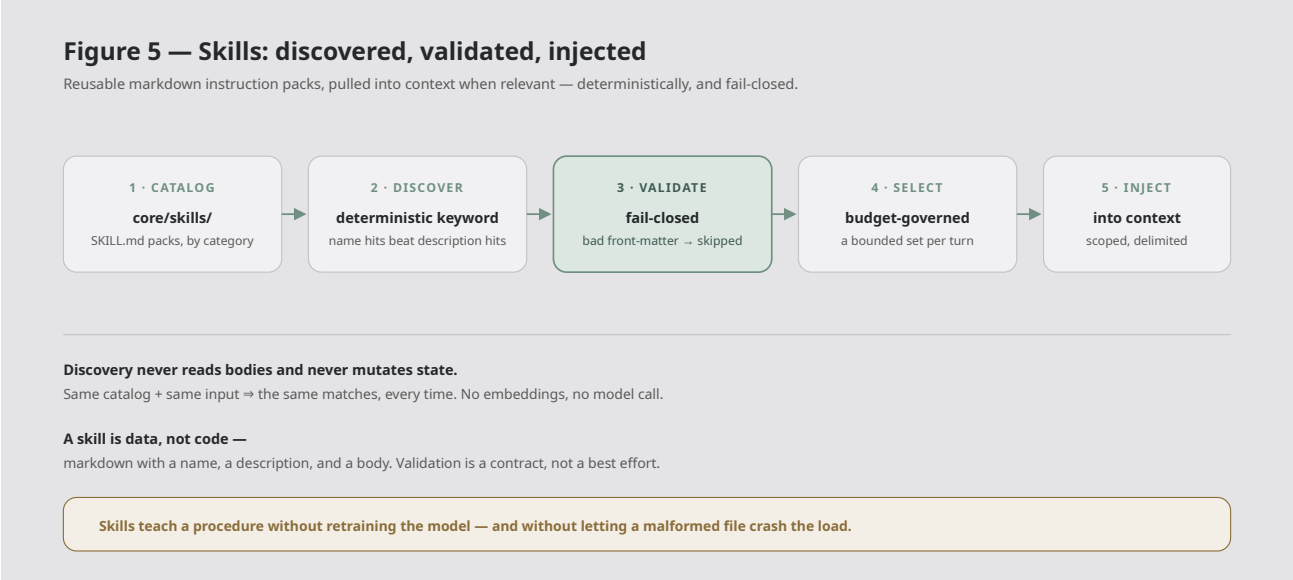


Figure 5 — a skill is data: discovered by keyword, validated fail-closed, selected under a budget, injected into context.

5.4 The tool surface (M8, M17, M19)

Tools are how the model acts. The **native** catalog is a **frozen array** — no plugin directory, no runtime registration into it — so every native tool flows through one audited path: **the model proposes; the runtime decides.** MCP tools enter through the bridge described in §5.7 under the same validation; they are namespaced and approval-gated, not appended to the native array. A call is validated (schema, bounds, no unknown fields), then either executes (approval none) or parks for approval. The hard limits are deliberate and enforced:

Limit	Value
Tool calls per step	exactly 1 (fan-out fails closed)
Tool steps per turn	5, then a forced text answer
Serialized result	64 KB
Wire argument bytes	64 KB per call / 256 KB total

Failures are structured results (`{ ok: false, failure: { code } }`), never thrown domain errors, and identical re-issues are refused as repeats (a few read-shaped tools are deliberately exempt). The surface spans files, web, skills, todo, memory, clarify, vision, image generation, a terminal, background processes, Discord moderation, a cron scheduler, `execute_code`, a headless browser, and delegation — plus everything MCP servers contribute.

