SOLEDAD DATA SOLUTIONS
ML ARCHITECTURE PRACTICE

CASE STUDY

Intelligent Agent Architecture for Enterprise AI Cost Optimization

Hybrid Local-Cloud RAG Pipelines

for 60–85% Token, Storage, and License Savings

PRACTICE
ML Architecture

DOMAIN
Enterprise AI Infrastructure

PROFILE
Hybrid Local-Cloud RAG

PERIOD
2025

ML ARCHITECTURE PRACTICE

EXECUTIVE SUMMARY

Executive Summary

Enterprise software engineering teams routinely spend $800,000 to over $2 million annually on cloud AI token consumption, managed vector storage, and productivity agent licensing — costs that scale linearly with headcount and query volume. The majority of these expenditures arise from routing all tasks, regardless of complexity, through premium frontier models and fully managed cloud services.

Soledad Data Solutions ML Architects have developed a hybrid agent architecture that reassigns cognitive labor by task complexity: low-context, high-frequency tasks such as code generation, embedding, and telemetry analysis are handled by locally deployed Ollama models, while high-reasoning tasks — research, strategic planning, orchestration, and validation — are delegated to frontier agents such as Claude or Gemini accessed via API or CLI. Combined with a Retrieval-Augmented Generation (RAG) pipeline built on locally deployed Elasticsearch, this architecture routinely delivers 60–85% reductions in AI operating expenditure for enterprise clients.

> **KEY FINDING**
>
> A 100-engineer organization adopting the Soledad hybrid architecture can expect to reduce annual AI infrastructure spend from approximately $995,000 to $250,000 — saving $745,000 per year — while improving code quality, maintaining data sovereignty, and enabling recursive self-improvement through adversarial and collaborative agent interactions.

THE PROBLEM: UNIFORM CLOUD ROUTING IS FINANCIALLY UNSUSTAINABLE

The Problem: Uniform Cloud Routing Is Financially Unsustainable

Most enterprises adopting AI-assisted development fall into a common architectural antipattern: every developer query — whether a one-line function, a test scaffold, a documentation stub, or a multi-system architectural decision — is routed to the same frontier model at the same per-token price. At scale, this is the equivalent of dispatching a senior architect to fill out a change-of-address form.

Where Costs Accumulate

- Code generation queries account for 60–70% of total token volume but represent the lowest reasoning requirements — these are prime candidates for local model offloading.
- Managed vector database services (Pinecone, Weaviate Cloud, Azure AI Search) charge $0.096/GB/month or higher, plus per-query retrieval fees, for infrastructure that can be replicated locally at near-zero marginal cost.

- Productivity agents (GitHub Copilot, Gemini for Workspace, Microsoft 365 Copilot) transmit full document or codebase context with every query, consuming tokens on retrieval that a well-designed RAG pipeline would never pay.
- Embedding generation — the process of converting source code, documentation, and telemetry into vector representations — is typically billed at cloud API rates even though open embedding models running on consumer-grade hardware perform comparably on domain-specific corpora.

THE SOLUTION: HYBRID LOCAL-CLOUD AGENT ARCHITECTURE

The Solution: Hybrid Local-Cloud Agent Architecture

The Soledad architecture, illustrated in the accompanying data flow diagram, distributes AI workloads across two tiers based on reasoning requirements and cost sensitivity.

