Context Engineering & Eval-First Architecture

Section 01 · The Stack at a Glance — eight architectural layers and one cross-cutting substrate. The Context layer and the Eval layer are the new cost centers in 2026; they are where time, money, and quality converge. For senior staff engineers shipping production-grade agents.

FIG 1 · The 8-Layer Stack + Eval Cross-Cutting Substrate

DIAGNOSTIC QUESTION

Where is your latency budget bleeding? Most teams blame the model. Real culprits are usually retrieval (Context), tool round-trips (MCP), or chained model calls inside an unbounded orchestration graph. Instrument all three before touching the model layer.

NEW COST CENTERS	WHAT CHANGED IN 2026	WHAT HASN'T CHANGED
Context (vector + memory) + Eval (CI-gated, judge-driven). Together they routinely exceed 30% of total spend.	MCP became the default tool protocol. Eval moved from spreadsheet to CI gate. Workhorse models hit the cost/quality knee.	You still need spans, retries, idempotency, kill switch. Prompts are still software. Long agents still need durable execution.

SECTION 02

Model Layer

Pick a workhorse first. Escalate only on eval failure.

No single model wins in 2026. Your job is not to chase benchmarks; it is to put a workhorse in production, wrap it in evals, and prove — with your own failure cases — that escalation pays for itself. Most teams discover that ~80% of production calls have no measurable quality lift on a frontier model. Those are 5–10× more expensive and 2–4× slower. That is a real budget you can spend elsewhere.

TIER	MODELS	WHEN TO USE	~COST / 1K IN-OUT	p50 LATENCY
Frontier	Claude Opus 4.x · GPT-5 · Gemini 2.5 Pro	Hard reasoning, long-horizon agents, code at scale, escalation tier on eval failure.	$0.015 / $0.075	1.6–3.2 s
Workhorse	Claude Sonnet 4.x · GPT-5 mini · Gemini 2.5 Flash	80% of production calls. Default until eval shows you can't.	$0.0025 / $0.012	0.6–1.4 s
Open-source frontier	Llama 4 (405B) · DeepSeek-V3.x · Qwen 3	Self-hosted, sovereignty/compliance, cost-sensitive batch.	$0.0009 / $0.0028 (Together / Fireworks)	0.9–2.0 s
Specialist	Voyage embed · Cohere rerank · BGE · Whisper · ElevenLabs	Embedding, reranking, ASR, TTS — never use a chat model for these.	$0.00010 / $0 (embed)	40–120 ms

DECISION RULE

Start with **Sonnet 4 or GPT-5 mini** for 80% of tasks. Escalate to Opus or GPT-5 only on a measured eval-failure category, with a guarded route in your gateway and a regression test that pins the workhorse for the rest.

CONTRARIAN TAKE

Stop benchmarking on MMLU. Public benchmarks are the table stakes everyone has already optimized for. Benchmark on your own failure cases — fifty real, structured, adversarial conversations from your own domain — and you will discover that the ranking inverts for your workload more often than not.

HARD NUMBER

Workhorse-first routing typically cuts model spend by **65–80%** at <1.5% measurable quality regression on a properly scoped eval set. If you don't have an eval set, you can't claim that number.

SECTION 03

Model Gateway

Abstraction is not optional; it is survival.

If you call the OpenAI or Anthropic SDK directly from application code, you have hard-coded a vendor into your dependency graph, your secrets manager, your retry policy, and your billing model. That is fine on day one. It is excruciating on day ninety, when you need to A/B a workhorse swap, route by region, or fail over during an incident. A gateway — **OpenRouter**, **LiteLLM**, **Portkey**, or **Cloudflare AI Gateway** — gives you a single OpenAI-shaped interface, provider arbitrage, caching, and per-tenant rate limits in a few hours of work.

GATEWAY	POSTURE	BEST FOR	WATCH-OUT
OpenRouter	Hosted, broadest coverage	Day-one experimentation, rapid model A/B.	Markup over direct providers; opaque routing.
LiteLLM	Open source, self-host	Startups; full control of routing & logging.	You own ops & uptime; needs proxy infra.
Portkey	Enterprise governance	Virtual keys, prompt versioning, audit logs.	Adds a vendor; pricing scales with traffic.
Cloudflare AI Gateway	Edge cache + log	Cheap caching layer in front of any provider.	Limited routing logic vs. dedicated gateways.