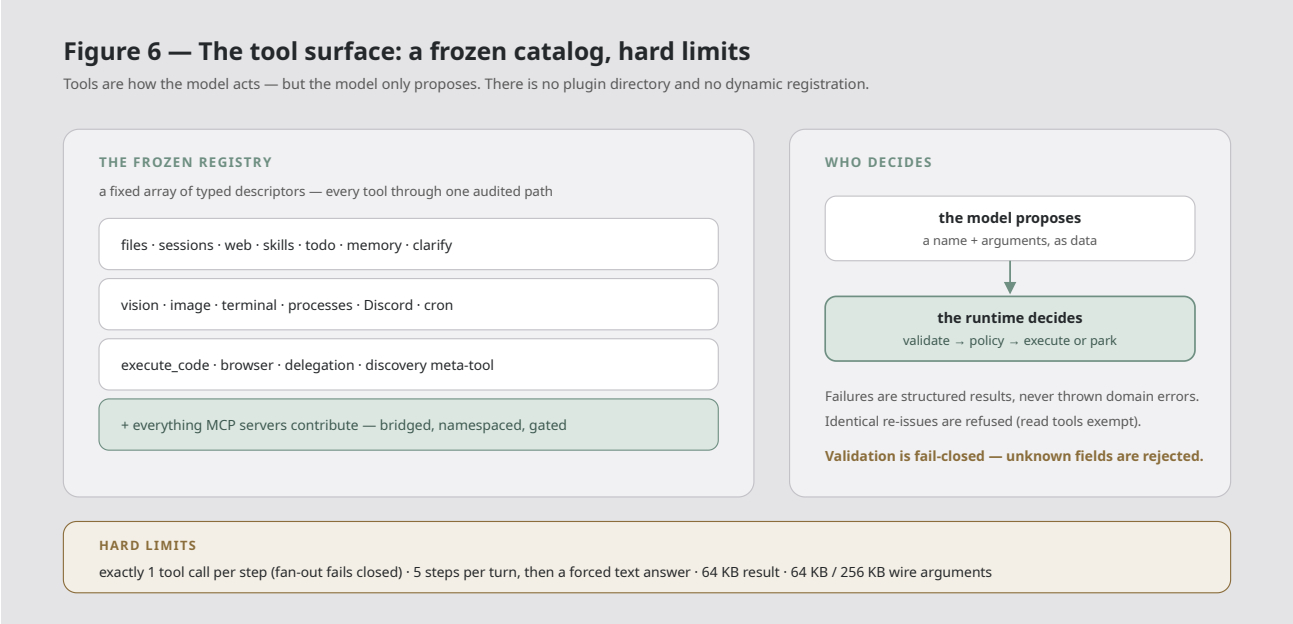


Figure 6 — a fixed registry, one audited path, and the limits that keep a turn bounded.

5.5 The invocation ledger — exactly-once (M8)

`tool_requests` is the **system of record for execution**. A call is claimed atomically (`UPDATE … WHERE state = … AND execution_token IS NULL`), executed once, and finished with its result. Resume, approval, and duplicate-resume all converge on the same ledger row; **resume observes the row, it never re-executes**. This is what makes the runtime auditable: the question "did it run, and what did it return?" has one answer, in one place.

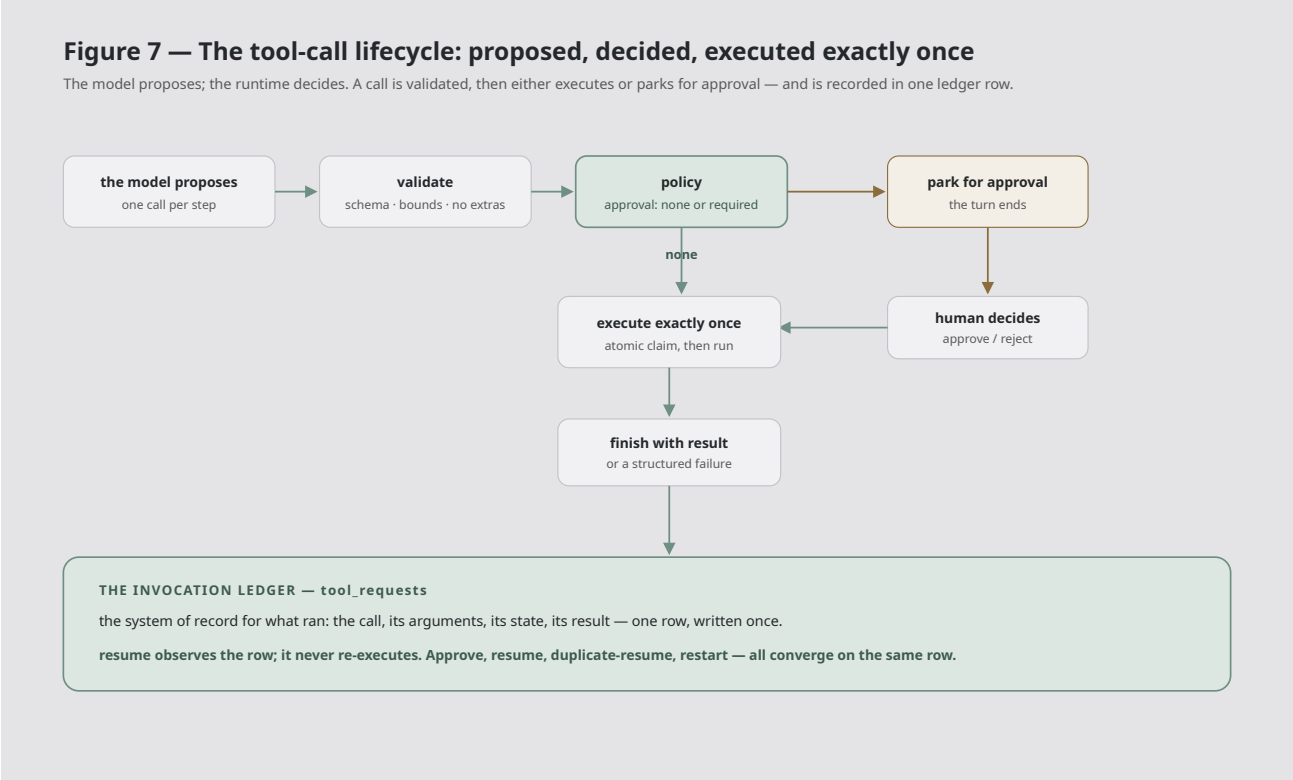


Figure 7 — proposed → decided → executed exactly once, recorded in the invocation ledger.