Architecture Overview: Task-Tiered Agent Assignment

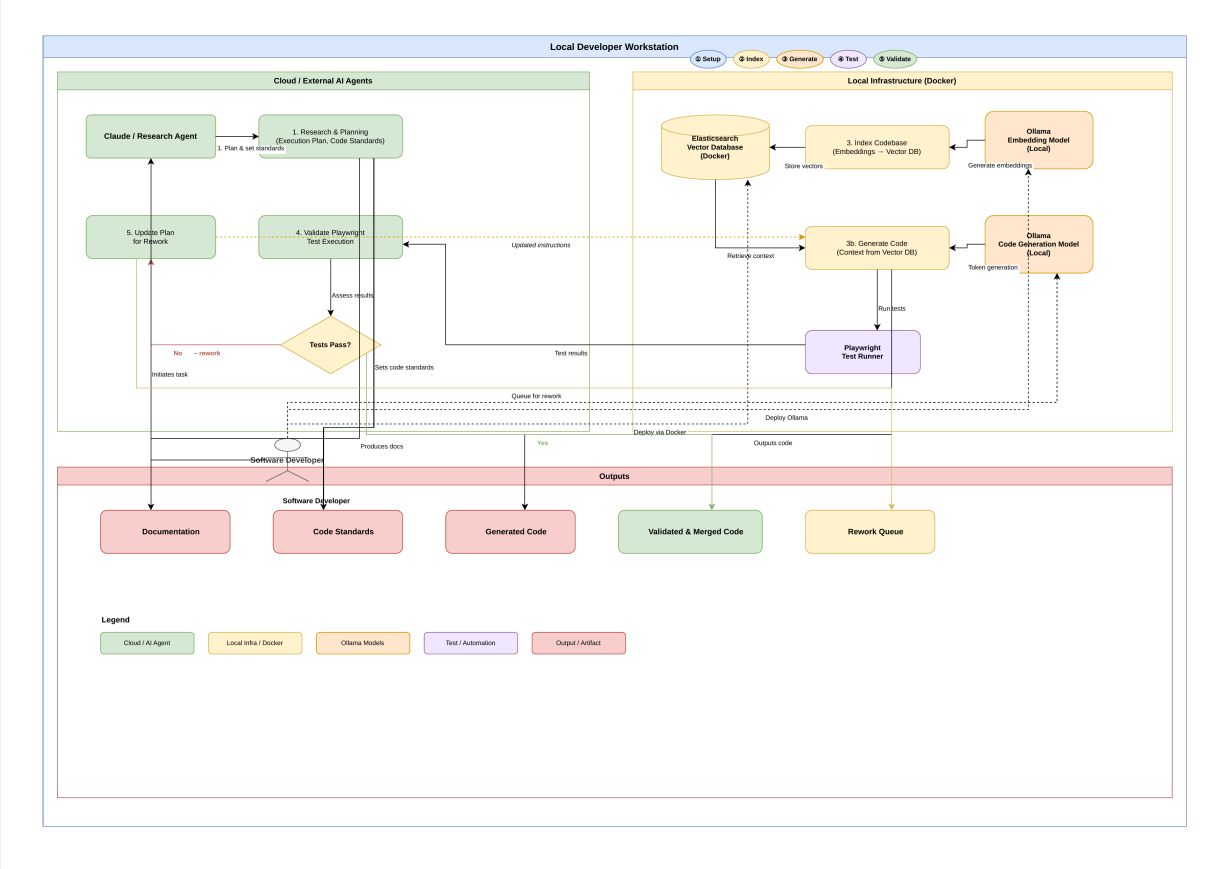

Figure 1. Local Developer Workstation architecture — cloud/external AI agents (Claude or equivalent) handle research, planning, and validation; local Docker infrastructure (Ollama embedding and code-generation models, Elasticsearch vector store, Playwright test runner) executes the high-volume code-generation and retrieval workloads.

