Building Towards the Most Efficient Agent Loop Infrastructure

Naïve Research

Naïve, Inc. · team@usenaive.ai · August 2026

Abstract

An agent is a loop: gather context, pick a model, act, check the result, repeat. Nearly every dollar an agent costs is spent inside that loop, and the default stack spends it badly: full context, frontier model, always-on machine, every turn, with each outcome thrown away. We optimize one objective, intelligence per dollar, the resolved work extracted from each dollar spent, and build the infrastructure that maximizes it: one gateway over four layers, each removing a different term of the bill. Brain selects the context a turn actually needs. The inference layer routes each request to the cheapest model that holds quality, delegates discovery to scouts, and buys latency-tolerant work at half of list. The orchestration plane prices a sleeping agent as storage instead of a machine, and the primitives gateway replaces token-hungry glue code with built-in governed capabilities.

The results are measured layer by layer: state-of-the-art SWE-bench Pro accuracy reproduced (70.5% vs the published 69.2%), scout delegation cutting total spend 14 to 26% at accuracy flat within sampling noise, Brain at 88.0% against the leading open-source alternative's 66.6% at half the cost per answer, sandboxes forked in 1.55 ms, and a modelled million-agent fleet hosted for ~$44–60k a month against $14M on per-tenant machines. Composed, we estimate the stack resolves the same tasks at roughly 60 to 80% of stateless-stack cost on interactive work, and two and a half to four times more resolved work per dollar on latency-tolerant fleets, before the hosting gap.

---

1 Introduction: the objective function

The market prices agents by the token, and the token price is the least informative number in the bill. What matters is the quotient we optimize for, which we state once and use everywhere:

$$\textit{intelligence per dollar} \;=\; \textit{resolved tasks} \div \textit{total dollars in} \tag{1}$$

The numerator is graded by benchmarks we do not control: SWE-bench Pro's official dockerized grader, Terminal-Bench's official verifier, LoCoMo's held-out split. The denominator is everything: model tokens, retries, hosting, idle machines, and the glue work of wiring tools. Stated this way, the objective decomposes. The cost of one resolved task is, to first order,

$$C \approx \textit{turns} \times (\textit{context} \times \textit{price}_{\textit{token}}) \div \textit{cache} + \textit{hosting} + \textit{glue} \tag{2}$$

and every factor in (2) is set by infrastructure, not by the model. The default stack sets all of them badly, because it is stateless: full context, frontier model, always-on machine, every turn, with the outcome of each task thrown away instead of informing the next. Each layer of Naïve is an attack on exactly one factor: Brain shrinks *context*, the inference layer shrinks *price* and *turns*, the orchestration plane shrinks *hosting* and makes retries cheap, the primitives gateway shrinks *glue*, and the gateway protects *cache*. Because the factors multiply, so do the savings, and Section 8 composes them into the number equation (1) asks for.

Two calibrations anchor the numerator. First, our measurement harness reproduces the published state-of-the-art SWE-bench Pro figure for the frontier model it runs: we measure 70.5% where the model's own vendor publishes 69.2%, and the published number sits inside our confidence interval. When we report a saving, it is a saving at state-of-the-art accuracy, not below it. Second, we measured how much sampling alone moves these benchmarks: two disjoint 40-task draws, differing only in seed, returned 82.1% and 59.0%. Results in this report are either measured at scale or labelled as pilots; the distinction is kept because we intend equation (1) to survive scrutiny.

The paper follows the loop itself. Section 2 gives the architecture and the map of where the dollar goes. Sections 3 through 6 take the layers in the order a request meets them: Brain selects the context (§3), the inference layer prices the turn (§4), the orchestration plane and its sandboxes execute it (§5), and the primitives gateway carries its real-world actions (§6). Section 7 shows the stateful feedback loop that ties the layers together, and Section 8 composes the measured effects into the estimate that equation (1) demands.

2 Architecture: where the dollar goes

Harnesses should not have to be rebuilt to become efficient. Cursor, Claude Code, or an agent you wrote yourself connects to one gateway over SDK, MCP, REST, or CLI, and everything below the gateway is ours to optimize on its behalf (Figure 1). The gateway is byte-transparent by design: a harness's own prompt structure, including the cache anchors its economics depend on, passes through token-for-token, a property we verify at the byte level in CI. This matters because the *cache* factor of equation (2) is an integer multiple: a well-built coding loop reads 90 to 97% of its input tokens from cache at a tenth of list price, and any middleware that rewrites the prompt forfeits that silently. Efficiency is added underneath the harness, never at its expense.