5.6 The agent loop and per-turn tool discovery (M9, M20.7)

A turn runs a bounded **plan → act → observe** loop with execution and time budgets, loop/repetition protection, and failure recovery. To keep each request small, ICOS does **not** send every tool schema on every turn: it injects a small always-on core plus tools discovered deterministically from the message, with a `search_platform_tools` meta-tool and `/tools pull` to reach the rest. Discovery only narrows what the model is *shown*, never what the operator *allows*; a pulled tool still approves normally. The invariant is `injected ⊆ universe`.

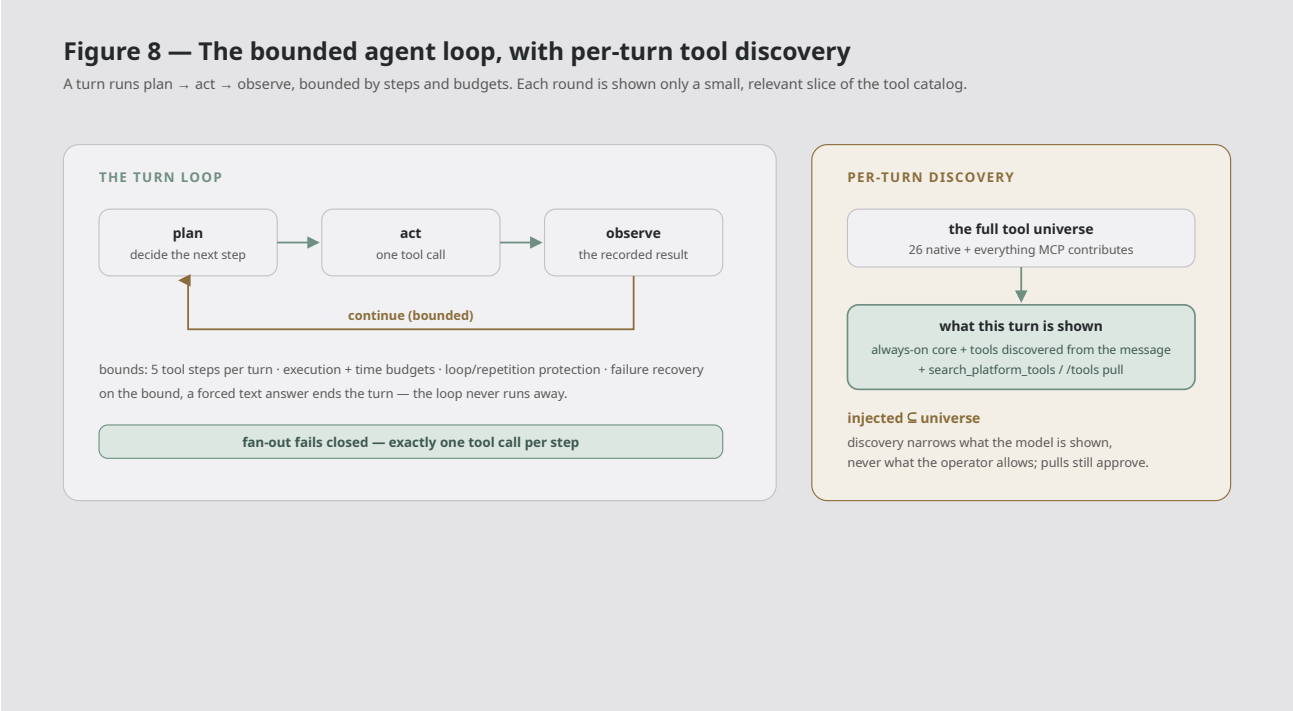


Figure 8 — a bounded plan → act → observe loop; each round sees only a relevant slice of the tool catalog.

5.7 MCP client and dynamic toolsets (M13, M19)

ICOS is a **client** to standard Model Context Protocol servers: their tools are bridged into the loop (namespaced, approval-gated, durably recorded), and their resources and prompt templates are exposed read-only. Servers can be added and the catalog reloaded **without a restart**, and each reports its health. Spawned servers receive a **minimal environment allowlist** — only the variables they explicitly reference — so a server cannot read the operator's other secrets. M19 adds dynamic **toolsets**: foreign descriptors carry their toolset, aliases resolve policy, and the operator surface exposes them.