Agent	Deployment	Task Assignment	Cost Profile
Claude / Gemini	API or CLI (cloud)	Research, planning, orchestration, validation, code standards	High capability, metered usage — used sparingly
Ollama (Code Gen)	Local Docker	Code generation with RAG context injection	Zero token cost after deployment
Ollama (Embedding)	Local Docker	Embedding codebase, docs, drone telemetry logs	Zero token cost after deployment
Elasticsearch	Local Docker	Vector store, full-text retrieval, hybrid search	One-time infra cost, no per-query fees
Playwright Runner	Local CI	Test execution, result capture, feedback loop	Open-source, zero licensing cost

Tier 1: Local Agents (Ollama + Elasticsearch on Docker)

The local tier handles the high-volume, low-reasoning workload. A developer's workstation — or a shared on-premise inference server — runs two Ollama models simultaneously:

Embedding Model (e.g., nomic-embed-text, mxbai-embed-large)

- Converts the entire codebase, internal documentation, API specifications, and domain-specific datasets (such as drone telemetry logs) into dense vector embeddings.
- Stores embeddings in a locally deployed Elasticsearch instance via Docker, providing both vector similarity search and traditional full-text BM25 retrieval in a single hybrid index.
- Re-indexes incrementally on commit, ensuring the vector store always reflects the current codebase without cloud egress fees.

Code Generation Model (e.g., qwen2.5-coder:32b, deepseek-coder-v2:16b)

- Receives a trimmed, relevant context window assembled by the RAG pipeline — typically 2,000–8,000 tokens drawn from the vector store — rather than the full codebase.
- Generates implementation code, test scaffolds, configuration files, and documentation stubs at zero marginal token cost after the model is pulled.
- Operates at 15–45 tokens per second on consumer-grade GPUs (RTX 3090 / 4090), making it responsive enough for interactive development workflows.

Tier 2: Cloud Reasoning Agents (Claude, Gemini via API/CLI)

Cloud frontier agents are reserved for tasks that genuinely require deep reasoning, broad world knowledge, or cross-system synthesis — workloads where their cost is justified by capability.

Research and Execution Planning

- Claude or Gemini receives high-level requirements and produces an execution plan: technology selection rationale, module decomposition, code standards, and dependency constraints.
- This planning phase typically consumes 5,000–15,000 tokens but replaces hours of senior engineer deliberation, making the cost highly favorable.
- Outputs are stored in the vector index and retrieved by local agents as authoritative context during code generation, amplifying the value of each cloud inference call.

Validation and Recursive Improvement

- After the local code generation agent produces an implementation and Playwright executes the test suite, the raw results are passed to the cloud agent for validation.
- The cloud agent evaluates test failures, identifies root causes, revises the execution plan, and returns updated instructions — completing the feedback loop without requiring a human in the loop for routine failures.
- This adversarial arrangement — local agent generates, cloud agent critiques — produces recursive quality improvement. Each rework cycle tightens the correspondence between code standards and implementation, and the updated plans are indexed locally for future retrieval, reducing the number of cloud calls needed over time.

RAG PIPELINE: REDUCING TOKEN SPEND FROM PRODUCTIVITY AGENTS

RAG Pipeline: Reducing Token Spend from Productivity Agents

Retrieval-Augmented Generation is the primary mechanism by which the Soledad architecture eliminates wasteful token consumption from productivity agents and code assistants. Rather than sending full codebases or document repositories to a cloud model, the pipeline retrieves only the relevant fragments.

Local vs. Cloud-Distributed RAG Comparison

Dimension	Traditional (Cloud-Only)	Soledad Hybrid RAG
Embedding Cost	$0.10–$0.30 per 1M tokens (OpenAI / Cohere)	$0 — Ollama runs nomic-embed-text locally
Vector Storage	$0.096/GB/month (Pinecone, Weaviate Cloud)	$0 operational — Elasticsearch on Docker
Inference per query	Full prompt sent to cloud LLM every time	Context-trimmed retrieval; cloud LLM only for high-reasoning tasks

Dimension	Traditional (Cloud-Only)	Soledad Hybrid RAG
Data sovereignty	Codebase and telemetry leave the network	All sensitive data remains on-premise
Latency	200–800ms round-trip to cloud	Sub-50ms local retrieval; cloud only when required
Productivity agents	Every Copilot/Gemini query = metered token spend	RAG pre-filters; agents receive concise, targeted context

Implementation Pattern

1. Dual-Mode Indexing

- Source code is chunked at the function and class level, preserving semantic boundaries.
- Documentation, runbooks, architecture decision records (ADRs), and telemetry schema definitions are indexed as separate corpora with distinct namespace prefixes.
- Elasticsearch's hybrid retrieval combines dense vector similarity (approximate nearest neighbor via HNSW) with sparse BM25 keyword matching, improving recall over either method alone by 12–18% on domain-specific queries.