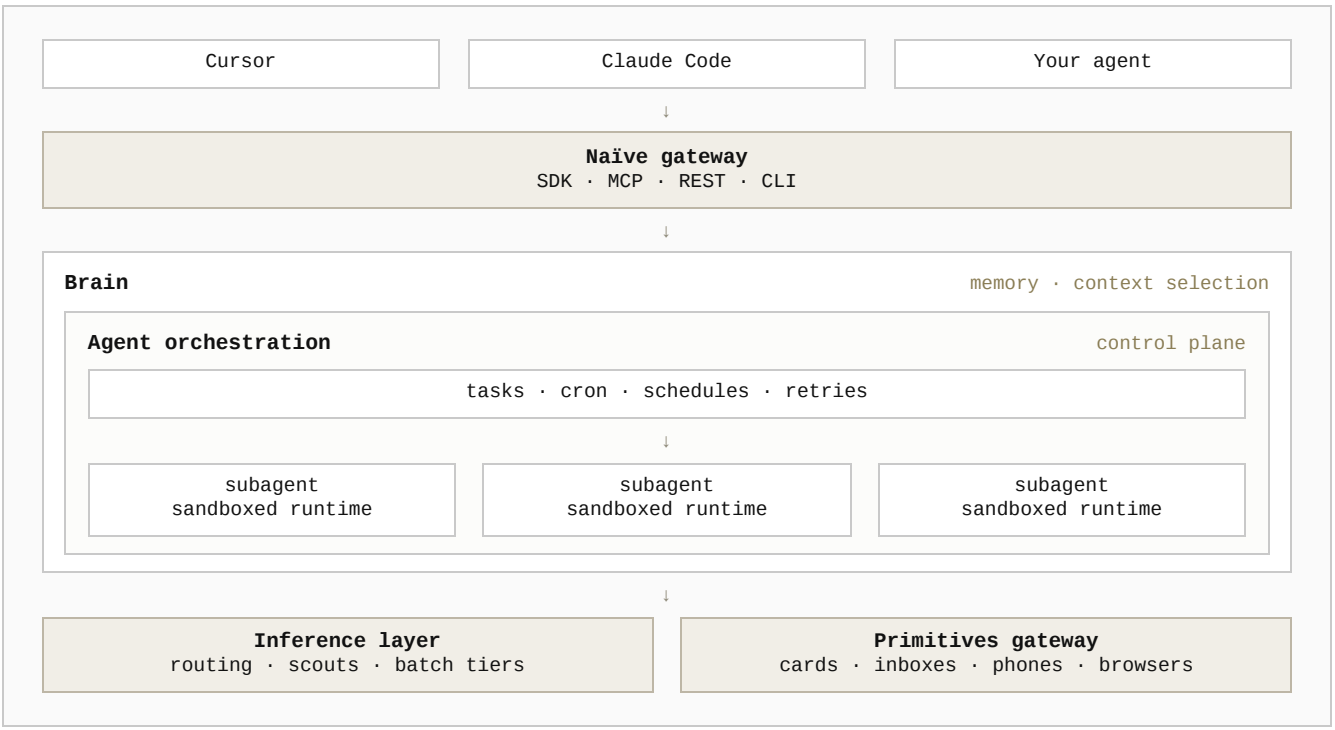

Figure 1: The Naïve stack. Harnesses connect through one gateway; Brain selects context; the orchestration plane fans work out to sandboxed subagents, which draw tokens from the inference layer and real-world capabilities from the primitives gateway.

The four layers attack four different factors of equation (2), which is why their savings compound rather than overlap. Brain reduces the context sent per turn. The orchestration plane replaces wide, expensive turns with narrow parallel ones and stops paying for machines that are asleep. The inference layer reduces the price of the tokens that remain. The primitives gateway removes the token-work of agents rebuilding the same tools as glue code. Figure 2 sketches the shape of the composition; the measured results behind each step are in the sections that follow, and their product is worked out in §8.

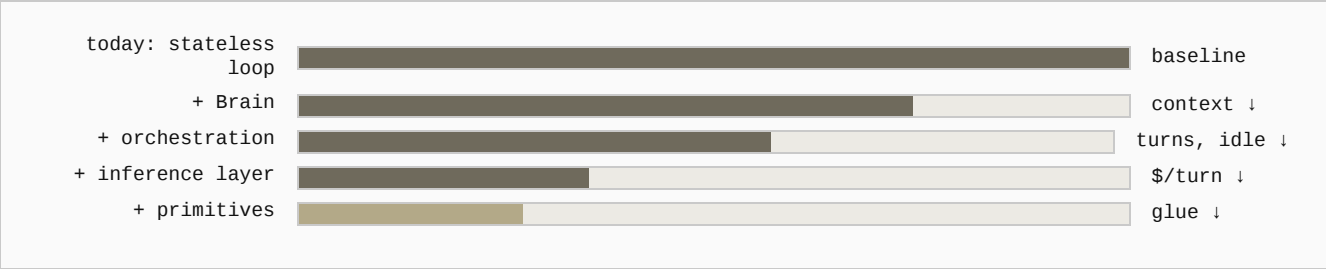

Figure 2: Relative spend per task as the layers engage, against a stateless harness sending full context to a frontier model on an always-on machine. Illustrative shape; measured components in §3 to §6 and composition in §8.