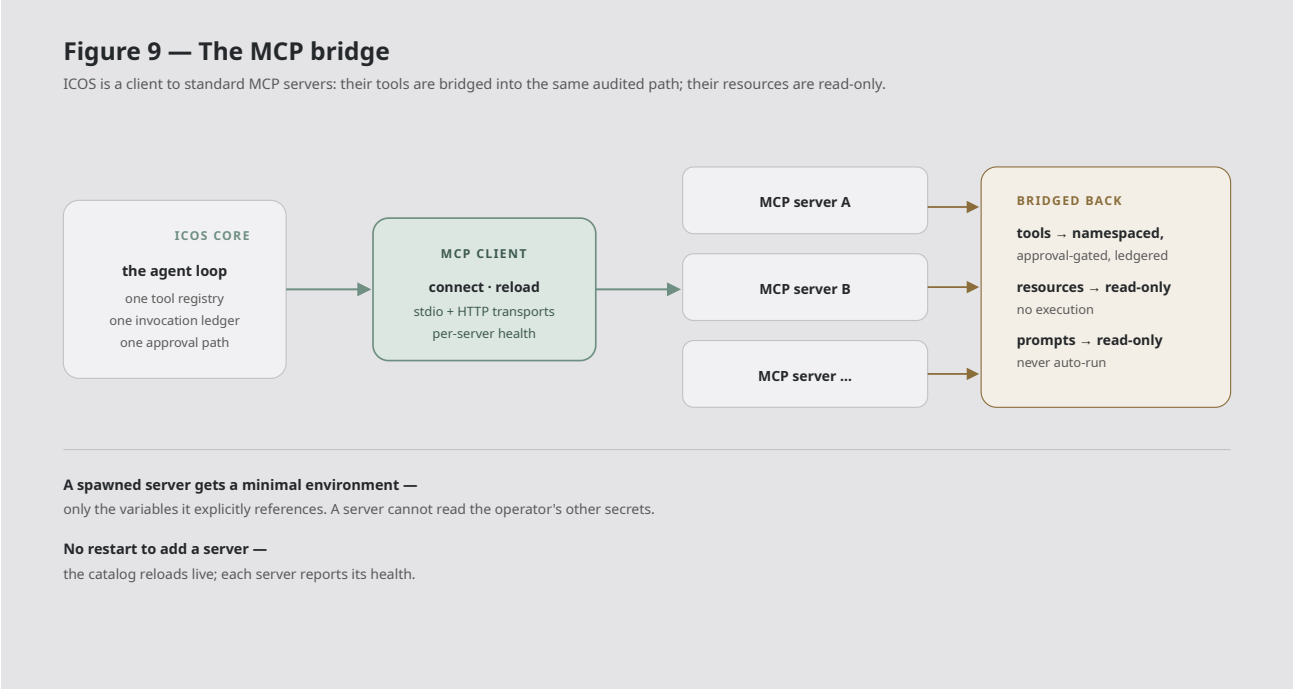


Figure 9 — ICOS as an MCP client: bridged tools into the same audited path; resources and prompts read-only.

5.8 External channels (M16)

Discord and email are first-class: inbound messages become turns, outbound sends are approval-gated (`channel.send`), and Discord registers native slash commands. Channels add no bypass — they enter the same loop and the same approval path.

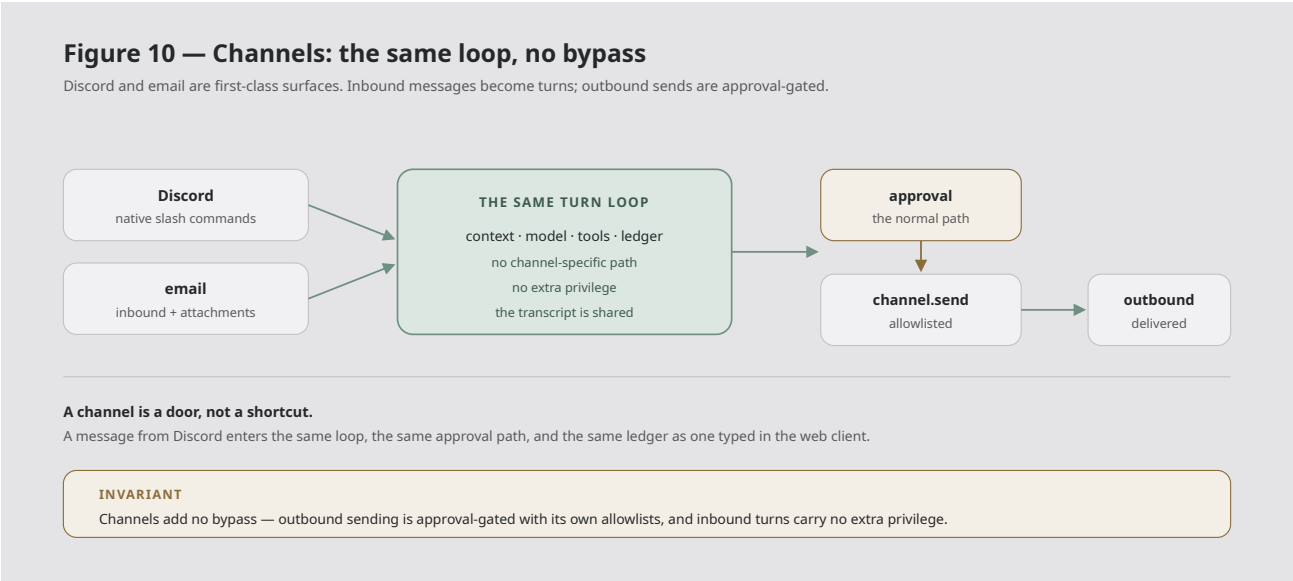


Figure 10 — a channel is a door, not a shortcut: inbound becomes a turn, outbound is approval-gated.

5.9 Security posture (S1–S6)

Secure by default for a single operator:

- **Loopback by default** (`HOST=127.0.0.1`); exposing beyond loopback requires a deliberate choice and produces a loud boot warning.
- **Authentication on by default** — a bootstrap token, HMAC-signed session cookies (httpOnly), CSRF double-submit on mutations, and per-IP rate limiting.
- **Encrypted secret vault** (AES-256-GCM) — write-only; no API, log, ledger row, or error body ever carries a value. Catalogs hold references.
- **Exact CORS origins** — `*` is refused.
- **Provider registry** — providers as data with key references resolved through the vault.