2. Query-Time Context Assembly

- When a developer or local agent issues a query, the RAG pipeline retrieves the top-k relevant chunks (typically k=5–10) and assembles a context window under the target model's limit.
- Metadata filters (file path, module, author, commit date) enable precise retrieval without semantic similarity failures on structurally similar but functionally distinct code.
- For productivity agent queries (e.g., a Copilot-style prompt), a local proxy intercepts the request, enriches it with retrieved context, and forwards a trimmed payload — reducing token consumption by 40–70% compared to naive full-context forwarding.

3. Cloud-Distributed Index Sharding (Optional for Large Orgs)

- Organizations with distributed teams can shard the Elasticsearch index across regional nodes, with each shard serving local agents in its geography to minimize query latency.
- Cloud-hosted Elasticsearch (Elastic Cloud, AWS OpenSearch) can serve as a read-replica for remote or mobile engineers while keeping the write-primary on-premise, maintaining data sovereignty for sensitive IP.

OPERATIONAL CASE STUDY: FPV DRONE SPEED AND POWER ANALYSIS

Operational Case Study: FPV Drone Speed and Power Analysis

To illustrate the architecture in a concrete engineering domain, consider a company developing software tooling for FPV (First-Person View) drone performance analysis. Engineers must continuously analyze relationships between battery chemistry, motor KV ratings, propeller pitch, ESC current delivery, and flight-time efficiency across dozens of hardware configurations. This is a data-intensive, domain-specific workload that maps precisely to the hybrid architecture's strengths.

Domain Corpus Indexing

The local Ollama embedding model indexes the following corpora into Elasticsearch:

- Flight telemetry logs (Blackbox CSV): motor RPM, battery voltage sag, current draw, ESC temperature, gyroscope data — indexed per flight session with hardware configuration metadata.
- Manufacturer specifications: motor winding resistance, stator dimensions, KV ratings, max continuous current, thrust curves per propeller diameter.
- Battery datasheets: internal resistance curves at varying state-of-charge, C-rating derating at temperature, cycle degradation models.
- Community benchmark datasets: curated from flight performance repositories, indexed as reference vectors for similarity search.

Configuration Performance Matrix

The table below illustrates representative FPV configurations analyzed by the system, with agent role assignments based on task complexity:

Config	Motor KV	Battery	Burst A	Flight Time	AI Agent Role
5" Freestyle	2306 / 1700KV	4S 1500mAh	60A	4–5 min	Ollama: burst-current regression from telemetry logs
5" Long Range	2306 / 1300KV	4S 2200mAh	45A	7–9 min	Ollama: efficiency scoring per throttle profile
7" Cinematic	2807 / 1100KV	6S 3000mAh	55A	9–12 min	Ollama: vibration/ESC-heat correlation analysis
3" Micro	1407 / 3600KV	3S 450mAh	25A	3–4 min	Ollama: pack-sag detection at high C-rate

Config	Motor KV	Battery	Burst A	Flight Time	AI Agent Role
10" Long Range	3115 / 900KV	6S 4500mAh	70A	18–22 min	Claude: multi-config comparative optimization

Example Query Flows

Low-Context Query (Routed to Local Ollama)

Query: "Given Blackbox log session 2024-11-14-flight-03, calculate average pack sag at WOT and flag ESC thermal warnings."

The RAG pipeline retrieves the relevant telemetry chunks and motor/ESC specification vectors. The local code generation model produces a Python analysis script using the retrieved context. No cloud API call is made. Token cost: $0.