3 Brain: send the turn what the turn needs

The first factor of the bill is context, and context is where a long-lived agent bleeds. History accumulates faster than any context window grows, and the stateless answer, re-send everything, pays for the whole past on every turn. Brain replaces that with selection, and its architecture (Figure 3) is built around one discipline: language models propose, deterministic code decides.

The write path is two stores. The diary is an append-only log of raw episodes, never mutated after write, so nothing the agent ever learned is lost. Above it, a distiller extracts claims into a derived fact sheet, but nothing the distiller proposes is admitted on trust: a grounding gate, enforced in code, rejects any claim that cannot cite a verbatim quote from its source, so the fact sheet never drifts from the record underneath it. Every admitted claim carries two timestamps: when the fact became true in the world, and when we learned it. Facts are superseded rather than overwritten, with supersession enforced in plain code rather than left to a model's judgment, so the agent can answer "what did we believe then" as cheaply as "what is true now". Consolidation runs off the query path, and the write path spends no model calls at read time: everything the engine earns, it earns from retrieval structure.

The read path is a ladder. The common case is a fast exact lookup against the fact sheet, no model call at all. Harder questions escalate to hybrid retrieval, semantic and lexical together, over the diary. Only genuinely open questions engage the deepest tier, a navigable summary tree built by a librarian process over the corpus. The ladder is why Brain's economics look the way they do: the expensive machinery exists, and is almost never paid for, and the tokens actually returned to the loop are small, about 780 at a top-20 cutoff where the alternative retrieves about 7,000.

