Production LLM Evaluation and Regression: An Engineering Setup Guide

Golden datasets, regression suites, evaluator models, drift detection, and handoff criteria for moving an LLM system into production.

Vishal Shukla, VP of Technology, ViitorCloud Enterprise

Enterprise practice of ViitorCloud Technologies

ISO 27001 certified since 2009

enterprise.viitorcloud.com

Why This Exists LLM applications fail in production in ways that traditional software does not. The model still responds. The responses still look plausible. Latency is in budget. Token usage is normal. And quality has quietly degraded against what the system was tested on, or the inputs have shifted, or the model is now hallucinating against a content estate that has changed underneath it.

None of those failure modes are caught by standard monitoring. They require an evaluation infrastructure that runs alongside the application from before launch and continues running for the life of the system. Engineering teams that ship without it discover the gap when the first user complaint arrives.

This is the setup guide for that infrastructure. Five parts. Built for a Python-native engineering team shipping an enterprise LLM application. Sized to be implementable inside a real sprint plan, not to look comprehensive in a slide deck.

How to Use This

Read in order. The parts depend on each other. The golden set in Part 1 is what the regression suite in Part 2 runs against. The evaluator model in Part 3 is what scores the regression suite. The drift detection in Part 4 catches what the regression suite missed. The handoff criteria in Part 5 control when the system meets a customer.

Each part shows a minimum viable setup, a mature setup, the common failure modes, and what "good" looks like. Most teams start at minimum viable, ship to internal users, and grow to mature before going customer-facing. That sequence is the right one.

Part 1. The Golden Evaluation Set

The golden set is the foundation. Everything else depends on it.

WHAT IT IS

A versioned, owned set of input-output pairs that defines what "correct" looks like for your LLM application. The single source of truth for whether the model is working.

WHY IT MATTERS

Without a golden set, you cannot tell if a model change improved the system or broke it. Eyeballing demos does not scale. The first failed model swap or prompt change without a golden set is the moment the team realises this.

MINIMUM VIABLE SETUP

- 25 to 50 input-output pairs covering the most important use cases.

- Stored in git, version-controlled like code.

- Owned by the business team that owns the application, not by the ML team.

- Each pair tagged with use case, expected behaviour, and edge case category.

- Reviewed at least once a quarter.

MATURE SETUP

- 200 to 500 pairs covering normal cases, known failure modes, and adversarial cases.

- Continuous expansion from real production failures, not synthetic examples.

- Categorised by difficulty, risk, and customer impact.

- Multiple expected outputs for non-deterministic queries. A range of acceptable answers, not a single string match.

- Low-scoring production outputs flagged for human review and considered for inclusion in the next set version.

COMMON FAILURE MODES

- The set is built once at launch and never updated. Six months later, it does not reflect what production actually does.

- The set is owned by the ML team, not by the business. The labels drift away from what users actually need.

- The set is synthetic. It does not catch real failure patterns because real failure patterns are not in it.

- The set is too small. Statistical noise drowns the signal.

WHAT GOOD LOOKS LIKE

- The set has grown from around 25 cases at launch to 300+ inside the first year, with growth driven by real production traffic and real incidents.

- The business owns the labels and reviews them quarterly.

- Every model or prompt change is evaluated against the set automatically.

- The set is the answer to "is the model working" for both engineering and the business.

Part 2. The Regression Suite

The golden set is the source of truth. The regression suite is the enforcement.

WHAT IT IS

An automated test suite that runs the golden set against the current model and prompt configuration on every change, and blocks the change if the system regresses.

WHY IT MATTERS

Regression in LLM systems is silent. The model still responds, the responses still look plausible, and only the metrics know that quality has degraded. Without a regression suite, the first signal of regression is a user complaint, which is too late.

MINIMUM VIABLE SETUP

- Pytest-style test integration. DeepEval and Braintrust both support this pattern.

- Pull request triggers a run against the full golden set.

- Pass/fail thresholds defined per metric (accuracy, faithfulness, relevancy, safety).

- A regression below baseline blocks the merge.

- Test results posted to the PR as a comment.

MATURE SETUP

- CI gate plus deployment gate. The CI gate runs at PR time. The deployment gate runs again at deploy time against the production-shaped configuration.

- Metric thresholds tightened on a documented cadence as the system matures. Looser thresholds at week one, tighter at month six.

- Adversarial test suite separate from the functional suite. Prompt injection, jailbreak attempts, data extraction attempts. Run on a documented cadence aligned with OWASP LLM Top 10 and NIST AI RMF.

- Results stored over time so the team can see trends, not just snapshots.

- Production monitoring feeds new failures back into the regression suite within a defined window (a week is a defensible target).

COMMON FAILURE MODES

- The thresholds are too loose. Regressions slip through because the gate accepted them.

- The thresholds are too tight. Every change blocks the deploy and the team starts disabling the gate.

- The suite runs but no one reads the output. The signal is there. The operating habit is not.