Case study — the cost of skipping the gateway.

A 25-engineer startup hard-coded the OpenAI Python SDK across 14 services. Three months in, a regional outage took down 60% of their agent traffic for 47 minutes. The remediation — adding a gateway, normalizing error handling, and refactoring 22 prompts to be provider-agnostic — cost roughly **180 engineer-hours**. The day-one version of that decision would have taken four. Below is the migration shape: one environment variable, one base URL.

bash gateway-migration.sh

```bash
# BEFORE — vendor hard-coded everywhere
export OPENAI_API_KEY=sk-...
client = OpenAI(api_key=os.environ['OPENAI_API_KEY'])

# AFTER — gateway in front, swap providers without redeploying app code
export OPENAI_API_KEY=sk-or-...                       # OpenRouter key
export OPENAI_BASE_URL=https://openrouter.ai/api/v1
client = OpenAI(api_key=os.environ['OPENAI_API_KEY'],
                base_url=os.environ['OPENAI_BASE_URL'])

# Switch endpoints with a single env change — Portkey / LiteLLM / Cloudflare
export OPENAI_BASE_URL=https://api.portkey.ai/v1
export OPENAI_BASE_URL=http://litellm.internal:4000/v1
export OPENAI_BASE_URL=https://gateway.ai.cloudflare.com/v1/<acct>/<gw>/openai
```

DECISION RULE

Abstraction is not optional; it is survival. Add a gateway from day one even if you call only one provider. The cost is zero. The optionality — fallback, caching, per-tenant keys, audit, A/B — pays back inside the first incident.

SECTION 04

Framework Selection

Don't pick a framework before shipping one agent in raw SDKs.

The framework debate is a tarpit. Most teams adopt a heavyweight framework before they understand the loop they need, then spend a quarter fighting its abstractions. Build the loop yourself first: call the model, parse tool calls, dispatch, append results, loop. You will write ~120 lines of code and learn every failure mode the framework hides. Only then are you qualified to choose.

FRAMEWORK	LANG	POSTURE	BEST FOR	AVOID WHEN
Vercel AI SDK	TS	Thin, ergonomic	Fullstack apps, RSC streaming, Next.js.	Multi-agent graphs; long-running back-end work.
Anthropic / OpenAI Agents SDK	TS / Py	Provider-native, low-level	Maximum control, custom loops, eval baselines.	You want a multi-vendor abstraction.
LangChain / LangGraph	Py / TS	Heavyweight, opinionated	Multi-agent state machines, branching, persistence.	Single-call agents; teams new to async loops.
LlamaIndex	Py / TS	Retrieval-first	RAG-heavy systems, document pipelines.	Tool-use-dominant agents.
Pydantic AI	Py	Type-safe agents	Strict-schema production Python; API back-ends.	UI-stream-heavy fullstack work.
Mastra	TS	Modern, agent-first	Greenfield TS, vibe-coded apps with workflows.	Large existing Python codebases.

SPECIFICITY

LangGraph state vs. raw async loops. LangGraph gives you typed nodes, edges, checkpointed state, and the ability to *resume* a run from a node after a crash. A raw while loop with an in-memory list of messages cannot. If your agent has >3 branches, human-in-the-loop pauses, or a recovery requirement, the LangGraph tax is paid back the first time you ship a long-running workflow. Below that bar, you are paying a tax for nothing.

DECISION RULE

Don't pick a framework before shipping one agent in raw SDKs. Adopt only when its abstractions earn their cost in your specific codebase. Re-evaluate every two quarters; framework velocity is high in 2026.