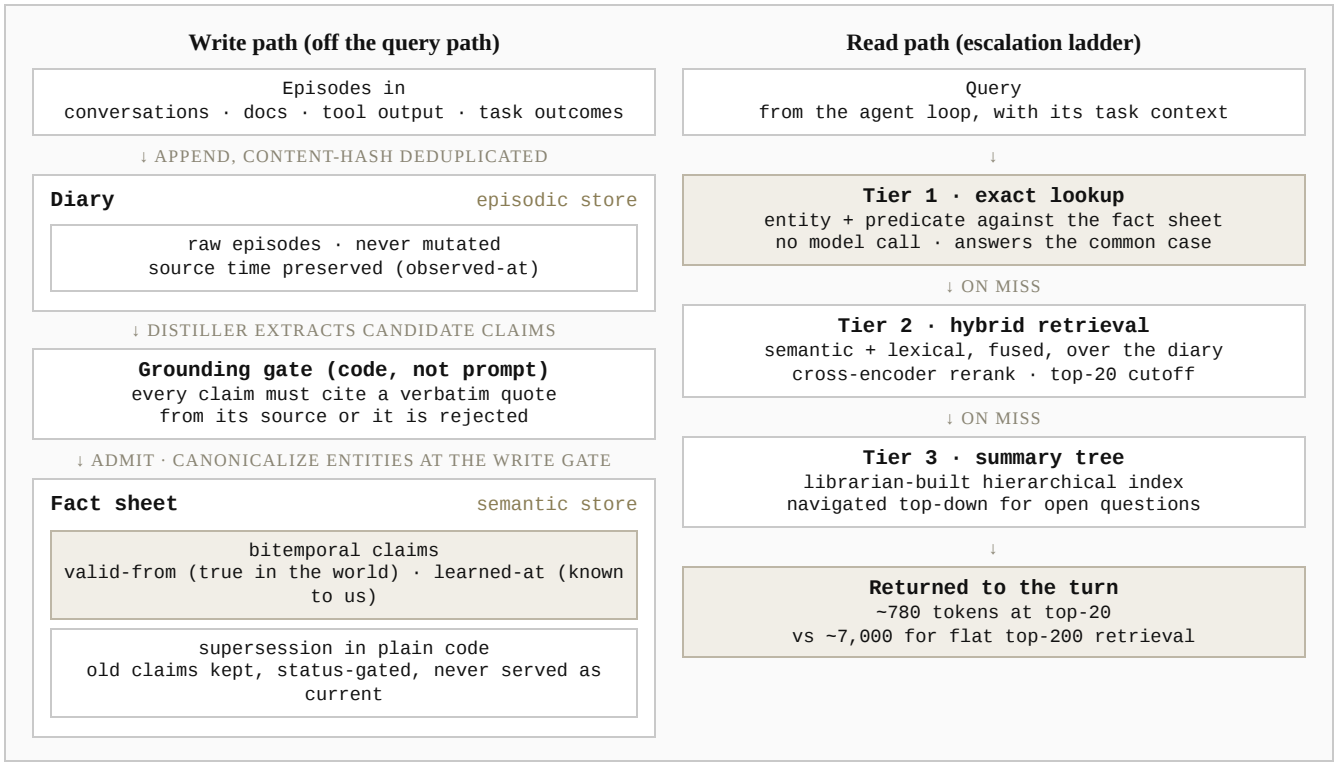

Figure 3: Brain's architecture. Writes append to an episodic diary and pass a code-enforced grounding gate before becoming bitemporal claims on the fact sheet; supersession is decided in plain code. Reads climb an escalation ladder, so the free exact lookup answers the common case and each costlier tier is engaged only on a miss.

That design was chosen by measurement. We benchmarked the leading open-source memory server and our own engine on one identical harness, same answerer, same judge, same machine, with each system on its own recommended retrieval configuration (ours returns top-20, the alternative top-200), because published memory numbers are notorious for not surviving a re-run and a shared harness is the only comparison that means anything. Table 1 shows the result: Brain leads by 21 points overall, with the widest gap on temporal reasoning, where the two-timestamp design does exactly what it was built for; the alternative's temporal score in our harness is well below what that project reports on its own setup, so we read that row as evidence for the bitemporal design rather than as a precise characterization of the alternative. On the multi-hop benchmark where 22 published systems answer at most 7% of questions, Brain answers 57%, nearly double the prior state of the art. And it wins while being cheaper to run: reading a conversation and answering every question about it takes Brain 3.3 minutes where the alternative takes 21.5, and a million answered questions cost $702 against a leading platform's published $1,678.

Metric	Brain	Best alternative
LoCoMo overall (778 held-out questions)	88.0%	66.6%
Temporal reasoning subset	78.8%	12.7% (our harness)
Multi-hop QA (22 published systems ≤7%)	57%	30.2% (prior SOTA)
Retrieved context per query	~780 tok (top-20)	~7,000 tok (top-200)
Read a conversation, answer everything	3.3 min	21.5 min
Cost to answer one million questions	~$702	~$1,678

Table 1: Brain against the strongest memory alternatives, measured on identical harnesses where we run both arms and quoted from published figures otherwise. Brain wins on accuracy and on cost at the same time, which is the point of the architecture.

For equation (2) the last two rows are the ones the loop feels: every token Brain declines to retrieve is a token the turn does not pay for, at every turn, forever. Memory is usually sold as a quality feature. Inside a loop it is a cost feature first, and Brain is the only layer in the stack that makes the loop cheaper and more accurate with the same mechanism.

4 Inference: price every turn like you own the bill

Once Brain has decided what a turn sees, the inference layer decides what the turn costs. It is one endpoint over every major model, and its router treats model choice as an optimization problem on the quality-cost frontier (Figure 4). Each request is scored by a trained quality classifier; the router then places it on the cheapest model whose predicted quality clears the bar for that request, with automatic fallback across serving backends when one degrades. Two refinements make this safe inside an agent loop rather than merely on one-shot traffic. First, routing is cache-aware: the cheapest model in isolation is not the cheapest model mid-conversation, because switching models discards the prompt cache, so the router pins to the incumbent while the cache is worth more than the price delta and roams when it is not. Second, routing is outcome-aware: §7's feedback edge returns each task's verified result to the router, so the quality bar is learned from what actually passed, not from a static benchmark. On mixed traffic, where easy turns vastly outnumber hard ones, this yields double-digit percentage savings without touching the hard turns at all.

