Rootline: which evidence does an AI need to find the root cause, and can it fix it?

Affan Shaik · ORCID 0009-0008-6203-3226

Technical report v0.1 · pilot (n=42 snapshots) · results are from a pilot and carry wide confidence intervals

Abstract. Rootline is an open, reproducible benchmark and replay app for evidence-grounded AI diagnosis of microservice failures. It injects controlled faults into the OpenTelemetry Demo, freezes each incident's telemetry into Parquet snapshots, and replays them to free and open models under controlled evidence conditions. This first release asks two questions: which telemetry signals models need, and whether they can tell network faults from application faults. On 42 snapshots, the best model (`gpt-oss-120b@groq`) reached 9.5% top-1 accuracy with combined evidence (95% CI 2.4%–19%). Models were confident far beyond their accuracy (mean top confidence 84.1%–95%), and every model claimed a fault on every healthy snapshot. This first release tests two of six pre-stated hypotheses (RQ1, RQ2); the other four are held for v0.2. 2 could be tested; 1 met their threshold and 0 remained significant after Holm correction. All results are reported, including failed hypotheses.

1. Introduction

Large language models are increasingly proposed for incident diagnosis, but most evaluations score free-text answers with another model and say little about *which evidence* the model used. Rootline grades by evidence and by execution: every evidence line has a stable ID that answers must cite, and the chosen fix can be applied to the live system (remediation is scored from v0.2). We evaluate only freely available open-weight models and free-tier APIs; results characterise what budget-constrained teams can achieve today, not the frontier.

Prior work. AIOpsLab [1] deploys microservice environments, injects faults and evaluates agents that act on the live system. ITBench [2] collects real-world IT automation scenarios for agents across SRE, compliance and FinOps tasks. RCAEval [3] provides 735 failure cases from three microservice systems with metrics, logs and traces, and fifteen root-cause-analysis baselines. Rootline is smaller and narrower: it holds the prompt budget fixed while varying which signals fill it, asks whether a model can place a fault in the network or the application, checks that cited evidence exists, and adds real Wi-Fi faults on hardware next to their emulated counterparts.

2. Method

2.1. Testbed and faults

The OpenTelemetry Demo runs on Docker Compose, pinned to a release tag and image digests recorded in every snapshot manifest. Fifteen incident types: ten application faults from the demo's feature flags, four network faults (Toxiproxy latency, connection resets and bandwidth limits, and `tc netem` packet loss) and one healthy control. Each incident runs a 5-minute baseline, 5 minutes of fault, and a snapshot of the window from 5 minutes before to 5 minutes after onset; the controller then clears the fault and checks recovery. Ground-truth labels are stored apart from snapshots and never reach a model.

2.2. Evidence conditions and perturbations

Every prompt is at most 6,000 tokens (tiktoken `cl100k_base`), so conditions differ in signal, not size: metrics only (per-service rates, error ratios and latency quantiles in 30-second buckets, as change from baseline), traces only (20 slowest and 20 errored traces as critical paths), logs only (Drain3 templates ranked by rate change) and combined (a third each). Perturbation P1 replaces every service name, including inside identifiers, with a consistent anonymous one. All fault and flag names are redacted from every pack.

2.3. Models, output and scoring

Models answer at temperature 0 with a strict JSON contract: layer, up to three ranked hypotheses with confidence and cited evidence IDs, and one action from a fixed menu (restart, disable a feature, clear a network fault, no action). Invalid output gets one retry with the validation error; a second failure counts as an abstention. We

report top-1 and top-3 accuracy on the (service, category) pair, layer attribution, evidence validity (cited IDs that exist and belong to the root-cause service), healthy false alarms, Brier score and 10-bin expected calibration error.

2.4. Statistics

McNemar's exact test for paired comparisons on the same snapshots, Fisher's exact test for unpaired ones, 10000-resample bootstrap confidence intervals (seed 20261107), and Holm correction across the hypotheses tested in this release (RQ1 and RQ2). RQ3 (name reliance), RQ4 (cascade), RQ5 (remediation) and RQ6 (classical baseline) are deferred to v0.2 and not tested here.