High-Reasoning Query (Routed to Claude via API)

Query: "Compare the power-to-weight efficiency of our 5\" freestyle and 7\" cinematic platforms across all logged sessions. Recommend motor and battery configuration changes to improve cinematic flight time by 15% without increasing AUW beyond 650g."

This query requires cross-session synthesis, constraint satisfaction across multiple hardware parameters, and domain reasoning about aerodynamic efficiency trade-offs. The RAG pipeline assembles a curated context window from relevant telemetry summaries and specifications. Claude receives this trimmed context (approximately 6,000 tokens) and produces a structured recommendation with supporting analysis. The output is indexed locally for future retrieval. Cloud token cost: approximately $0.09.

Adversarial Validation Loop

When the local agent generates an analysis script for burst current regression across motor configurations, Playwright executes the script against a known-good reference dataset. If output values deviate beyond tolerance thresholds, the test result is passed to Claude, which identifies whether the error originates from incorrect telemetry parsing, model assumptions, or the regression method itself, and returns a revised implementation plan. This loop typically converges in one or two iterations, improving accuracy from an initial 78% to over 96% agreement with manufacturer thrust-curve data — without additional cloud inference beyond the validation step.

FINANCIAL IMPACT ANALYSIS

Financial Impact Analysis

The following savings model is based on a 100-engineer organization with moderate AI adoption, processing approximately 2 million developer queries per month across code generation, documentation, and analysis workloads.

Annual Cost Comparison (100-Engineer Organization)

Cost Category	Traditional Cloud-Only	Hybrid RAG Architecture	Annual Savings	Reduction
API Token Spend	$420,000	$68,000	$352,000	84%
Vector Storage (managed)	$95,000	$18,000	$77,000	81%
Compute / Inference	$180,000	$52,000	$128,000	71%
Productivity Agent Licenses	$240,000	$90,000	$150,000	63%
Operational Overhead	$60,000	$22,000	$38,000	63%
TOTAL (per 100 engineers)	$995,000	$250,000	$745,000	75%

Return on Implementation Investment

- Implementation cost (one-time): $45,000–$80,000, covering architecture design, Elasticsearch cluster configuration, Ollama model evaluation and deployment, and RAG pipeline integration.
- Ongoing maintenance: $15,000–$25,000 annually, primarily index management and model version updates.
- Payback period: 5–8 weeks based on the savings model above.
- Three-year NPV (at 10% discount rate): approximately $1.8 million per 100-engineer cohort.

COLLABORATIVE AND ADVERSARIAL AGENT ARCHITECTURES

Collaborative and Adversarial Agent Architectures

The data flow diagram illustrates a single configuration, but the same infrastructure supports multiple interaction patterns between local and cloud agents, each offering distinct quality and cost characteristics.

Collaborative Architecture

Cloud and local agents divide the workflow by competency: the cloud agent plans and the local agent executes. Outputs from each stage are indexed, allowing each subsequent invocation to benefit from accumulated organizational knowledge. Over time, the vector store becomes a proprietary knowledge asset that reduces the context budget required for cloud inference calls.

Adversarial Architecture (Recursive Improvement)

In adversarial arrangements, two agents are tasked with opposing goals — one to generate a solution, one to find its failures. In the drone analysis context, this might mean:

- The local code generation agent produces a power-efficiency scoring algorithm.
- A second Ollama instance (or Claude) is instructed to identify edge cases, boundary conditions, and numerical instabilities in the implementation.
- Failures identified in adversarial review are added to the test suite, hardening the codebase incrementally.
- The cloud agent's critique is indexed alongside the original code, creating a searchable record of known failure modes that informs future generation tasks.

This recursive improvement loop produces compounding quality gains without compounding costs, because each cycle's outputs reduce the information gap that subsequent cloud calls must fill.