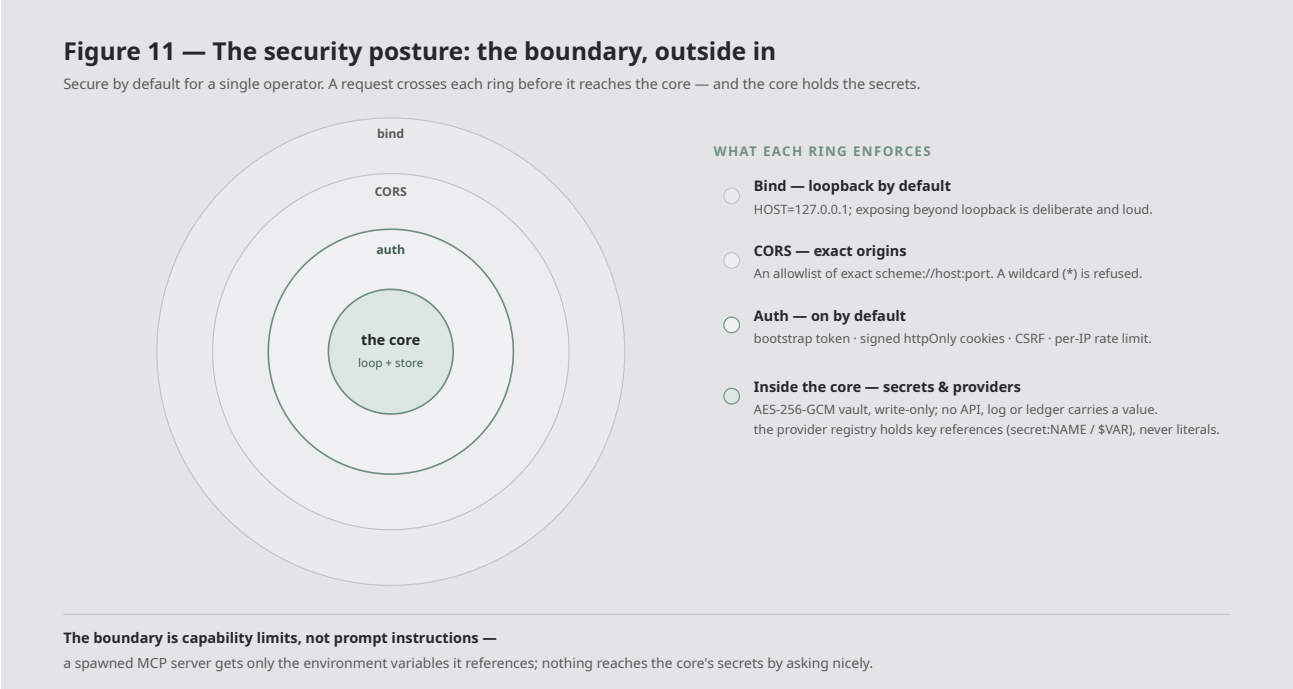


Figure 11 — the boundary, outside in: bind → CORS → auth → the core and its secrets.

5.10 Memory ledgers — candidates and promotions (M4; M10–M12 in Report II)

Memory is split into an immutable evidence ledger and a derived belief store, and both are ledgers rather than mutable rows. `memory_candidates` is append-only: each extracted triple is a row carrying its session and message provenance, extractor model and version, source role, negation marker, and confidence/importance/stability scores. A row is never rewritten; a later belief cites it rather than editing it. This is what makes memory auditable: the system can show the observation a belief came from, even after the belief changes.

Promotion from candidate to belief is the exactly-once path (M10–M12, described in full in Report II). `promotion_journal` holds one row per candidate (`candidate_id UNIQUE`), with the approval id as idempotency key and crash-recoverable states (`proposed → promoting → committed / denied / failed`). A replay observes the journal row and never re-promotes; denial and failure are terminal.

The failure it survives is a crash between approval and commit: recovery resumes or abandons the row, never doubles it.

The vector index is derived from committed claims and is rebuildable, so losing it degrades recall, never the record. Report II owns belief semantics — contradiction, revision, decay, and recall; this report owns the ledger contract they run on.

---

6. The invariance claim (modularity demonstrated)

The strongest evidence a runtime architecture can offer is not that it *looks* modular, but that later capabilities were added **without redesigning it**. That claim is cheap to assert and expensive to check, so this section states it precisely, shows the layers it covers, and says what would count as a counter-example.

The **core runtime** was complete before any of the layers below were attempted: the turn loop, the single provider boundary, the transcript, the invocation ledger, the approval path, the skills catalog, the frozen tool registry, and the security posture. Two of the layers below (M16 channels and M19 dynamic toolsets) are also described in §5 because this report evaluates them; their place in the table is about *when* they were added, not *whether* they are in scope.

