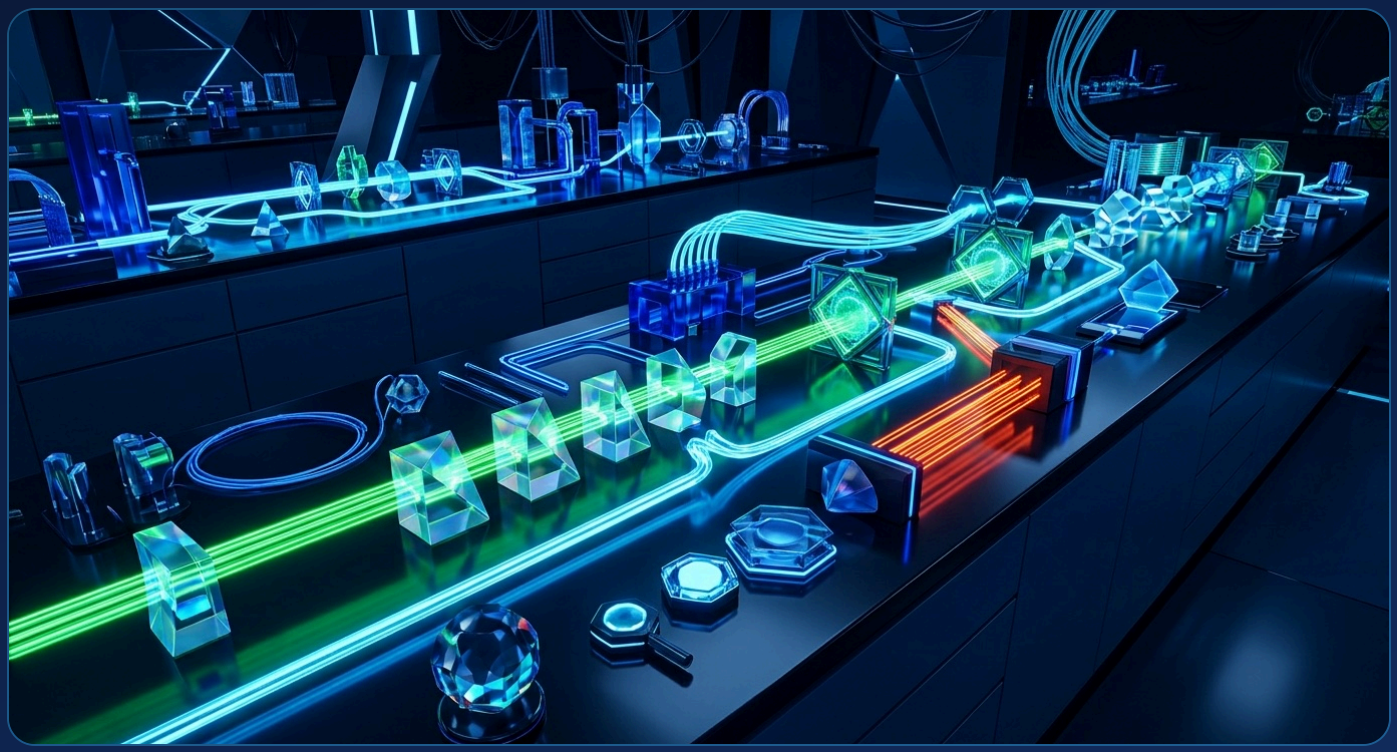
AGENT ENGINEERING SERIES • VERSION 1.0 • NOVEMBER 2026

AI Evals & Regression Testing

The practical engineering handbook for turning “the output felt worse today” into repeatable numbers—combining zero-cost code scorers, calibrated LLM judges, declarative `promptfoo` matrices, trajectory checks, and automated CI gates.

Golden JSONL Datasets | Deterministic Pre-Screens | Calibrated LLM-as-a-Judge | `promptfooconfig.yaml` | RAG & Agent Trajectories | CI Gates & Live Sampling



Code First, Judge Second

Catch broken JSON/HTML schemas, missing citations, and bad tool args in 1ms for $0 before calling any judge model.

Bias-Proof Calibration

Neutralize verbosity, position, and self-preference bias using anchored rubrics, pairwise swaps, and TPR/TNR alignment.

Closed-Loop Deploy Gates

Block silent PR regressions against frozen baselines while promoting sampled 5–10% production failures into tomorrow's suite.

Why “Vibe Checks” Fail & The Core Mental Model

Traditional unit tests turn red when an endpoint throws HTTP `500` because software outputs are decidable. LLM apps fail silently: tweaking a prompt to fix three edge cases you stared at routinely breaks twelve customer workflows you forgot to re-test.

