LLM Eval & Observability Cheat Sheet

The one-reference guide to measuring and monitoring LLM systems in production — metrics, eval types, trace fields, dashboards, alerts, and failure modes.

A free guide from ProductionAIEngineer.com · v1.0 · Source file — deliver as the lead-magnet PDF (print-friendly, 2 pages).

If you can't measure it, you can't ship it responsibly. This is the reference you pin to the wall: what to evaluate, what to log, what to graph, and when to page someone. Model-agnostic — the patterns hold whether you run OpenAI, Anthropic, or open models.

---

1. The metrics map

Quality (generation)

Metric	What it tells you	How
Faithfulness / groundedness	Answer is supported by context (no hallucination)	LLM-as-judge vs source; NLI
Answer relevance	Answer addresses the question	LLM-as-judge; embedding similarity
Correctness	Matches ground truth	Exact/semantic match vs golden set
Completeness	Covers all parts of the question	Rubric / judge
Coherence & tone	Reads well, on-brand	Rubric / judge

Retrieval (RAG)

Metric	What it tells you
Recall@k	Was the right chunk retrieved?
Precision@k / context precision	How much retrieved context was relevant?
MRR / NDCG	Is the best chunk ranked high?
Context sufficiency	Is the answer *derivable* from context?

Agents

Metric	What it tells you
Task success rate	Did it complete the task?
Tool-call accuracy	Right tool, right args?
Steps to completion	Efficiency / loop health

Metric	What it tells you
Recovery rate	Handled tool errors gracefully?

Safety & compliance

Metric	What it tells you
Refusal correctness	Refuses when it should (and only then)
Toxicity / PII leakage	Unsafe or sensitive output rate
Injection resistance	Withstands adversarial inputs
Jailbreak rate	% of adversarial prompts that succeed

Operations

Metric	Target guidance
Latency p50 / p95 / p99	Set per use case; watch p95
Tokens in/out per request	Cost driver; trend it
Cost per request / per user / per day	Budget enforcement
Error & timeout rate	< 1% healthy for most apps
Cache hit rate	Higher = cheaper + faster
Throughput (req/s)	Capacity planning

---

2. Eval types (when to use each)

Type	When	Cadence
Unit evals	Single prompt/function correctness	Every commit (CI)
Offline / batch	Score a config on the golden set	Every change + nightly
Regression	Prevent "fixed one thing, broke another"	Every PR (gate merge)
LLM-as-judge	Scale subjective scoring	In offline + online
Human eval	Calibrate judges; final sign-off	Weekly sample
A/B / online	Real user impact of a change	Continuous in prod
Red-team / adversarial	Safety + injection resistance	Pre-launch + periodic

Golden dataset rules

- 25–50 cases to start; grow from real failures and user feedback.
- Version it with the code; every prod incident becomes a new case.

- Cover: happy path, edge cases, out-of-scope (should refuse), adversarial.

LLM-as-judge cautions

- Calibrate against human labels; report agreement (%).
- Guard against position bias, verbosity bias, self-preference.
- Use a strong judge model + a clear rubric; pin the judge version.

---

3. The canonical trace (log these fields on every call)

```
{
  "request_id": "uuid",              // correlate across services
  "session_id": "uuid",              // per user/conversation
  "user_id_hash": "sha256",          // pseudonymous, never raw PII
  "timestamp": "ISO-8601",
  "route": "rag_query | agent_task | chat",
  "input": "user query (redacted)",
  "retrieved": [                      // RAG only
    { "chunk_id": "…", "source": "…", "score": 0.83 }
  ],
  "prompt_tokens": 1234,
  "completion_tokens": 256,
  "total_tokens": 1490,
  "model": "provider/model@version",
  "temperature": 0.2,
  "prompt_version": "rag_v7",
  "latency_ms": { "retrieval": 120, "rerank": 40, "generation": 900, "total": 1080 },
  "cost_usd": 0.0031,
  "cache_hit": false,
  "tools_called": [                   // agents only
    { "name": "search", "args_hash": "…", "ok": true, "ms": 210 }
  ],
  "output": "model answer (redacted)",
  "citations": ["source#section"],
  "eval": { "groundedness": 0.92, "relevance": 0.88 }, // if scored inline/async
  "feedback": { "thumb": null, "edited": false },
  "error": null,
  "status": "ok | refused | error | timeout"
}
```

> **Rule:** redact PII/secrets *before* logging. Store the hash, not the human. Traces are a data-privacy surface.