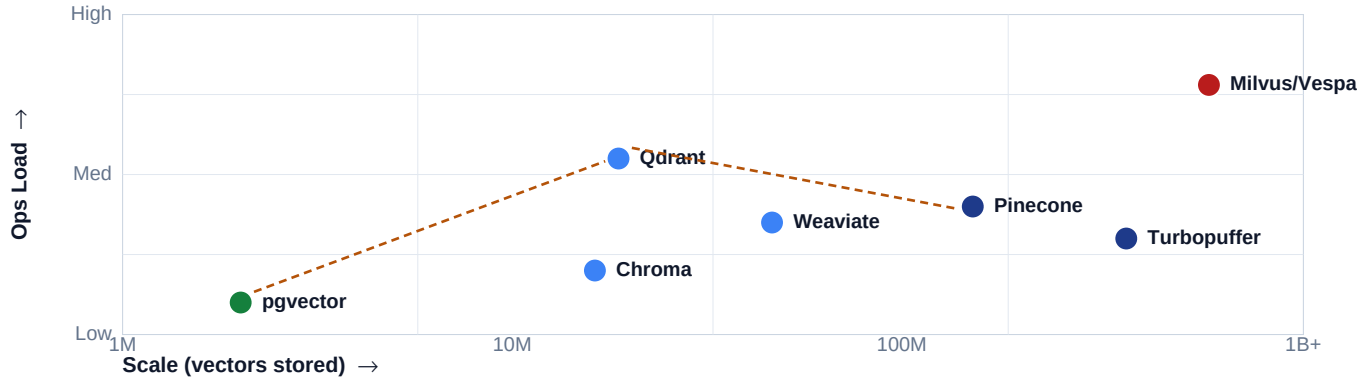
SECTION 05

Context Engineering

The biggest 2026 cost center after the model. The most common silent-failure surface.

Context engineering replaced prompt engineering as the dominant frame. Everything outside the model that shapes what it sees on each turn — chunking, embedding, retrieval, reranking, memory layering — is engineering work. The most common production regression in the wild is not a bad prompt; it is a drift in the retrieval distribution that nobody alarmed on. Three-step escalation: **pgvector** for the first 50M vectors and any team already on Postgres; **Qdrant** for hybrid search and 50M–500M scale; **Pinecone** when you value zero ops over cost. **Weaviate** (hybrid + GraphQL) and **Chroma** (prototyping) live at the edges.

FIG 2 · Vector DB Selection Matrix — Scale vs. Operational Load



Escalation path (dashed): pgvector → Qdrant → Pinecone. Don't skip steps.

FAILURE MODE

Cosine similarity on un-normalized vectors causes silent accuracy decay. If your embedding pipeline does not L2-normalize before storing, cosine similarity becomes a function of vector magnitude — i.e., it depends on the unrelated property of how long your text is. Top-k retrieval looks approximately right and is subtly wrong.

python vector-normalize.py

```python
# Always L2-normalize before insert AND before query.
import numpy as np
def l2_normalize(v): n = np.linalg.norm(v, axis=-1, keepdims=True); return v / np.clip(n, 1e-12, None)
embeddings = l2_normalize(embed_batch(texts))    # CRITICAL at index time
store.upsert(ids, vectors=embeddings.tolist())
q = l2_normalize(embed(query))                   # same normalization, same model version
hits = store.search(q.tolist(), top_k=20, metric='cosine')
assert abs(np.linalg.norm(q) - 1.0) < 1e-5       # sanity check on every deploy
```

DIAGNOSTIC QUESTION

Is your RAG retrieval deterministic enough to debug? Pin embedding model version, tokenizer, chunker, normalization, HNSW M/efSearch, and reranker. If any drifts silently between deploys, you have a non-debuggable retrieval system. Replay the same query at the same commit and assert the same top-k.

DECISION RULE

Escalate vector stores in order: pgvector → Qdrant → Pinecone. Don't skip steps. Skipping pgvector for a managed vendor is the most common premature optimization in 2026 — Postgres migration cost is small; managed-vendor rollback is not.

SECTION 06

Tool Use (MCP)

Standardize tool discovery on MCP. Stop wrapping APIs by hand.

By April 2026, the **Model Context Protocol** (MCP) is the dominant tool-definition layer across major model vendors. Function calling at the wire level still exists; MCP sits above it as the discovery, schema, and transport standard. The practical effect: a tool you write once is consumable by every framework and every model, with no per-vendor adapter. The cost of *not* standardizing is a graveyard of one-off integrations.