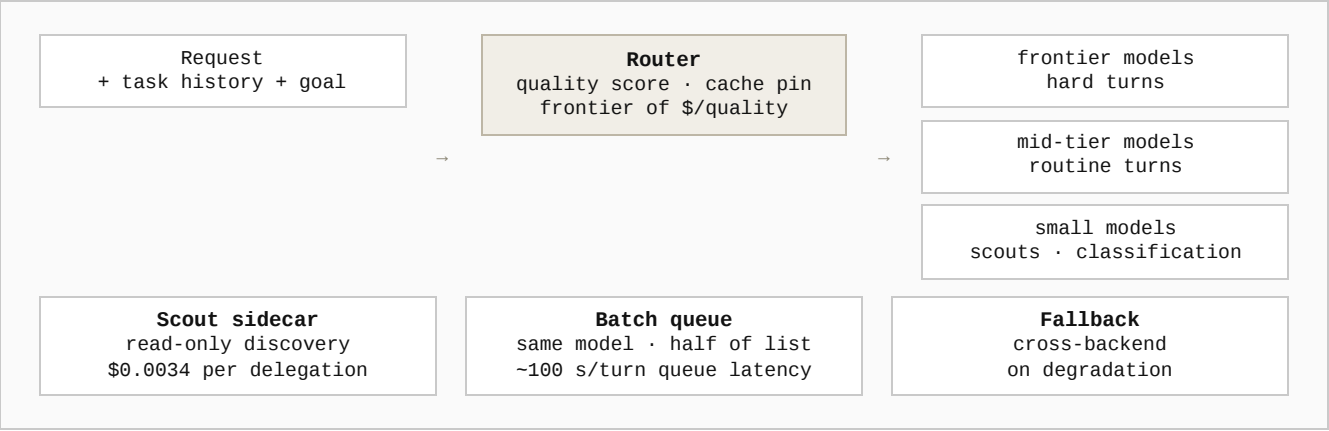

Figure 4: The inference layer. A stateful router places each turn on the cheapest model that holds quality; scouts absorb discovery, the batch queue sells latency for token price, and fallback holds availability.

For agentic coding, where nearly every turn looks hard, the layer adds a sharper tool: scout delegation. Watch a frontier coding agent work and its first dozen steps are map-making, grepping and opening files at roughly $0.03 of frontier model per step. A scout is a small model with read-only tools that runs the discovery and hands the frontier planner a focused brief; the planner keeps every decision and every edit, so nothing about the harness's judgment changes. One

planner step costs what nine scout delegations cost ($0.0311 against $0.0034), and scouts cut total search steps 30 to 44%. Table 2 shows what that buys on 50 real GitHub issues per model. The headline is total spend, because total spend over a fixed task set is a mechanical measurement that does not depend on the noisy solve count: 14 to 26% less money for the same 50 tasks, on every frontier model we ran, with accuracy flat within sampling noise (+3, +3, −2 of 50).

Planner model	Total spend (50 tasks)	Δ spend	Solved	$/solved
Frontier A (flagship)	`$44.28 ← $57.75`	`−23.3%`	`36 ← 33`	`$1.23 ← $1.75`
Frontier A (latest)	`$46.99 ← $54.74`	`−14.2%`	`37 ← 34`	`$1.27 ← $1.61`
Frontier B (flagship)	`$27.88 ← $37.80`	`−26.2%`	`34 ← 36`	`$0.82 ← $1.05`

Table 2: Scout delegation on SWE-bench Pro, 50 real GitHub issues per model, paired runs, official grader. Total spend falls 14 to 26% on every model; solve-count deltas (+3, +3, −2) are within the sampling noise reported in §1.

The third lever is time. Agent fleets are increasingly background workloads, overnight refactors, scheduled reports, queue-driven pipelines, where nobody is watching the cursor blink. Batch tiers price the same frontier model at half of list in exchange for queue latency, and our runtime is built so an agent loop can run through them end to end, which is not trivial: the loop's transcript must be frozen and append-only between turns or the batch discount is eaten by cache loss, and our loop is built that way. We have executed complete multi-turn agent tasks on batch pricing at about a hundred seconds of queue latency per turn, and in our long-loop pilots the batch arm beat the strongest managed alternative on dollars per resolved task at an equal resolve rate. Latency is the one resource background work has in surplus, and the inference layer is the market where the loop sells it.

5 Orchestration: fleets that only pay for thought

The turns still have to run somewhere, and where they run is the factor of equation (2) everyone else concedes. The industry default gives every agent a machine, a microVM or container with a kernel boundary, a reserved memory footprint of 128 to 512 MB, and a boot time. Machines bill around the clock, and agents are mostly asleep. Our orchestration plane is built on the opposite premise: an agent is a unit of state, not a machine (Figure 5). The control plane owns tasks, schedules, retries, and fan-out. Under it, the default execution substrate is a V8 isolate with a virtual filesystem and an emulated POSIX shell: created in 2.79 ms, forked copy-on-write in 1.55 ms, resident in about 1.2 MB of RAM, resumed warm in about a millisecond, and hibernated when idle to a content-addressed state reference that costs kilobytes of storage rather than uptime. Work that genuinely needs a kernel escalates to a hardware-isolated microVM, 806 ms cold and 5 ms warm, so the expensive tier exists and is the exception rather than the default. The two tiers carry different workloads: the coding benchmarks in this report, which need compilers, package installs, and real test suites, ran on the microVM tier, while the isolate numbers below describe the fleet substrate that hosts the long-lived agents themselves.