---

4. Dashboards (the panels that matter)

Health (real-time)

- Request rate, error rate, timeout rate.
- Latency p50/p95/p99 (total + per stage).
- Cost/hour + tokens/hour.

Quality (rolling)

- Groundedness / relevance trend (from online judge sample).
- Refusal rate (spikes = retrieval broke or corpus gap).
- Retrieval hit-rate trend (RAG).
- Thumbs-down rate + top negative-feedback clusters.

Cost

- Cost/request, cost/user, cost/day vs budget.
- Cache hit rate.
- Token distribution (spot prompt bloat).

Agents

- Task success rate, avg steps, tool-error rate, runaway-loop count.

---

5. Alerting guidance (thresholds to start from)

Alert	Suggested trigger	Why
Error rate	> 2% for 5 min	Provider/outage/regression
Latency p95	> 2× baseline for 10 min	UX degradation
Cost spike	> 1.5× daily budget pace	Runaway loop / abuse
Refusal rate	> 2× baseline	Retrieval or corpus break
Eval regression	groundedness ↓ > 5 pts	Bad prompt/model change
Injection/jailbreak	any confirmed success	Security incident
Cache hit rate	↓ > 30% suddenly	Cache misconfig / cost risk

> Page on **user-facing** and **cost/security** alerts. Ticket the rest. Alert fatigue kills observability.

---

6. Failure modes → what to check

Symptom	First thing to check
Hallucination	Retrieved context + groundedness score

Symptom	First thing to check
Wrong/vague answers	Retrieval hit-rate, then prompt
Latency spike	Per-stage latency in the trace
Cost spike	Tokens/request + tool-loop count
Rising refusals	Corpus freshness + retrieval threshold
Intermittent errors	Timeout/retry config, provider status
Agent won't stop	Loop bound + stop condition
Regressions after deploy	Diff prompt_version; run regression suite

---

7. Regression testing (make quality a CI gate)

- [ ] Golden set runs on every PR that touches prompts, models, retrieval, or chunking.
- [ ] Merge blocks if any core metric drops beyond a set threshold.
- [ ] Every production incident adds a case to the set.
- [ ] Prompts and datasets are versioned; results are stored and diffable.
- [ ] Judge model + version pinned so scores are comparable over time.

---

8. Practical examples

Minimal offline eval loop (pseudocode)

```
score = 0
for case in golden_set:                    # 25–50 versioned cases
    ctx = retrieve(case.query)             # capture chunk ids + scores
    ans = generate(case.query, ctx)        # capture tokens, latency, cost
    g = judge_groundedness(ans, ctx)       # LLM-as-judge, calibrated
    r = judge_relevance(ans, case.query)
    log_trace(case, ctx, ans, g, r)        # the canonical trace above
    score += (g ≥ 0.8 and r ≥ 0.8)
print(f"pass rate: {score}/{len(golden_set)}")
```

Online sampling: score ~1–5% of live traffic with the same judges; alert on trend.

Human calibration: weekly, label 20 cases by hand; compare to the judge; if agreement < 80%, fix the rubric.

Your next step

1. **Weekly production lessons** → subscribe to **Production AI Notes**.
2. **Fix the system you're measuring** → grab the free **Production RAG Checklist**.

3. **Prove it in interviews** → the **AI Engineer Interview & Portfolio Kit** turns your eval harness into a portfolio project and system-design answer.

Built by Gururaj · ProductionAIEngineer.com · Ship AI that survives real users. Public/synthetic examples only. Opinions my own.