- The suite covers the happy path only. The first adversarial attack in production exposes the gap.

WHAT GOOD LOOKS LIKE

- The CI gate has caught at least one regression that would have shipped without it, inside the first three months.

- The team can produce a chart of accuracy, faithfulness, and adversarial pass rate over time.

- New failure patterns from production are added to the suite within a week.

- The threshold history is documented.

Part 3. Automated Evaluator Models (LLM-as-Judge)

This is the scale layer. Human evaluation does not scale. LLM-as-Judge does, if you trust it.

WHAT IT IS

Using an LLM to grade the outputs of another LLM against a defined rubric. The evaluator scores faithfulness, relevancy, safety, helpfulness, and any custom metric the application requires.

WHY IT MATTERS

Production traffic is too high-volume for human evaluation. LLM-as-Judge is the only way to evaluate quality at scale. But naive LLM-as-Judge has known biases (verbosity inflation, self-preference, position effects) that will produce systematically wrong scores if not corrected.

MINIMUM VIABLE SETUP

- Use a frontier model as the judge (GPT-4 class, Claude Sonnet class). A stronger judge produces more reliable scores.

- Write the rubric explicitly. "Score 1 to 5 on faithfulness, where 1 means..." Vague rubrics produce vague scores.

- Calibrate the judge against a human-scored subset before trusting it for production scoring. Disagreement on 100 cases is your baseline.

- Use pairwise comparison rather than absolute scoring where possible. Comparison is more reliable than scoring.

MATURE SETUP

- Open-source judge model where data residency or cost matters. Prometheus, a 13B-parameter evaluator LLM, achieves Pearson correlation of 0.897 with human evaluators, on par with GPT-4's 0.882.

- Bias detection in the judge. Test for verbosity inflation, self-preference, and position effects. Document the bias profile.

- Ensemble judging. Multiple judges score the same output. Disagreement triggers human review.

- Custom rubrics per use case. Faithfulness for RAG, helpfulness for support copilots, safety for any customer-facing surface.

COMMON FAILURE MODES

- The judge is weaker than the model it is judging. The judge cannot reliably detect errors it would itself make.

- The rubric is vague. The judge interprets it differently across runs.

- Bias goes uncorrected. Verbosity inflation makes long, mediocre answers look better than short, correct ones.

- The judge is trusted without calibration. The first production failure exposes that the judge's scores did not predict the failure.

WHAT GOOD LOOKS LIKE

- Judge calibration data is documented. The team knows the judge's correlation with human scoring.

- Judge biases are documented and corrected for in the rubric.

- The judge runs as part of the regression suite and as part of production monitoring.

- The team has at least one example of catching a real failure because the judge flagged it.

Part 4. Drift Detection

This is the early warning system. Production monitoring beyond uptime.

WHAT IT IS

Continuous measurement of input distribution, output distribution, and quality metrics in production, with alerts when any of them drift outside expected ranges.

WHY IT MATTERS

LLM systems drift in ways that look fine to traditional monitoring but are not fine. The model still responds in latency budget. Token counts look normal. But the inputs are different from what the system was tested on, or the outputs have started to skew, or quality has degraded against the golden set. The drift is invisible without active measurement.

MINIMUM VIABLE SETUP

- Sample 1 to 5 percent of production traffic into an evaluation log.
- Run the LLM-as-Judge against the sampled traffic on a daily cadence.
- Alert on metric regression below a defined threshold.
- Track input distribution metrics: query length, topic distribution, user segment. Alert on significant shift.
- Track output distribution metrics: response length, refusal rate, citation rate (for RAG). Alert on significant shift.

MATURE SETUP

- Higher sample rate (10 to 25 percent) on sensitive paths.
- Embedding-based input drift detection. New query types that do not match the training distribution surface as outliers.
- Cohort analysis. Quality by user segment, by use case, by time of day. Drift often shows up in one cohort first.
- Hallucination rate measured continuously. Under 3 percent on monitored queries is a defensible benchmark for production-grade enterprise copilots.
- Automated remediation for known drift patterns. Cache invalidation, prompt rollback, model rollback.

COMMON FAILURE MODES

- Monitoring is built on latency and uptime only. Quality drift is invisible.
- Sample rate is too low. Statistical noise drowns the signal.
- Alerts fire too often. The team starts ignoring them.
- The team monitors but does not act. Drift is logged, not addressed.

WHAT GOOD LOOKS LIKE

- The team can show drift on the metrics that matter (quality, hallucination, refusal rate) over the last 90 days.
- At least one drift event has been caught and addressed before users noticed.
- The alert rate is sustainable. Not so noisy that alerts get muted.
- Production monitoring data feeds the golden set on a documented cadence.

Part 5. Handoff Criteria for Customer-Facing

This is the gate that controls scope. The moment an LLM moves from internal users to customer-facing is the moment the cost of failure changes by an order of magnitude.