SOURCE	WHAT IT GIVES YOU	WHEN TO REACH
Anthropic MCP servers	First-party reference servers (Filesystem, Memory, GitHub, Slack, etc.).	Default for canonical workplace tools; lowest support risk.
Composio	Managed catalog of 250+ integrations exposed as MCP, with auth.	You need many SaaS integrations and don't want to run servers.
Arcade	Auth-aware MCP servers — OAuth, PKCE, token refresh handled.	Multi-tenant agents that need per-user OAuth across providers.
Custom MCP (TS/Py SDK)	Wrap your internal services. Hours, not weeks.	Domain-specific tools — every team needs at least three.

What MCP buys you: a single discovery handshake (tools/list), JSON-Schema-defined inputs, structured outputs, and a capability-negotiation contract that means your agent can be pointed at a new server without code changes. What it does *not* buy you: correctness. You still need eval coverage on every tool's expected and adversarial paths.

DECISION RULE

Standardize on MCP for tool discovery; it reduces integration debt by ~40%. Treat any non-MCP tool you write as legacy from the moment it is committed. Internal services get an MCP wrapper before they get an agent client.

SPECIFICITY

Tool hallucination is the #1 production failure that points back to the tool layer (see Section 10). The fix lives here: tighter JSON-Schema constraints, required arrays on every input, narrow enums, and a negative-test suite of malformed calls in your eval harness. MCP makes those constraints portable.

SECTION 07

Orchestration

Durable execution is what separates demos from production.

If your agent ever needs to wait for a human, sleep for hours, or survive a deploy, it needs durable execution. The framework's in-memory loop is not enough — process death, region failover, and rate-limit backoff all require state that lives outside the process. Three serious choices in 2026: **LangGraph**, **Inngest**, and **Temporal** (with **Restate** as a credible up-and-comer). They are not interchangeable.

TOOL	MENTAL MODEL	USE WHEN	AVOID WHEN
LangGraph	Typed state-machine graph; nodes & edges; checkpointed.	Multi-agent branching, deterministic graph state, replayable traces.	Linear, single-call agents; non-Python/TS stacks.
Inngest	Event-driven step functions; serverless-first.	Async webhooks, scheduled work, fan-out/fan-in serverless.	Hard real-time UX where latency <100ms matters.
Temporal	Workflow + activity model; deterministic replay.	Long-running, stateful workflows; multi-week jobs; strict SLAs.	Small teams without ops capacity; greenfield prototypes.
Restate	Durable promises & virtual objects; lighter than Temporal.	Modern Temporal-like guarantees with smaller runtime cost.	Existing Temporal investment; need broad ecosystem.

SPECIFICITY

Use Temporal for long-running, stateful workflows; LangGraph for graph-based state machines. Two concrete tells: if you need durable execution that survives *multi-day* sleeps, payments-grade idempotency, or polyglot worker fleets — **Temporal**. If your agent's correctness depends on conditional branching across nodes with checkpointed state, in a Python/TS stack — **LangGraph**. Pick the model that matches your reasoning, not the brand.

DECISION RULE

If your agent waits for a human or runs >60s, you need durable orchestration. Below that bar, the framework loop is enough. The fastest way to fail in production is to build a 12-hour agent on top of a 60-second runtime.

SECTION 08

Observability

The cheapest insurance you can buy. Buy it on day one.

Observability is the difference between an agent you can fix and one you have to throw away. Every production agent emits structured spans for model.call, tool.call, retrieval, and reflection, with token counts, costs, latencies, and the full prompt/response. Without that, debugging is divination — you cannot replay a regression, you cannot diff a bad deploy, and you cannot prove a fix.

TOOL	POSTURE	BEST FOR	TYPICAL OVERHEAD
LangSmith	Hosted; deep LangChain/LangGraph integration	Tracing + evals when you're already on LangChain.	~2–4% inference cost
Arize Phoenix	Open source, OTel-native, self-hostable	Standardized OTel pipelines; existing telemetry infra.	Self-host cost only
Helicone	Drop-in proxy + cost dashboard	Fastest setup; great cost reporting; gateway-friendly.	~1–2% inference cost
OTel + Honeycomb / Datadog LLM	Existing observability stack	You already have a span pipeline; want LLM in the same UI.	Reuses existing budget

HARD NUMBER

Observability cost should be <5% of total inference cost. If it is higher, your stack is inefficient — you are double-logging, capturing too many spans, or serializing entire prompts that you never query. Budget the observability line as a fraction of inference, not a fixed dollar amount.