3. Results

RQ	Hypothesis	Effect	p (Holm)	Result
RQ1	Combined signals beat the best single signal by >= 10 pp top-1	−0.024	1	not supported
RQ2	Layer accuracy is >= 15 pts lower on network faults than application faults	0.3	0.165	met, not significant
RQ3	Anonymising service names drops top-1 by >= 20% relative	–	–	deferred to v0.2
RQ4	A cascade keeps >= 90% of the strongest model's accuracy with <= 50% of its tokens	–	–	deferred to v0.2
RQ5	Remediation success is >= 20 pts below diagnosis accuracy	–	–	deferred to v0.2
RQ6	The best LLM beats LightGBM on unseen fault types but not on seen ones	–	–	deferred to v0.2

Table 1: The six pre-stated hypotheses. v0.1 tests RQ1 and RQ2; the rest are deferred to v0.2. Effects and tests are defined in the method; details per row are in results.json.

Model	Tier	Top-1	Top-3	Layer	Evidence validity	ECE
`gpt-oss-120b@groq`	C	9.5%	19%	66.7%	16.9%	0.746
`gemma3-4b-q4@ollama`	A	4.8%	9.5%	47.6%	14.8%	0.837
`qwen3-8b-q4@ollama`	B	2.4%	2.4%	31%	17%	0.926

Table 2: Combined evidence, real service names (pilot (n=42 snapshots)).

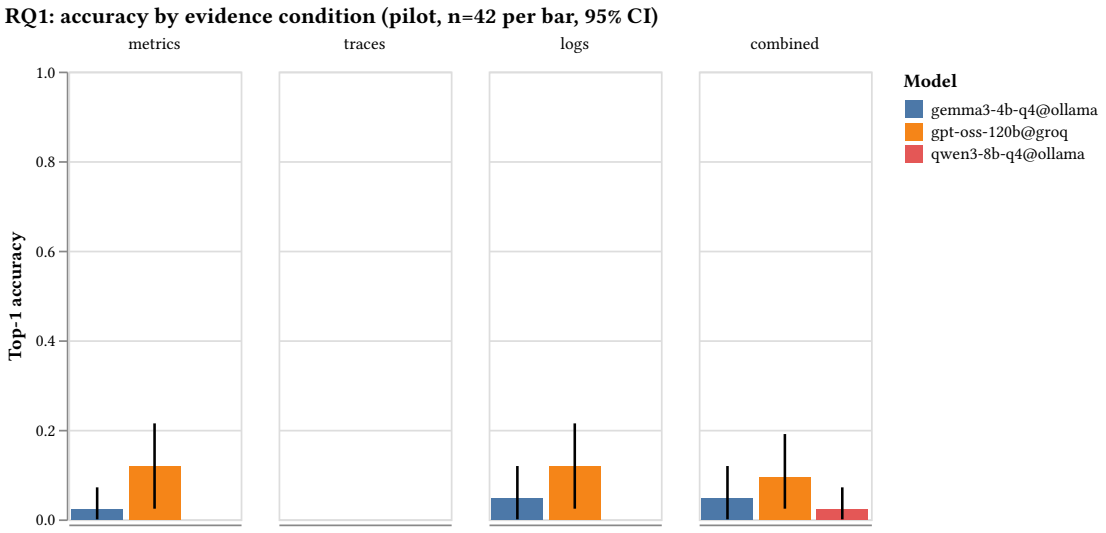


Figure 1: RQ1: top-1 accuracy by evidence condition, with 95% bootstrap confidence intervals.