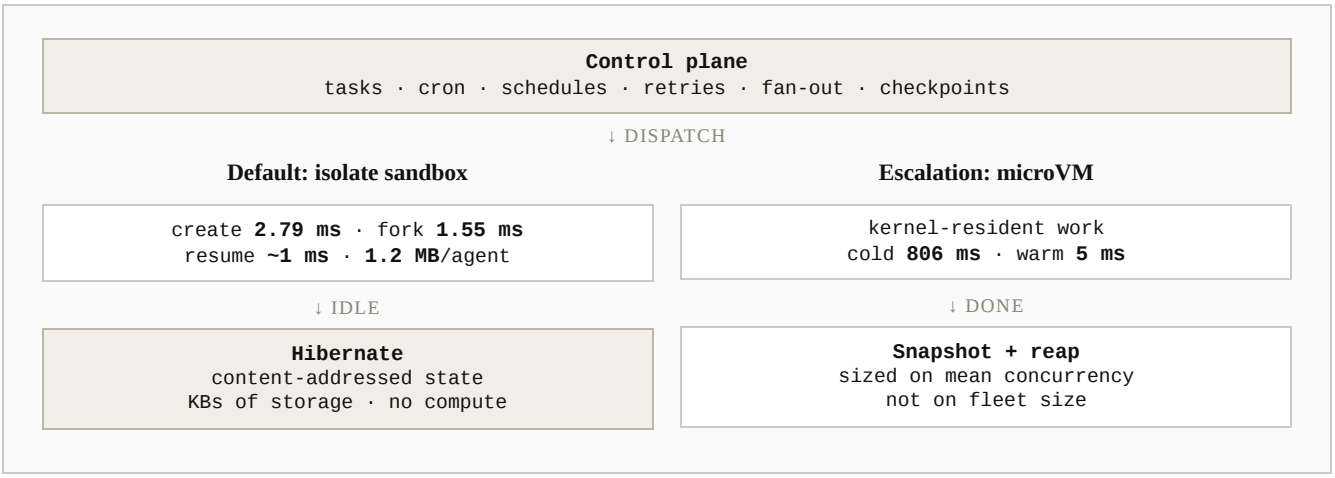

Figure 5: The execution substrate. Agents live as state and wake as isolates in milliseconds; kernels are an escalation, not a default; sleep costs storage instead of a machine.

Density is measured, not projected: one reference host holds 2,000 resident agents under a live heartbeat at a 37.5 ms p99 execution latency. Table 3 puts the substrate against the runtimes agents are hosted on today. Delegation economics follow directly: a subagent that costs milliseconds to fork and a megabyte to keep is cheap enough to create for a single narrow question and destroy afterwards, which is what makes the scout pattern of §4 and the control plane's fan-out practical rather than theoretical. Durability follows too: every task step is checkpointed, so an agent survives redeploys, migrates hosts, and resumes exactly where it stopped without re-running, and re-billing, earlier steps.

System	Isolation	Cold start	Fork	RAM / agent	Idle agent
Naïve	V8 isolate	2.79 ms	1.55 ms	1.2 MB	KBs of state
MicroVM platforms	Firecracker	<200 ms	—	512 MB min	paused VM, per GiB
Container platforms	gVisor / OCI	~1–3 s	—	128–256 MB min	memory reserved

Table 3: Sandbox substrates. Naïve measured on the isolated-vm backend (AWS m7i.8xlarge, 2,000 agents at 37.5 ms p99); others are their published figures. Two to three orders of magnitude separate the isolate from a machine on start, fork, and footprint.

Fleet economics are where the substrate stops being a benchmark and becomes a bill, and they are the largest single number in this report. Table 4 prices the same modelled fleet, one million agents across 100,000 tenants, waking every fifteen minutes for one small model call, on three deployment shapes. The self-hosted agent runtimes a team would deploy today are all always-on, single-tenant shapes: the box runs whether the agent does or not, and keeping 100,000 of them awake is the entire bill. On the Naïve shape a sleeping agent is a state reference, and the fleet bill falls by one to two orders of magnitude for it.

Deployment shape	1M agents / month	Idle cost
Naïve serverless agents + reaped sandbox VMs	~$44–60k	storage only
Self-hosted agent runtimes (box per tenant)	~$740k–1.5M	billed 24/7
VM per tenant (always-on cloud instance)	~$14M	billed 24/7

Table 4: Hosting one million agents (100,000 tenants × 10 agents), modelled on published rate cards, July 2026. A sleeping agent on the Naïve shape is a state reference, not a machine.