SPECIFICITY

Minimum span set per turn: agent.turn (root) → retrieval (with query, k, latency, hit IDs) → model.call (with model, prompt hash, tokens, cost) → tool.call (one per tool, with input schema hash, latency, error). Without those four, you cannot answer 'what changed' on a regression.

CONTRARIAN TAKE

Most teams over-rotate on dashboards and under-rotate on *diff tooling*. The single highest-leverage observability artifact is a **prompt-and-response diff between two deploys**, on the same eval set, surfaced as a PR comment. Build that before you build a dashboard.

SECTION 09

Eval Layer

Eval is cross-cutting. It is also where 2026 teams under-invest.

Eval is the substrate that makes every other layer measurable. Without a CI-gated eval set, you cannot ship a prompt change, a model swap, a retriever tuning, or a tool refactor with confidence. Eval is the only thing that turns an agent into software. The bar is not high — five test cases per category beats fifty cherry-picked anecdotes. The bar is just non-zero.

TOOL	POSTURE	USE WHEN
Promptfoo	OSS, YAML-driven, CI-friendly, model-as-judge built in.	You want eval-as-code in version control with PR-blocking checks.
Braintrust	Hosted; datasets + playground + eval orchestration.	Cross-functional teams; non-engineers contribute test cases.
DeepEval	Pytest-style; assertion-based; deep RAG metrics (faithfulness, etc.).	Python codebases that already test with pytest; RAG-heavy systems.
LangSmith / OpenAI Evals	Vendor-attached, deeply integrated.	You're already on the corresponding stack and want zero glue code.

Eval coverage minimum (non-negotiable):

1. Correctness — known-good answers, judged or asserted. **2.** Regression — last quarter's bugs frozen as test cases. **3.** Safety — refusal, jailbreak, PII leak. **4.** Cost — token-budget caps per turn. **5.** Latency — p95 thresholds per route.

DECISION RULE

Five test cases per category beats fifty anecdotes. Categories are above. Anecdotes are screenshots in Slack. Anecdotes do not run in CI; categories do. Move every interesting screenshot into a category within 24 hours.

SPECIFICITY

Automate evals in CI/CD; manual evals are for exploration only. A blocking GitHub Action runs your eval suite on every PR that touches prompts/, tools/, retrievers/, or models/. Failure rolls the PR back. Manual eval is for finding new categories — it is never the gate.

SECTION 10

5 Production Failure Modes

Five failures that kill agents in the wild — and the layer that fixes each.

These are the failures we see most often when teams ship to real users. Each maps to a layer; each has a fix that is boring and operational, not glamorous. The pattern: failures cluster at the seams between layers, not inside them.

FAILURE	WHERE IT LIVES	SYMPTOM	FIX (LAYER)
Context window overflow	Context / Framework	Silent truncation; agent forgets system prompt or earliest tool result; quality cliff at long sessions.	Token-count guards before every model call; sliding-window summarizer; promote facts to long-term memory. **(Context + Framework)**
Tool hallucination	Tool Use / Eval	Model invents tool names or arguments; calls non-existent endpoints; returns made-up IDs.	Tighten JSON-Schema with required + enums; reject & retry on schema fail; negative-path eval cases. **(MCP + Eval)**
Eval overfitting	Eval	Suite is green; users still complain. Tests were authored by the same person fixing the bugs.	Hold-out set sourced from real traffic; rotate test authors; refresh 20% of cases per quarter. **(Eval)**
Retrieval drift	Context	Top-k slowly degrades after a re-embedding or chunker change; nothing throws.	Pin embedding model + chunker version; alarm on hit-rate against a frozen probe set; replay regression. **(Context + Observability)**
Runaway loop / cost	Orchestration / Gateway	Agent recurses on its own output; bill spikes; users see no progress.	Hard step-cap per run; per-tenant cost guard at the gateway; reflection node with no-progress detection. **(Orchestration + Gateway)**

CONTRARIAN TAKE

Four of the five failures above have nothing to do with the model. The model is rarely the bug. The seam between two layers — Context and Framework, MCP and Eval, Orchestration and Gateway — is almost always where the bug lives.