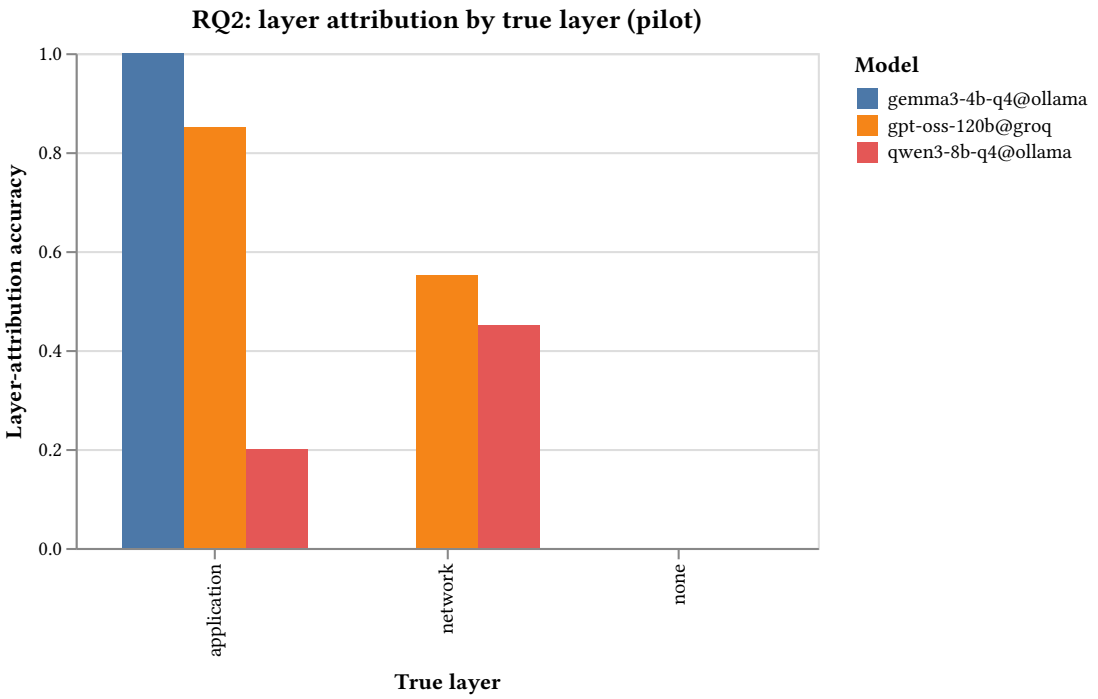


Figure 2: RQ2: layer-attribution accuracy by the true layer of the fault.

Baseline	Scheme	n	Top-1	Service accuracy
lightgbm	unseen	42	4.8%	21.4%
lightgbm	seen	42	21.4%	35.7%
isolation_forest	unsupervised	40	–	25%

Table 3: Classical baselines. Isolation Forest predicts no category, so it has no top-1.

3.1. Results by evidence condition

Model	Top-1				Layer			
	Metrics	Traces	Logs	Combined	Metrics	Traces	Logs	Combined
`gemma3-4b-q4@ollama`	2.4%	0%	4.8%	4.8%	45.2%	42.9%	42.9%	47.6%
`gpt-oss-120b@groq`	11.9%	0%	11.9%	9.5%	42.9%	35.7%	26.2%	66.7%
`qwen3-8b-q4@ollama`	0%	0%	0%	2.4%	35.7%	33.3%	50%	31%

Table 4: Every model on every snapshot under each evidence condition, real service names. All conditions share the same 6,000-token budget.

3.2. Exploratory: either end of a broken link

Not pre-registered; it does not change any result above. For a fault on a link, the pre-registered scoring expects the downstream service (payment for checkout ⟶ payment). Models often name the caller. Here the same answers are re-scored with one change: for link faults, naming either end of the link with the right category counts as correct.

Model	Top-1	Top-1, either end	Link faults	Link top-1	Link top-1, either end
`gemma3-4b-q4@ollama`	4.8%	4.8%	22	0%	0%
`gpt-oss-120b@groq`	9.5%	16.7%	22	0%	13.6%
`qwen3-8b-q4@ollama`	2.4%	9.5%	22	0%	13.6%

Table 5: Combined evidence, unperturbed. Exploratory re-scoring of the same answers; the hypothesis tests use the pre-registered rule.

3.3. Manipulation check

Does each injected fault show in its own snapshot? For every snapshot, the spans that should carry the fault (the caller's calls across a broken edge, or the root-cause service's own spans) before and after fault onset.