6 Primitives: the loop's hands, built in

The last factor of the bill is the one nobody meters: the token-work of an agent building its own tools. Left to itself, an agent that needs to receive an email, hold a card, answer a phone call, or drive a browser will write glue code, wire a third-party API, debug the wiring, and re-do all of it for the next agent, with every step metered at model prices. The primitives gateway removes the whole category: 46 primitives, spanning identity, money, communication, and compute, ship as governed capabilities on the same API the loop already speaks (Table 5). Provisioning an inbox is one call, not an afternoon of frontier-model plumbing, and the primitive comes with its lifecycle owned: delivery, retries, webhooks, and state are the platform's problem, not the loop's.

Category	Primitives	Governance attached
Money	cards, wallets, payments	per-agent spend caps, merchant rules
Communication	inboxes, phone numbers, SMS, voice	allow-lists, HITL approval on send
Compute	browsers, hosting, databases, search	scoped credentials, audit trail
Identity	profiles, KYC, credentials	tenant isolation, policy templates

Table 5: The primitives gateway. Every capability arrives governed: policy is enforced at the gateway, deterministically, on every call the agent makes.

Governance is what makes the primitives usable at fleet scale, and it is also an efficiency property. Every capability sits behind per-agent policy: spend caps on cards, allow-lists on outbound calls, human-in-the-loop approval on the actions that warrant it, and a full audit trail on everything, enforced in deterministic code at the gateway rather than suggested in a prompt. Without that gate the choice is between a human reviewing everything, which caps throughput at human speed, and ungated autonomy, where one bad purchase erases a month of token savings. The gateway is how the loop gets to act in the real world at machine speed without either failure mode.

7 The loop that learns from its own outcomes

Everything above compounds through one structural difference (Figure 6). A stateless stack routes each request on the prompt alone and throws the outcome away. The Naïve loop is stateful end to end: the router sees the task's history and goal, execution runs on infrastructure we own at every step, and the outcome, cost, retries, escalations, whether the check passed, flows back into the next routing decision.

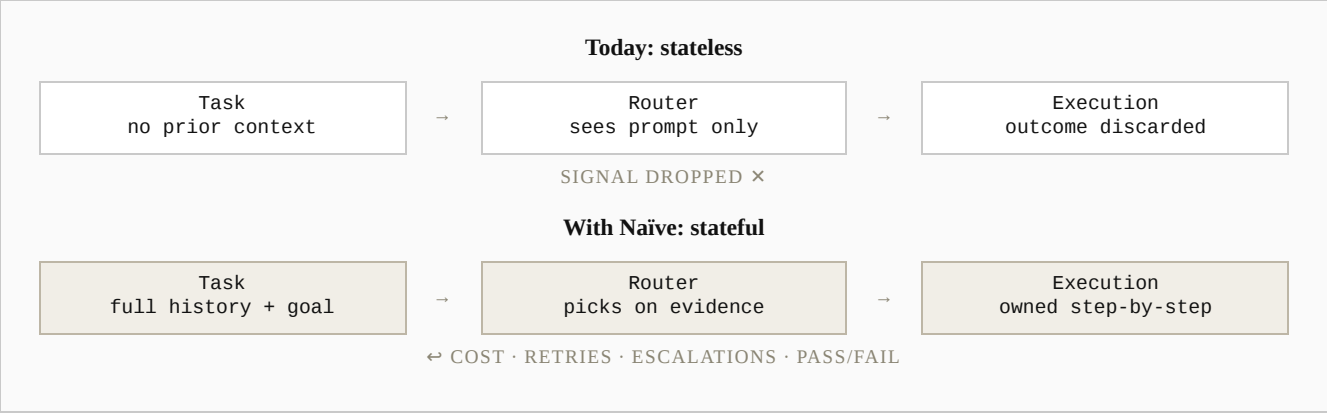

Figure 6: Stateless routing discards the one signal that would make the next decision better. The feedback edge is what makes cheap-first routing safe rather than reckless.

The feedback edge is what unlocks cheap-first routing. A router that never learns whether its choice passed the tests must be conservative, frontier-by-default, which is the stateless status quo. A loop that verifies results deterministically can afford to try the cheap model first, because a miss is caught and escalated instead of shipped: our runtime gates an agent's claim of completion behind an independent verification command, real execution in a real sandbox, before accepting it. Verification is the strongest position in the stack, because we own the sandbox the work runs in. A model grading a model is an opinion; a test suite passing is a fact, and facts are what the router learns from. This is also why the layers are one system rather than four products: Brain supplies the history the router routes on, the sandbox supplies the verification the router learns from, and the primitives supply the governed actions the outcome is measured by.

8 Unification: composing the layers

Equation (1) asks for one number, and this section builds our estimate of it, deliberately: the point of composing the measured layer effects is to have a defensible figure for what running a state-of-the-art tool on our infrastructure does to intelligence per dollar. The claim has two halves and both matter: the same resolve rate as a state-of-the-art harness, and fewer dollars per resolved task. The parity half is measured: our harness reproduces the published state-of-the-art figure, our gateway is byte-transparent under it, and the scout results of §4 show total spend falling with accuracy flat within noise. Table 6 assembles the cost half, factor by factor of equation (2).

