

# The Debugging Tax: Cost Modeling for Agentic AI Built on Open-Weight Models

Oct 9, 2026 · @Tarik

## Abstract

Debugging, not inference, is usually the largest and least predictable cost of building an AI agent. Agents fail in multi-step, stochastic ways, so every fix needs replayed trajectories, repeated evaluation runs and engineer time.

This paper adds an explicit debugging term to the agent cost model. It argues that building on an open-weight model lowers that term in three ways: replay and evaluation become flat-rate compute instead of metered tokens, the model is fully inspectable and version-pinned, and recurring failures can be trained out rather than prompted around.

The advantage is largest for single-task agents. A narrow task bounds the failure surface, keeps evaluation suites small, lets a small fine-tuned model match a large general one, and makes debugging cost converge instead of grow. The paper closes with a worked example, the conditions under which the argument fails, and practical recommendations.

## 1. Introduction

Most agent budgets are built from inference price: tokens per task times price per token. That number is easy to get and usually wrong, because it prices the agent that works, not the agent you spend months making work.

An agent differs from a chatbot in one way that matters for cost: it takes many dependent steps. It plans, calls tools, reads results and acts again. A small error at step 3 can surface as a wrong action at step 12, so failures are delayed, compound and are hard to attribute.

Agents are also unreliable across repeated runs. On the  $\tau$ -bench retail benchmark, a GPT-4o agent solved about 61% of tasks in one try but only about 25% when it had to succeed on all eight tries ([Sierra,  \$\tau\$ -bench](#); [reliability framework, arXiv](#)). Closing that gap is debugging work, and it is paid for in replayed runs and engineer hours.

This paper covers three questions:

1. How should debugging appear in an agent's cost calculation?

- 2. How does building on an open-weight model (Llama, Qwen, DeepSeek, Mistral and similar) change that term?
- 3. Why is the change largest for single-task agents?

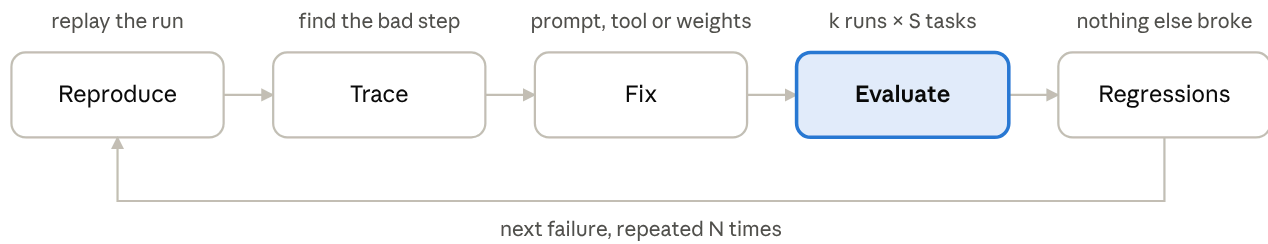
Throughout, "open model" means a model whose weights you can download, run, inspect and fine-tune. "Closed model" means one reached only through a vendor API.

## 2. What debugging means for an agent

Agent debugging is the work of finding why a trajectory went wrong and proving the fix holds across many runs. Five failure classes account for most of it.

| Failure class          | Typical symptom                              | Why it is costly to debug                               |
|------------------------|----------------------------------------------|---------------------------------------------------------|
| Tool-call errors       | Wrong tool, malformed JSON, missing argument | Cheap to see, but variants recur across many prompts    |
| Hallucinated arguments | Invented IDs, dates or file paths            | Looks valid; found only by checking against real state  |
| Planning loops         | Agent repeats a step or never stops          | Burns tokens on every reproduction; long traces to read |
| Context drift          | Forgets a constraint after many turns        | Appears only deep in long runs; slow to reproduce       |
| Nondeterminism         | Same input passes, then fails                | Needs many repeated runs to measure, let alone fix      |

Each fix follows the same loop: reproduce the failure, trace it to a step, change something (prompt, tool schema, model, code), re-run the evaluation suite, and check nothing else regressed. The loop runs dozens to hundreds of times before launch and keeps running after it.

**Every fix re-runs the whole evaluation suite**

agent debugging loop · 5 steps, repeated N times

The Evaluate step is where token cost concentrates: it multiplies repeats, suite size and trace length on every pass.

Two properties make this expensive. First, every pass through the loop re-runs the agent many times, because a single passing run proves little when  $\text{pass}^k$  is far below  $\text{pass}^1$ . Second, the fix is often indirect: with a closed model you can only change the words around the model, not the model.