What followed the core runtime was nine capability additions, each with its own question, plan, implementation, and committed evidence:

Added	What it required of the runtime
M14 identity (immutable core + reviewed evolving layer)	a new store (`persona.db`) + a grounding band; **no loop change**
M15 drift detection	a detector over existing state; observe-only
M15.5 claim verification	a verifier role behind the **same** provider boundary
M16 external channels	new adapters into the **same** loop and approval path
M17/M18 tools + skills	entries in the **same** frozen registry and catalog
M19 dynamic MCP toolsets	descriptors carrying a toolset; policy unchanged
M20 web client	a client of the **same** REST/SSE/WS surface
M20.6/M20.7 context + discovery	context assembly and injection; loop unchanged
M21 subagents	the child turn reuses the loop; approval path shared

Read as a sequence, the table is an argument about *where change was allowed to land*. Identity added a store and a band without touching the loop. Drift detection added an observer that reads state the runtime already keeps. Verification added a role behind the same client, so the provider boundary did not learn a second shape. Channels added adapters that enter the existing loop and the existing approval path rather than their own. The wider tool surface and the skill library grew arrays that already existed; dynamic MCP toolsets added a field to a descriptor and left policy alone. The web client consumed the same surface the test client had been using since M1. Context assembly and tool discovery changed what the loop *sees* — which tools are injected — without changing what it *does*. Subagents reused the loop for a child turn and shared the approval path with the parent.

None of this is proof by assertion. Each milestone plan names the contracts it expects to reuse, and each evidence file records what the implementation actually touched. Where a layer did force a change, the change is in the record rather than in a footnote: the streaming and non-streaming paths were reconciled after a real divergence (§7), and the transcript later gained an attachments column for display metadata without changing what reaches the model. The distinction that matters — "added a store" versus "changed the loop" — is visible in the same repository, and a reader who doubts the claim can check it there.

The figure makes the shape concrete: the foundation's contracts sit underneath, and every later layer is an addition at the edge.

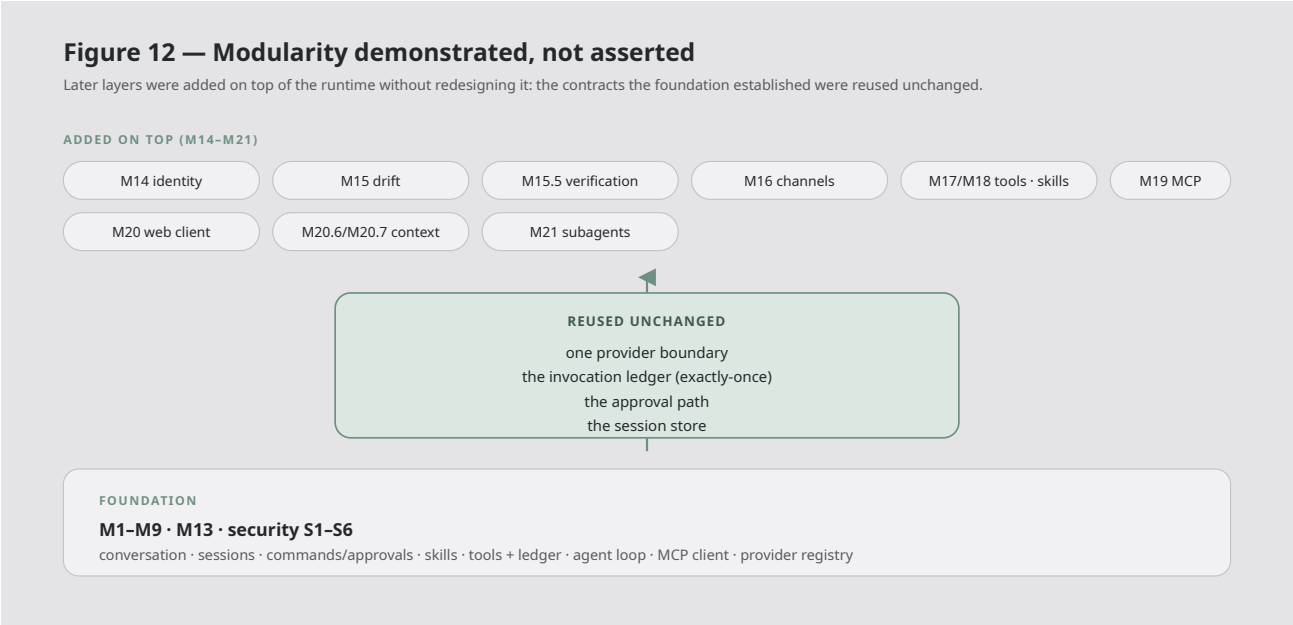


Figure 12 — later layers on top; the foundation's contracts reused unchanged.

A counter-example would be a layer that forced the loop, the ledger, the approval path, or the provider boundary to be reshaped. None is recorded in the milestones above; if one appears later, it belongs in the same evidence trail, which is the point of keeping the trail.

Two qualifications belong with the claim. It does **not** say the design was inevitable or optimal: a different foundation might have accommodated the same layers with less work, and this report offers no comparison. And it does not say that no code changed — only that the *contracts* did not: the provider boundary, the invocation ledger, the approval path, and the session store kept their shapes, and the layers above them were written against those shapes. That is a narrower claim than "modular", and it is the one the evidence supports.