Snapshot	Measured	Errors before	Errors after	p95 before (ms)	p95 after (ms)
adHighCpu-r1	ad spans	0%	0%	1	1
adHighCpu-r2	ad spans	0%	0%	1	1
emailMemoryLeak-r1	email spans	0%	0%	276	219
emailMemoryLeak-r2	email spans	0%	0%	50977	48886
healthy-r1	checkout calls to payment	0%	0%	35	143
healthy-r2	checkout calls to payment	0%	0%	11	66
hwContention-r1	checkout calls to payment	0%	0%	37	173
hwContention-r2	checkout calls to payment	0%	0%	30	165
hwContention-r3	checkout calls to payment	0%	0%	150	1949
hwContention-r4	checkout calls to payment	0%	0%	40	161
hwWeakSignal-r1	checkout calls to payment	0%	0%	166	43
hwWeakSignal-r2	checkout calls to payment	0%	0%	40	76
hwWeakSignal-r3	checkout calls to payment	0%	0%	39	44
hwWeakSignal-r4	checkout calls to payment	0%	0%	44	45
hwWifiOutage-r1	checkout calls to payment	0%	5.8%	55	8157
hwWifiOutage-r2	checkout calls to payment	0%	5.8%	35	4579
hwWifiOutage-r3	checkout calls to payment	0%	1.5%	44	4207
hwWifiOutage-r4	checkout calls to payment	0%	9.3%	36	12490
imageSlowLoad-r1	frontend-proxy spans	0.2%	0.3%	521	522
imageSlowLoad-r2	frontend-proxy spans	0%	0%	162	2827
intlShippingSlowdown-r1	shipping spans	0%	0%	917	877
intlShippingSlowdown-r2	shipping spans	0%	0%	747	672
kafkaQueueProblems-r1	fraud-detection spans	0%	0%	85	1001
kafkaQueueProblems-r2	fraud-detection spans	0%	0%	94	1001
[illegible]	[illegible]	0%	[illegible]	[illegible]	[illegible]
[illegible]	[illegible] spans, product-catalog	[illegible]	[illegible]	[illegible]	26[illegible]

Table 9: Fault-window per snapshot, from the frozen telemetry only (no model).

4. Hardware arm: real Wi-Fi faults

Two Arduino UNO Q boards route the checkout → payment edge over a real Wi-Fi link, with three faults triggered on the boards: reduced transmit power, 2.4 GHz channel contention and a 30-second Wi-Fi outage. Each is compared with its emulated counterpart.

Model	Real fault	Emulated	Top-1 real	Top-1 emulated	p (Fisher)
gemma3-4b-q4@ollama	hwWeakSignal	netemLoss	0%	0%	1
gemma3-4b-q4@ollama	hwContention	toxiLatency	0%	0%	1
gemma3-4b-q4@ollama	hwWifiOutage	toxiReset	0%	0%	1
gpt-oss-120b@groq	hwWeakSignal	netemLoss	0%	0%	1
gpt-oss-120b@groq	hwContention	toxiLatency	0%	0%	1
gpt-oss-120b@groq	hwWifiOutage	toxiReset	0%	0%	1
qwen3-8b-q4@ollama	hwWeakSignal	netemLoss	0%	0%	1
qwen3-8b-q4@ollama	hwContention	toxiLatency	0%	0%	1
qwen3-8b-q4@ollama	hwWifiOutage	toxiReset	0%	0%	1

Table 7: Emulation validity: a large gap means emulated faults are not a fair stand-in for real ones.

5. Discussion

5.1. Mostly wrong, and confidently so

The headline is negative. With combined evidence the strongest model named the right service and category first in 9.5% of snapshots, yet its top hypothesis carried a mean stated confidence of 84.1%. The small models were more confident still (88.5% and 95%) and less often right (4.8% and 2.4%). A team that acted on the confidence field would be misled: none of the models is calibrated well enough for its confidence to separate good answers from bad ones.