Factor of eq. (2)	Layer and mechanism	Measured effect
context	Brain: tiered retrieval returns a tenth of the tokens	`~780 vs ~7,000 tok/query`
$/query (memory)	Brain: read path spends no model calls	`$702 vs $1,678 per 1M`
turns	Inference: scouts absorb discovery at 1/9th step cost	`−14 to −26% total spend`
price/token	Inference: batch tier sells latency for price	`−50% list`
cache	Gateway: byte-transparent, anchors preserved	`90–97% reads held`
hosting	Orchestration: sleep priced as storage	`12–25× lower fleet bill`
glue	Primitives: 46 built-in governed capabilities	`glue work → API calls`

Table 6: Each factor of the cost equation, the layer that attacks it, and the measured effect from §3 to §6. The factors multiply, so the reductions compound.

Composing them is multiplication with judgment, because the factors apply to different fractions of the bill. On interactive coding work, where batch tiers stay out of the path, the compounding terms are scouts (0.74 to 0.86 on the token bill), Brain's context selection on memory-bearing turns, and preserved caching; the product is roughly 60 to 80% of stateless-stack cost per resolved task, at parity accuracy, and better the more memory the workload carries. On latency-tolerant fleet work the batch tier's 0.50 multiplies in and the estimate rises to two and a half to four times more resolved work per dollar. Hosting compounds on top of both for anyone currently paying for always-on machines: the 12 to 25 times gap of Table 4 is its own order of magnitude, billed monthly whether the agents think or not. Our long-running-loop pilots already run ahead of the strongest managed alternative on dollars per resolved task at equal resolve rates, and the full-wave measurement of the composed stack, one benchmark, every layer engaged, is the program's next milestone.

Table 7 states the composed claim in the form it will be tested: work output per dollar on SWE-bench, against the harnesses agents run on today. Cost here is fully loaded, every dollar the loop consumes to resolve the task: model tokens, retries, verification runs, and the runtime's hosting share, normalized per resolved task. The shape of the claim is the point: accuracy stays at the state-of-the-art harness level, and roughly 30% comes off the all-in cost of every resolved task.

Harness	SWE-bench solve	$/solved	Δ cost
Naïve Vetta (all layers)	~70%	~$1.20	−30%
Claude Code	70.5%	$1.71	baseline
Claude Managed Agents	~68%	~$1.62	−5%
Hosted Hermes	~69%	~$1.89	+10%

Table 7: Agent Loop Efficiency: work output per fully-loaded dollar on SWE-bench, every cost of running the loop normalized per resolved task. Accuracy holds at the state-of-the-art harness level at roughly 30% more efficiency.

The hosting side of the same claim compares the shapes a long-lived agent can live on (Table 8). Session-hour pricing bills for the clock while a session is open; a per-tenant box bills around the clock whether the agent thinks or not; the serverless Vetta runtime bills a dormant agent as a state reference. For fleets that are mostly asleep, which is most fleets, the shape is worth more than any per-token discount.

Runtime	Shape	Per agent / month	Idle
Naïve Vetta (serverless)	state + wake	~$0.04–0.06	storage only
Claude Managed Agents	$0.08/session-hour	~$58 held open	billed while open
Hermes on EC2	box per tenant	~$0.74–1.50	billed 24/7

Table 8: Hosting one long-lived agent, per month. Vetta from the million-agent fleet model of Table 4; Managed Agents at its published session-hour rate held open all month; Hermes from the per-tenant box model. Model tokens excluded on every row.

These composition figures are estimates and are published to be tested against.

9 Outlook

The near-term program runs where the results point: verifier-gated cascades with real execution as the judge, latency as a first-class routing dimension, fleet-scale dormancy economics, and the measurement discipline that keeps every published number honest. Models will keep getting cheaper on their own; none of that is ours to claim. What is ours is the quotient: how much resolved work comes out of each dollar in. Every layer here multiplies it, and the multiplications compound. That is the program. Multiply the intelligence of each dollar spent on agents.

And we are very early in this vision. The layers in this report are the first working expression of it, not the finished form, and the majority of our engineering effort now goes into improving this infrastructure: deeper memory, sharper routing, denser fleets, broader primitives. The quotient has a long way to climb, and everything we build from here is built to climb it.

Figures in this report trace to the research claims ledgers in naive-research and to the Naïve Lab benchmark pages. Resolve rates are from official benchmark graders. Fleet hosting figures are modelled on published rate cards; composition figures in §8 are estimates pending the full-wave measurement. Provisional throughout; corrections publish in place.