---

7. Evaluation

Method. Every milestone was verified with unit tests, end-to-end tests (the full application driven through real HTTP), and live runs against a real model and database, with results committed as evidence under `.reference/plans/evidence/`. Suite size grew cumulatively with the architecture; recorded at v0.8 (22 November 2026, `a95708b`) it is **1,327 unit / 90 e2e / 245 web** (plus one deliberately skipped live-verifier spec). One release-hardening slice, R1/B1/B2, is unit and e2e only: its live run was cancelled and is recorded as the honest remainder in its evidence file, so the blanket live-run statement is scoped to the milestones.

Invariants checked. Provider independence is checked at the boundary rather than trusted: no provider branch exists in the conversation loop outside `LlmClient`, and a live swap of the endpoint changes behaviour without a code change. Exactly-once execution is checked across the whole recovery surface — approve, resume, duplicate resume, and restart — because those are the paths where a double execution would hide. Transcript immutability is checked as an operation rather than a promise: rows are never deleted, `/undo` flags rather than removes, `/fork` copies, and deleting a session clears the transcript while keeping the memory derived from it. The fail-closed surfaces are checked by injecting malformed completions and a missing persona reference frame. Approval authority is checked adversarially: model text cannot approve, and an interaction writes no message row. Discovery is checked as a bound rather than a permission: `injected ⊆ universe`, and a pulled tool still approves.

Failure-mode catalogue (reproducible traces). The series treats failures as first-class. Examples, each found in a real run or live operator use and pinned with a regression test: a **streaming-only** ledger conflict (the non-streaming path was immune, so tests passed while the UI broke); a **one-call-per-response** guard that refuses fan-out; **DSML/tool-syntax leakage** from a model into replies (counted as its own taxonomy bucket, not silently repaired); and a **vault-isolation** gap in two e2e suites exposed by real operator use.

Every terminal turn failure lands in exactly one bucket. The taxonomy is deliberately per-fix, not per-symptom, and the counts survive a restart:

Bucket	Recorded when
`provider_http`	the provider returns a non-2xx response

Bucket	Recorded when
`provider_timeout`	the request or stream times out
`malformed_completion`	the completion is structurally invalid
`upstream_error`	the provider reports an upstream failure
`ledger`	a ledger or claim conflict
`tool_failure`	a tool returns `{ ok: false }`
`tool_syntax_leak`	tool-call markup leaks into a reply
`unknown`	unclassified

The append-only `turn_errors` table retains 30 days; `GET /core/health` exposes 24-hour counts by bucket, the leakage count, and the last failure. This is what "records what happened" means in §8 — not a claim that nothing failed.

Persistence and model swap. Sessions, memory, and identity survive a full container restart and a browser refresh; a live swap of the underlying model changes behaviour without changing code.

Ablation. A provenance-on/off or approval-on/off ablation (measuring blocked actions or unresolved claims) is the planned quantitative step and is **not yet run** — stated here rather than implied.

What the evaluation does *not* establish. It shows the mechanisms behave as specified and survive restart; it does **not** show that ICOS outperforms an alternative. There are no baselines or comparisons, the model boundary in automated tests is mocked, and there is no multi-operator or scale study. §8 is explicit.

---

8. Limitations

The evaluation is deliberately narrow, and the narrowness is the honest part.

The most consequential limitation is that **the model boundary is mocked in automated tests**. The unit and e2e suites substitute a deterministic client, so they verify the machinery — the loop, the ledger, the approval path, context assembly — and not the model's behaviour or the quality of its output. Live runs exercise the real boundary, but they are observations rather than a controlled experiment, and the boundary is a dependency the runtime can make legible rather than one it can control.

There are **no baselines**. This report claims architectural properties, not measured superiority over an alternative design; such a comparison would need an equivalent system built the other way, and no such artefact is offered here. The project is also **single-implementer and single-machine**: one author's judgement shaped both the design and the evaluation, and one deployment shape — a container pair

on a laptop — is all that has been exercised. Scale is unverified in the same sense; the associative and vector surfaces have no large-n study, which is deferred to a later report.

Two limitations concern what the system can *promise*. **Hallucination mitigation is post-hoc**: claims are audited after a turn, and the runtime cannot yet block a bad output before it is shown (Report III). **Security is single-user**: authentication keeps non-operators out, but multi-tenancy is not modelled and the threat model assumes one trusted operator on one host.

Finally, **provider flakiness is boundable, not fixable**. A malformed completion fails closed; a 502-class failure is retried once when no output has been produced, and beyond that the runtime surfaces the provider's own message rather than guessing at intent. What remains is not a policy the runtime can close — it is a dependency it can only make legible, which is why the failure taxonomy in §7 records what happened instead of asserting that nothing did.

---

9. Future work

Three threads follow directly from the evaluation and its limits.

The first is the **quantitative step**. §7 records an ablation — provenance on versus off, approval on versus off, measuring blocked actions or unresolved claims — that has not been run; it is the natural next experiment, together with a scale study on the memory and retrieval surfaces. Until one exists, the architectural claims stand on mechanism and evidence rather than on measured effect.

The second is the **rest of the series**. Reports II and III cover the memory and identity layers this report references. Report IV covers agency (M22), and Report V covers subagent delegation (M21) in depth; this report uses M21 only as an invariance data point, not as a full evaluation of delegation. This report is deliberately the foundation volume, and the later volumes are where the epistemic and behavioural claims are evaluated.