5.2. A layer answer can be a default, not a judgement

Layer accuracy looks better than service accuracy, but the answers show why. Gemma 3 4B answered "application" on all 42 of 42 snapshots, so its 100% on application faults and 0% on network faults measure a habit, not a skill. Qwen3 8B leaned the other way (27 network answers against 15 application answers). Only the strongest model split its answers (28 application, 14 network) in a way that tracked the faults. Layer accuracy should therefore be read next to the answer distribution, and v0.2 should report balanced accuracy.

5.3. The caller gets the blame on broken links

On faults that sit on a link, the strongest model got the layer right in more than half of them and the kind of fault in a few, and in most it named the caller (checkout) as the root cause rather than the service at the far end (payment), which the pre-registered rule scores. Under that rule no model got any link fault exactly right. The exploratory re-scoring in Section 3.2 shows how much this one rule decides. Both readings are defensible: from the caller's spans, the slow or failing dependency is visible but the fault itself is not. A benchmark that wants to separate "where it hurts" from "what broke" has to ask for both.

5.4. Real radio faults read more like network faults

For the strongest model, layer accuracy on the hardware faults was 75% (12 snapshots), against 25% on the emulated network faults (8 snapshots). This was not tested and the samples are small, but it fits the manipulation check: the Wi-Fi outage and contention produced clearly slow and failing calls, while several emulated faults (connection resets on a link that keeps one connection open, light packet loss) barely moved the evidence. An emulated fault that leaves no trace cannot be diagnosed, so emulation validity should be judged on faults that are visible in both settings.

5.5. Nothing was ever “nothing”

On the healthy snapshots, across all evidence conditions, the strongest model claimed a fault in 8 of 8 answers and blamed `checkout` most often; every other model also claimed a fault in every answer. Ordinary background noise in a live shop, such as the occasional failed request or a service that logs errors by design, was enough to produce a confident diagnosis. A useful assistant has to be able to say “no fault”; v0.2 needs more healthy snapshots to measure this properly.

5.6. More evidence did not help

RQ1 expected combined evidence to beat the best single signal. For the strongest model, combined evidence scored 9.5% top-1 against 11.9% for logs and 11.9% for metrics alone, while traces alone scored 0%. Because every condition has the same budget, the combined pack carries only a third of each signal. With answers this rarely right, the difference is within noise, but the design question stands: at a fixed budget, breadth costs depth.

5.7. What v0.2 should change

Report balanced layer accuracy and ask for both the symptom and the cause on link faults; add healthy snapshots; strengthen or replace the faults that left no trace; run each job more than once, since hosted models were not fully deterministic at temperature 0; and test the four held-back questions (name reliance, cascades, remediation and the classical baseline).

6. Limitations

This is a pilot: two replicas per incident type give wide confidence intervals, and this release tests two of six pre-stated hypotheses. One testbed on one machine stands in for production systems that are larger and noisier, and a few services were given more memory to keep the testbed itself up; snapshots that overlapped a testbed crash were rerun. Models may have seen the demo’s public code, and service names reach the model (name reliance, RQ3, is v0.2). Several injected faults left little trace in the evidence (Section 3.3); they are scored as they are. On hardware, lowering transmit power left no visible trace with the board near the router, and the relay sent no telemetry of its own. Free-tier limits changed during 2026, so provider choices are recorded per run.

7. Reproducibility

Code (Apache-2.0) and data (CC BY 4.0) are public. `make reproduce` regenerates every table and figure in this report from the published runs without network access. Inputs: 504 runs (sha256 `4245100bc21f`), 0 remediation trials.

AI tools assisted with writing the code.

References

[1] Y. Chen *et al.*, “AIOpsLab: A Holistic Framework to Evaluate AI Agents for Enabling Autonomous Clouds,” 2025. [Online]. Available: https://arxiv.org/abs/2501.06706

[2] S. Jha *et al.*, “ITBench: Evaluating AI Agents across Diverse Real-World IT Automation Tasks,” 2025. [Online]. Available: https://arxiv.org/abs/2502.05352

[3] L. Pham, H. Zhang, H. Ha, F. Salim, and X. Zhang, “RCAEval: A Benchmark for Root Cause Analysis of Microservice Systems with Telemetry Data,” 2024. [Online]. Available: https://arxiv.org/abs/2412.17015