SECTION 11

Decision Rules Summary

One line per layer. Print this page. Pin it above your monitor.

LAYER / TOPIC	RULE
MODEL	Default to a workhorse (Sonnet 4 / GPT-5 mini). Escalate only on a measured eval-failure category.
MODEL GATEWAY	Add a gateway on day one. Abstraction is not optional; it is survival.
FRAMEWORK	Don't pick a framework before you've shipped one agent in raw provider SDKs.
CONTEXT	Escalate vector stores in order: pgvector → Qdrant → Pinecone. L2-normalize before insert and query.
TOOL USE (MCP)	Standardize on MCP. Treat any non-MCP tool as legacy from the moment it's committed.
ORCHESTRATION	Long-running or human-in-the-loop > 60s → durable execution. Temporal for stateful workflows; LangGraph for graphs.
OBSERVABILITY	Emit spans for model.call, tool.call, retrieval, reflection. Keep observability cost <5% of inference cost.
EVAL	Five test cases per category beats fifty anecdotes. Eval gates run in CI; manual eval is for exploration.
RUNTIME	Edge for <30s user-facing agents; Modal/Fly for long-running or GPU; serverless for event-driven.
FAILURE-FIRST POSTURE	Bugs live at the seams between layers, not inside them. Instrument the seams first.
BUDGET	Eval + Context together can exceed 30% of total spend. Plan for it; don't be surprised by it.
MIGRATION	Re-evaluate frameworks every two quarters. The 2026 ecosystem moves fast; sunk cost is a tax.

META-RULE

Every rule above is operational, not aesthetic. If you are about to break one, write down the trade-off before you do — in a commit message, a design doc, or a Slack thread. Trade-offs you can name are survivable. Trade-offs you can't are technical debt.

SECTION 12

Next Steps

Audit your current stack in one week. Then ship the deltas.

Most production agent stacks have three to five real gaps and a dozen imagined ones. The audit below is designed to find the real gaps in five days. Run it with one engineer; it does not need a meeting.

Week 1 Audit Template

DAY	FOCUS	OUTPUT
Mon	Inventory. List every layer in your stack, the tool you use, and the version. Identify hard-coded vendor calls.	One-page stack diagram + a list of gateway-bypass call sites.
Tue	Cost & latency. Pull last 30 days of model + tool spans. Compute cost-per-turn, p50 / p95 latency, and tail.	Cost waterfall by layer; latency budget table.
Wed	Eval coverage. Count cases per category (correctness, regression, safety, cost, latency).	Eval coverage matrix; named gaps.
Thu	Failure replay. Re-run the last five production incidents against current code. Note which ones still reproduce.	Regression test cases promoted into the eval suite.
Fri	Decisions. Pick the top three deltas. Write a one-page memo per delta — problem, fix, owner, gate.	Three memos. Three PRs scoped for the next sprint.

Actionable checklist — score yourself out of 12.

■	Gateway in front of every model call (no direct vendor SDKs in app code).
■	Workhorse model is the default route; frontier is opt-in via eval-failure tag.
■	Vector store choice matches the scale escalation rule (pgvector → Qdrant → Pinecone).
■	All embeddings L2-normalized at insert and query; embedding model version pinned.
■	Every tool exposed via MCP — including internal services.
■	Durable orchestration in place for any agent with sleeps, human-in-loop, or runs >60s.
■	Spans for model.call, tool.call, retrieval, reflection emitted on every turn.
■	Observability cost is <5% of inference cost (verified, not assumed).
■	Eval suite gates PRs that touch prompts, tools, retrievers, or models.
■	≥5 cases per eval category (correctness, regression, safety, cost, latency).
■	Per-tenant cost guard + hard step-cap on every agent loop.
■	Last 5 production incidents are reproducible against current code as eval cases.

FINAL RULE

Score <6 of 12: you have a prototype, not a production stack. Score 6–9: you have a functional stack with two or three high-leverage deltas. Score ≥10: you are ahead of the median; spend the surplus on adversarial eval and tail-latency work. The goal is not 12; the goal is to know which boxes you've chosen not to check, and why.