WHAT IT IS

A checklist of conditions that have to be met before the LLM can serve external customers. The definition of "ready for prime time."

WHY IT MATTERS

Internal-only LLMs can fail without consequence beyond an embarrassed engineering team. Customer-facing LLMs fail in public, fail under regulatory scrutiny, and fail in front of the people paying for the product. The handoff is the highest-stakes change the system goes through.

PASS CRITERIA

1. The golden set has been stable for 30 days without regression. Not 30 days of running. 30 days of running without a failed regression event.
2. Adversarial test pass rate is at or above the documented threshold. Prompt injection, jailbreak, and data extraction tests have been run and pass.
3. Production monitoring is live and alerting. Not configured. Live. With at least one test alert successfully routed and acknowledged.
4. The incident response playbook for AI-specific incidents exists and has been rehearsed. Hallucination at scale, adversarial attack, model drift, regulatory enquiry. Each has a named owner and a documented first-hour response.
5. A staged rollout plan exists. Customer-facing launch goes to a small cohort first, with traffic ramped on a documented schedule based on monitoring signal.
6. A rollback path exists, and has been tested. The team can revert to the previous model, prompt, or system state inside a defined RTO.
7. Legal and compliance have signed off on the customer-facing version, including the user-facing disclaimers and the data handling. The sign-off matches the version that will actually ship.

COMMON FAILURE MODES

- The team launches without the staged rollout. The first failure affects 100 percent of users.

- The rollback path exists in theory but has never been tested. The first time someone tries it under pressure, it does not work.

- Monitoring is live but the on-call rotation is not. Alerts fire to no one for the first 48 hours.

- The legal sign-off is for a version that no longer matches what shipped, because changes were made after sign-off without re-review.

WHAT GOOD LOOKS LIKE

- All seven pass criteria are documented as met before the customer-facing switch is flipped.

- The staged rollout starts at 1 to 5 percent of traffic and ramps on observed signal, not on a calendar.

- The first week of customer-facing operation is staffed for incident response, not staffed as normal operations.

The Stack We Recommend in 2026

For a Python-native engineering team building an enterprise LLM application, this is the stack we have shipped against most often. It covers the four-stage pipeline (local dev, CI, deployment gate, production monitoring) without major gaps.

- CI evaluation: DeepEval for pytest-style integration with the regression suite.

- RAG-specific evaluation: RAGAS for faithfulness, context recall, and answer relevancy. Plug into the same CI suite.

- Production traceability and online evaluation: Braintrust for production observability, dataset management, and continuous evaluation.

- Prompt-level evaluation: Promptfoo where the team is iterating on prompts as a separate cycle from code.

- Open-source judge: Prometheus where data residency or cost rules out frontier models for evaluation.

- Adversarial corpora and threat catalogs: OWASP LLM Top 10, NIST AI RMF, MITRE ATLAS for adversarial test design. This is not the only valid stack. It is the one we have shipped against most often. Substitute LangSmith, TruLens, or hyperscaler-native equivalents (Vertex AI Studio for evals, Azure AI Foundry for evals) where they fit your existing

platform decisions better.

A Note on Sequence Most teams build these five parts in a different order than we recommend. The common order is regression suite first (because CI feels productive), then golden set later (because the suite has to run against something), then evaluator model after launch (because human evaluation was the launch plan), then drift detection after the first production incident, then handoff criteria when the customer-facing launch is two weeks away.

That order is wrong. The right order is: golden set, regression suite, evaluator model, drift detection, handoff criteria. The golden set is the foundation. Everything else depends on it. Build it first, even when it feels less productive than wiring up tests.

How We Use This Setup

We run this setup, or a variant of it, on every enterprise LLM engagement at ViitorCloud Enterprise. The version we use internally includes the golden set template, the regression suite test scaffolding, the LLM-as-Judge rubrics for common use cases (RAG, support copilot, classification), the drift detection dashboards, and the customer-facing handoff checklist. We will send the working set on request as part of a 30-minute scoping call.

Book a 30-minute scoping call to review your LLM evaluation setup.

- References and Tools Cited

- DeepEval. Open-source pytest-style framework for LLM evaluation in CI.

- RAGAS. Open-source framework for evaluating RAG systems on faithfulness, context recall, and answer relevancy.

- Braintrust. Production observability, dataset management, and continuous evaluation platform.

- Promptfoo. Open-source framework for prompt-level evaluation.

- Prometheus (13B). Open-source evaluator LLM achieving 0.897 Pearson correlation with human evaluators.

- LangSmith and TruLens. Alternative platforms used in similar pipelines.

- OWASP LLM Top 10. Adversarial threat catalog for LLM systems.

- NIST AI Risk Management Framework and MITRE ATLAS. Frameworks used inside the adversarial test suite design.

ViitorCloud Enterprise

Deep expertise for enterprise programs that have to hold up in production.

enterprise.viitorcloud.com | enterprise@viitorcloud.com | (+91) 84889 64723