The third is a small set of **deliberate contract changes**. Blocking (pre-send) mitigation would move hallucination handling from a post-turn audit into the turn itself, which changes the turn contract on purpose rather than by accident. Richer outbound channels — attachments, structured cards — belong in the same category. Each is an addition this architecture can absorb, which is a quieter version of §6's argument: the interesting future work is the work the contracts already anticipate.

---

10. Reproducibility

- **Artifact:** `a95708b2b74b75d7e404e4e30aaa9d9bad86c995` (tag `v0.8`, 22 November 2026)
  - **Repository:** https://github.com/ExileResearch/icos
  - **Release:** v0.8
- **Snapshot:** 1,327 unit / 90 e2e / 245 web (one deliberately skipped live-verifier spec), recorded at the artifact commit.

- **Setup:** Docker Compose — `bin/stack-local` launches a healthy `core` and web client using local DMR-based models for conversation and memory, with data in one mounted folder.
- **Commands:** from `core/`: `npm test` (unit), `npm run test:e2e` (e2e), `npm run lint`, `npx tsc --noEmit`. From `web-client/`: `npm test`, `npm run build`, `npm run lint`.
- **Evidence:** per-slice files under `.reference/plans/evidence/` (milestones 1–9, 13, 14, 15, 15.5, 16, 17–21, `security-hardening/`, and `release-1.0/`), browsable at https://github.com/ExileResearch/icos/tree/v0.8/.reference/plans/evidence.
- **Licensing:** code is PolyForm Noncommercial 1.0.0 (`LICENSE.md`); the report text license is declared separately on the Zenodo record and at arXiv submission.

Appendix A — Milestone and evidence map

`M#` and `S#` are the project's internal milestone and hardening-slice identifiers. Every row below points to the directory or file that owns the verification record; each record carries its own date and verification notes.

ID	Capability	Evidence (repository path)
M1	sessions, persistence, first live run	`milestone-1-evidence-gemma4.md`
M2	streaming path	`milestone-2-evidence-streaming.md`
M3	session search and lifecycle	`milestone-3-evidence-sessions.md`
M4	memory candidate extraction	`milestone-4-evidence-extraction.md`
M5	provider boundary	`milestone-5-evidence-providers.md`
M6	interaction protocol	`milestone-6/`
M7	skills	`milestone-7/`
M8	tool surface and invocation ledger	`milestone-8-evidence-live.md`
M9	bounded agent loop	`milestone-9/`
M13	MCP client	`milestone-13/`
M14	identity	`milestone-14/`
M15	drift detection	`milestone-15/`
M15.5	claim verification	`milestone-15.5/`
M16	external channels	`milestone-16/`
M17	broader tool surface	`milestone-17/`

ID	Capability	Evidence (repository path)
M18	skill library	`milestone-18/`
M19	dynamic MCP toolsets	`milestone-19/`
M20	web client, context, discovery	`milestone-20/`
M21	subagent delegation	`milestone-21/`
S1–S6	security hardening	`security-hardening/`
R1/B1/B2	error taxonomy, retry, leakage counts	`release-1.0/r1-b1-b2-observability-evidence.md`
FG5	user-story traceability matrix	`release-1.0/fg5-user-stories-evidence.md`

All paths are relative to `.reference/plans/evidence/` in the repository at the cited tag. `M10–M12` (belief memory) are Report II's scope and are intentionally absent from this table; `M22` (agency) belongs to a later report.

Acknowledgements

AI assistance. The manuscript was compiled and edited with AI assistance. The author reviewed every claim against the committed evidence and takes responsibility for the content.

References

- Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, 111(4), 1036–1060.
- Fowler, M. (2005). *Event Sourcing.* https://martinfowler.com/eaaDev/EventSourcing.html
- Laird, J. E. (2012). *The Soar Cognitive Architecture.* MIT Press.
- Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *NeurIPS.*
- Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S. G., Stoica, I., & Gonzalez, J. E. (2023). *MemGPT: Towards LLMs as Operating Systems.* arXiv:2310.08560.
- Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., & Bernstein, M. S. (2023). *Generative Agents: Interactive Simulacra of Human Behavior.* arXiv:2304.03442.
- Sumers, T. R., Yao, S., Narasimhan, K., & Griffiths, T. L. (2024). *Cognitive Architectures for Language Agents (CoALA).* arXiv:2309.02427.
- Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). *ReAct: Synergizing Reasoning and Acting in Language Models.* arXiv:2210.03629.

Systems and harnesses

- Anthropic. *Claude Code.* https://claude.com/claude-code
- OpenAI. *Codex.* https://openai.com/codex
- Nous Research. *Hermes Agent.* https://hermes-agent.nousresearch.com
- OpenClaw. https://docs.openclaw.ai ; NVIDIA. *NemoClaw / OpenShell.* https://www.nvidia.com/en-us/ai/nemoclaw
- Zechner, M. *Pi.* https://pi.dev
- LangChain. *LangChain.* https://github.com/langchain-ai/langchain ; *LangGraph.* https://github.com/langchain-ai/langgraph ; *Deep Agents.* https://github.com/langchain-ai/deepagents
- Microsoft. *AutoGen.* https://github.com/microsoft/autogen ; *Semantic Kernel.* https://github.com/microsoft/semantic-kernel
- Letta. *Letta (formerly MemGPT).* https://github.com/letta-ai/letta