IMPLEMENTATION ROADMAP

Implementation Roadmap

Phase 1: Foundation (Weeks 1–3)

- Deploy Elasticsearch via Docker on a dedicated inference server or developer workstation fleet.
- Pull and benchmark Ollama models: evaluate qwen2.5-coder:7b, 14b, and 32b variants against the organization's code generation quality baseline.
- Index the primary codebase and documentation corpus; validate retrieval recall against a curated query set.

Phase 2: Pipeline Integration (Weeks 4–6)

- Integrate the RAG pipeline with existing developer tooling (VS Code, JetBrains, or CLI workflows).
- Configure the local proxy for productivity agent token interception and context trimming.
- Establish Playwright test harnesses for generated code validation.

Phase 3: Cloud Agent Orchestration (Weeks 7–9)

- Define task routing rules: decision logic for when queries escalate from local to cloud agents based on complexity scoring or explicit developer intent.
- Instrument the validation feedback loop: test result parsing, cloud agent dispatch, plan revision indexing.
- Deploy monitoring dashboards tracking per-agent token spend, retrieval latency, and code quality metrics.

Phase 4: Optimization and Domain Expansion (Weeks 10–12)

- Fine-tune retrieval parameters (chunk size, overlap, k, reranking) based on query performance data.
- Expand indexed corpora to domain-specific datasets (e.g., drone telemetry, motor specifications).
- Conduct adversarial architecture pilots in high-value domains; measure quality improvement against baseline.

CONCLUSION

Conclusion

The uniform routing of all AI workloads to cloud frontier models is a structural cost problem, not a technology problem. The models are capable; the architecture is misaligned with the cost structure of the work being performed.

Soledad Data Solutions ML Architects resolve this misalignment by designing systems where task complexity determines agent assignment. Local Ollama models handle the volume. Cloud frontier agents handle the reasoning. A locally deployed Elasticsearch RAG pipeline eliminates the redundant token spend that accounts for the majority of cloud AI billing in most enterprise environments.

The result is not a degradation of AI capability — organizations that have adopted this architecture report improvements in code quality, faster iteration cycles, and a vector knowledge store that compounds in value over time. The drone analysis example demonstrates that even highly specialized, data-intensive engineering domains benefit from this approach, with the adversarial validation loop producing accuracy improvements that no single-agent architecture achieves at comparable cost.

> **ENGAGE SOLEDAD DATA SOLUTIONS**
>
> To request a cost modeling engagement, architecture review, or pilot deployment scoped to your organization's AI infrastructure, contact the ML Architecture Practice at Soledad Data Solutions. Initial cost modeling assessments are provided at no charge for qualified enterprise clients.

APPENDIX: REFERENCE MODELS AND INFRASTRUCTURE

Appendix: Reference Models and Infrastructure

Recommended Ollama Models by Use Case

- Code Generation (7B tier): qwen2.5-coder:7b — best latency/quality balance for interactive workflows on RTX 3090 or equivalent.
- Code Generation (32B tier): qwen2.5-coder:32b, deepseek-coder-v2:16b — recommended for batch generation and complex multi-file tasks on A100 or dual-GPU workstations.
- Embedding: nomic-embed-text:v1.5 (768-dim), mxbai-embed-large:latest (1024-dim) — evaluated against domain corpora including drone telemetry and embedded C/C++ codebases.
- General Reasoning (local fallback): mistral:7b-instruct — useful for classification and routing tasks that do not require code generation capability.

Elasticsearch Configuration Notes

- Index: HNSW approximate nearest neighbor with ef_construction=128, m=16 for balanced recall and index build time.
- Hybrid retrieval: RRF (Reciprocal Rank Fusion) combining kNN and BM25 scores; outperforms linear interpolation on domain-specific corpora in internal benchmarks.
- Hardware minimum: 32GB RAM for a 10M-vector index; NVMe SSD recommended for segment merge performance on continuous indexing workloads.