### 3. A cost model for agent development

The total cost of an agent has four parts, and debugging should be its own line rather than hidden inside "engineering".

$$C_{\text{total}} = C_{\text{build}} + C_{\text{infer}} + C_{\text{debug}} + C_{\text{ops}}$$

The debugging term is the sum over every debugging iteration before and after launch:

$$C_{\text{debug}} = \sum_{i=1}^N \left( k \cdot S \cdot t \cdot p \cdot h_i \cdot w \right) + \lambda \cdot c_{\text{incident}}$$

| Symbol | Meaning                                           | Typical range         |
|--------|---------------------------------------------------|-----------------------|
| N      | Debugging iterations (fix, re-evaluate)           | 50–500 over a project |
| k      | Repeated runs per task, to measure reliability    | 3–10                  |
| S      | Tasks in the evaluation suite                     | 50–2,000              |
| t      | Tokens per trajectory (all steps, input + output) | 10k–200k              |
| p      | Blended price per token                           | see Section 6         |

| Symbol       | Meaning                                       | Typical range      |
|--------------|-----------------------------------------------|--------------------|
| $h, w$       | Engineer hours per iteration, hourly cost     | 2–20 h; \$80–200/h |
| $\lambda, c$ | Production failures that escape, cost of each | domain-specific    |

The compute part,  $k \times S \times t \times p$ , is where naive estimates break. One evaluation sweep of 300 tasks, 8 runs each, at 50k tokens per trajectory is 120 million tokens. A project that runs 200 sweeps processes 24 billion tokens on debugging alone, often more than its first year of production traffic.

The human part,  $h \times w$ , is usually larger still. It grows with how hard failures are to see and how many indirect levers an engineer must try. Both parts depend directly on the choice of model, which is the subject of the next section.

The ranges above are illustrative planning values, not measured industry averages. Teams should replace them with their own logs.

## 4. How open-weight models change the debugging term

Open weights lower both halves of the debugging term: compute becomes cheaper and flat-rate, and engineer hours fall because the model itself becomes debuggable.

**Cheaper, flat-rate evaluation.** Hosted open models are an order of magnitude cheaper per token. In February 2026, DeepSeek V3.2 cost \$0.14 / \$0.28 per million input/output tokens against \$1.75 / \$14.00 for GPT-5.2 and \$3.00 / \$15.00 for Claude Sonnet 4.x ([inference.net](https://inference.net)). Self-hosted, the marginal cost of one more evaluation sweep approaches zero once GPUs are paid for. Teams then run larger suites and more repeats, which finds bugs earlier.

**Deterministic replay.** With your own inference stack you control sampling, seeds, batch composition and the exact model build. A failing trajectory can be replayed token for token. Vendor APIs rarely guarantee this, so reproducing a flaky failure can take many paid attempts.

**Version pinning.** A hosted closed model can be updated or retired by the vendor. Each change can silently break passing behaviour and restart debugging. Open weights stay fixed until you choose to upgrade.

**Inspection.** You can read token log-probabilities, find where confidence dropped, and probe internal activations. A low-confidence tool argument is a precise bug report; a closed API usually shows only the final text.

**Fixing the model, not the prompt.** The biggest lever is fine-tuning. A recurring failure can be turned into training examples and trained out, instead of patched with ever-longer prompts. In Predibase's LoRA Land study, LoRA-tuned 7B models beat GPT-4 by about 10 points on average across 31 tasks, at under \$8 per fine-tune ([arXiv 2405.00732](#); [Predibase](#)). Longer prompts also raise every future token bill; a fix in the weights does not.

**What open models add.** These gains are not free. You take on GPU capacity, serving software, monitoring and upgrades. Self-hosting only beats per-token pricing at high utilisation. Open models may also trail the frontier on hard multi-step reasoning, which can create failures a stronger model would not have. Section 7 covers when these costs win.

## 5. Why single-task agents are the economic sweet spot

A single-task agent, one that does one job such as invoice extraction, ticket triage or a fixed refund workflow, shrinks every variable in the debugging term at once.

**Fewer steps, compounding less.** If each step succeeds with probability  $s$ , an  $n$ -step run succeeds with probability  $s$  to the power  $n$ . At  $s = 0.98$ , a 5-step task succeeds 90% of the time; a 30-step open-ended task succeeds 55% of the time. Narrow agents run short, fixed paths, so there are fewer places for errors to compound.

**A bounded failure surface.** A narrow agent uses a handful of tools on one kind of input. Its failure modes can be listed, and once each is fixed it tends to stay fixed. A general agent meets new combinations of tools and inputs indefinitely, so its debugging never converges.

**Small evaluation suites.** A narrow input space can be covered by a few hundred representative cases. A general agent needs thousands to approach the same confidence.  $S$  in the cost formula falls by an order of magnitude.

**Small models are enough.** A single task does not need broad world knowledge, so a 7B–30B open model can match a frontier model once tuned. LoRA Land found task-specific fine-tuning beat GPT-4 on 85% of the tasks it tested, and served 25 such adapters from one A100 GPU ([Predibase Fine-Tuning Index](#)). Small models also make every replay and sweep cheaper.

**Fine-tuning converges.** With a narrow scope, each batch of failure cases improves the very distribution the agent will see in production. Debugging becomes a data-collection loop with falling marginal cost, not an open-ended search for prompt wording.

| Factor              | General-purpose agent        | Single-task agent      |
|---------------------|------------------------------|------------------------|
| Steps per run       | 10–50+, variable             | 3–10, mostly fixed     |
| Tools               | Many, open set               | 2–5, fixed             |
| Eval suite needed   | Thousands of tasks           | Hundreds of tasks      |
| Model size needed   | Frontier                     | Small, fine-tuned      |
| Debugging over time | Keeps growing                | Converges              |
| Best lever          | Better prompts, bigger model | Fine-tune open weights |

## 6. Worked example

In this illustrative model, a single-task agent on an open model costs about \$73k to debug, against \$144k on a closed mid-tier model and \$1.66M for a general-purpose agent. Scope cuts the cost roughly tenfold; open weights then halve what remains.

| Scenario                           | Suite × runs × tokens | Iterations | Debug tokens | Compute cost | Engineer cost | Total debug cost |
|------------------------------------|-----------------------|------------|--------------|--------------|---------------|------------------|
| A. General agent, closed model     | 1,500 × 8 × 80k       | 300        | 288 B        | \$1,210k     | \$450k        | \$1,660k         |
| B. Single-task agent, closed model | 300 × 8 × 30k         | 120        | 8.6 B        | \$36k        | \$108k        | \$144k           |
| C. Single-task agent, open model   | 300 × 8 × 30k         | 120        | 8.6 B        | \$1.3k       | \$72k         | \$73k            |

### Assumptions:

- Closed model priced like Claude Sonnet 4.x at \$3 / \$15 per million input/output tokens; open model priced like hosted DeepSeek V3.2 at \$0.14 / \$0.28 ([inference.net](https://inference.net), Feb 2026).
- Agent traces are 90% input tokens, giving blended prices of \$4.20 and \$0.15 per million.
- Engineers cost \$150 per hour: 10 hours per iteration for A, 6 for B, 4 for C. C is lower because replay is deterministic and recurring failures are fixed by fine-tuning rather than prompt trial and error.
- C's fine-tuning runs add roughly \$1k, which does not change the result.

Three readings follow. Narrowing scope is the largest single saving, because it shrinks suite size, trace length and iterations together. Once scope is narrow, engineer time dominates, so the open model's value is less its token price than its debuggability. Prompt caching would cut A's and B's compute by perhaps half or more, but would not close the gap in engineer hours.

Production inference follows the same pattern. At 100,000 tasks per month and 30k tokens each, B costs about \$12,600 per month and C about \$450, before self-hosting.

## 7. When this does not hold

The open, single-task approach loses when its fixed costs outweigh the debugging savings. Five cases are common.

- **Low volume or short life.** A prototype or an agent run a few hundred times a month never repays MLOps setup. A closed API is cheaper.
- **Frontier-only capability.** Some tasks need reasoning no tuned small model reaches. A weaker model then creates new failures, raising debugging cost instead of lowering it.
- **No in-house ML skills.** Fine-tuning, serving and evaluation need people who can run them. Without them, engineer hours rise rather than fall. Hosted open-model APIs and managed fine-tuning narrow this gap.
- **Idle self-hosted GPUs.** Self-hosting beats per-token pricing only at high, steady utilisation. Spiky workloads are usually cheaper on a hosted open-model API.
- **Compliance and licensing.** Some open-weight licences restrict use, and regulated settings may require vendor guarantees. Check the licence before building on a model.

There is also a scope risk. A single-task agent that keeps gaining features becomes a general agent again, and its debugging cost stops converging.

## 8. Recommendations and conclusion

Budget for debugging explicitly, scope agents narrowly, and choose open weights when the task is stable and the volume is real.

1. **Add a debugging line to every agent budget.** Estimate  $N$ ,  $k$ ,  $S$  and  $t$  up front and track them as the project runs.
2. **Measure  $\text{pass}^k$ , not just  $\text{pass}^1$ .** Reliability across repeated runs is what debugging buys; measure it directly.

3. **Split general goals into single-task agents.** Several narrow agents with a simple router are cheaper to debug than one broad agent.
4. **Prototype on a frontier API, then move to open weights.** Use the strong model to find what works and to generate training traces, then fine-tune a small open model for production.
5. **Turn every fixed bug into training and test data.** Each failure becomes a regression test and, later, a fine-tuning example.
6. **Pin model versions and log full trajectories.** Replay is only possible if the model and the inputs are both fixed.
7. **Start on a hosted open-model API; self-host only at steady high volume.**

Debugging is the hidden majority of agent cost. It scales with how many runs each fix needs and how many hours each failure takes to understand. Open-weight models make those runs cheap and those failures visible and trainable. Single-task scope keeps the number of failures finite. Together, they turn agent debugging from an open-ended expense into a cost that converges.

## References

1. Sierra Research. [τ-bench: Benchmarking AI agents for the real world](#). 2024.
2. [Beyond pass@1: A Reliability Science Framework](#). arXiv 2603.29231, 2026.
3. Inference.net. [LLM API Pricing Comparison 2026](#). February 2026.
4. Zhao et al., Predibase. [LoRA Land: 310 Fine-tuned LLMs that Rival GPT-4, A Technical Report](#). arXiv 2405.00732, 2024.
5. Predibase. [LoRA Land: Open-Source LLMs That Beat GPT-4](#). February 2024.
6. Predibase. [The Fine-tuning Index](#).