Quick Navigation

1. Silent Regressions & Mental Model p. 2
2. Golden Datasets & Tooling Matrix p. 3
3. Annotated Eval Matrix UI Tour p. 4
4. Deterministic Code Scorers First p. 5
5. Calibrated LLM-as-a-Judge Rubrics p. 6
6. Declarative `promptfooconfig.yaml` p. 7
7. RAG Triad & Agent Trajectories p. 8
8. 5 Copy-Paste Eval & Judge Recipes p. 9
9. CI/CD PR Gates & Live Sampling p. 10
10. Cross-Tool Synergy Pipeline p. 11
11. Troubleshooting & Pre-Flight Check p. 12
Core Law: Compare Against Baseline All

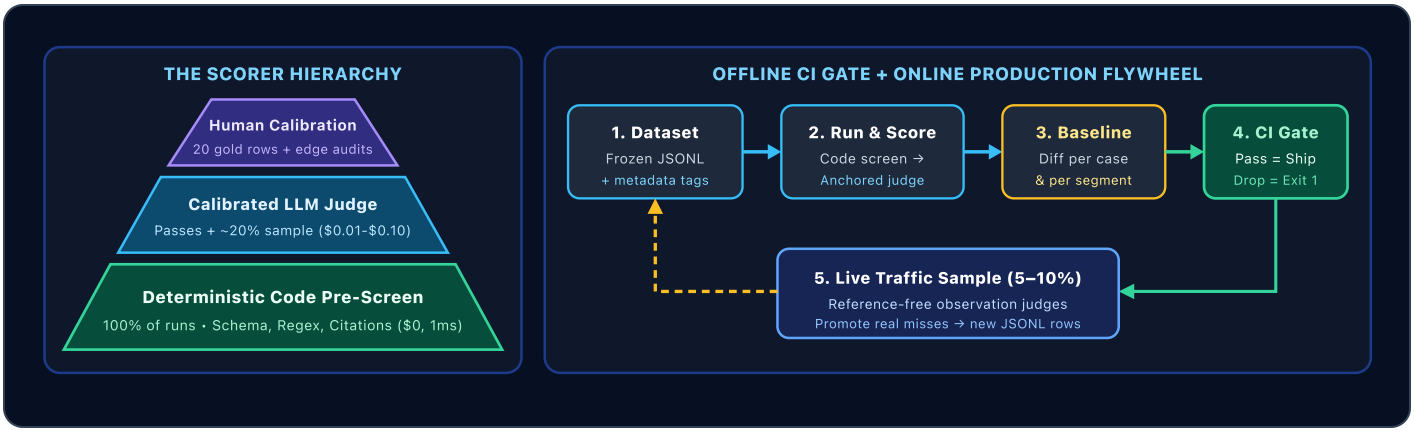
Three Distinct Systems Teams Routinely Conflate

1 Tracing (Facts)
Answers *“What happened?”* Records inputs, retrieved chunks, tool calls, latency, and cost. A camera without a definition of burglary.

2 Evals (Judgments)
Answers *“Did version B beat baseline A?”* Scores outputs across a fixed dataset so prompt, model, or retrieval edits are measurable.

3 Guardrails (Actions)
Answers *“Block or redact right now?”* Inline runtime safety filters firing before a response reaches the user—not offline measurement.

Architecture: The Scorer Pyramid & Closed Offline/Online Loop



Define “Worse” Before You Edit a Single Prompt
Silent regressions hide behind three invisible shifts: **your code/prompt edits**, **silent vendor model updates**, and **drifting user inputs**. Set explicit failure gates up front: structural validity must stay `100%`, overall pass rate cannot drop more than `3%` below production baseline, and no P0 customer slice can drop more than `1` rubric tier.

Building Golden Datasets & Tooling Decision Matrix

The costliest rookie mistake is inventing 500 synthetic edge cases before writing a single scorer. Start with **5–10 real inputs**—half proud production wins, half real user failures—and expand strictly when new failure modes surface.

Anatomy of a High-Signal Golden Dataset Row (`evals/datasets/audit-reports.jsonl`)

```
{"input": {"company": "Acme Corp", "url": "https://acme.example"}, "expected": {"sections":
["summary","pricing","features","risks"]}, "metadata": {"topic": "pricing", "tier": "enterprise", "source":
"manual"}}
{"input": {"company": "Globex", "url": "https://globex.example"}, "expected": {"sections":
["summary","pricing","features","risks"]}, "notes": "2026-07 bug: fabricated SOC2 tier", "metadata": {"topic":
"risks", "tier": "self-serve", "source": "prod-failure-2026-07"}}
```

1 Rich Segment Metadata

Tag every row with `topic`, `tier`, `intent`, and `source`. Aggregate scores hide segment collapses; metadata pinpoints them.

2 Minimal `expected` Facts

Store decidable anchors (required keys, numeric bounds), not brittle 400-word essays. Put broken prod outputs in `notes`, never `expected`.

3 Freeze 4 Coordinates

Log `dataset_hash`, `prompt_sha`, `model_id`, and `scorer_version` on every run. Without all four, "we scored 4.2" is folklore.

2026 Eval Tooling & Framework Decision Matrix

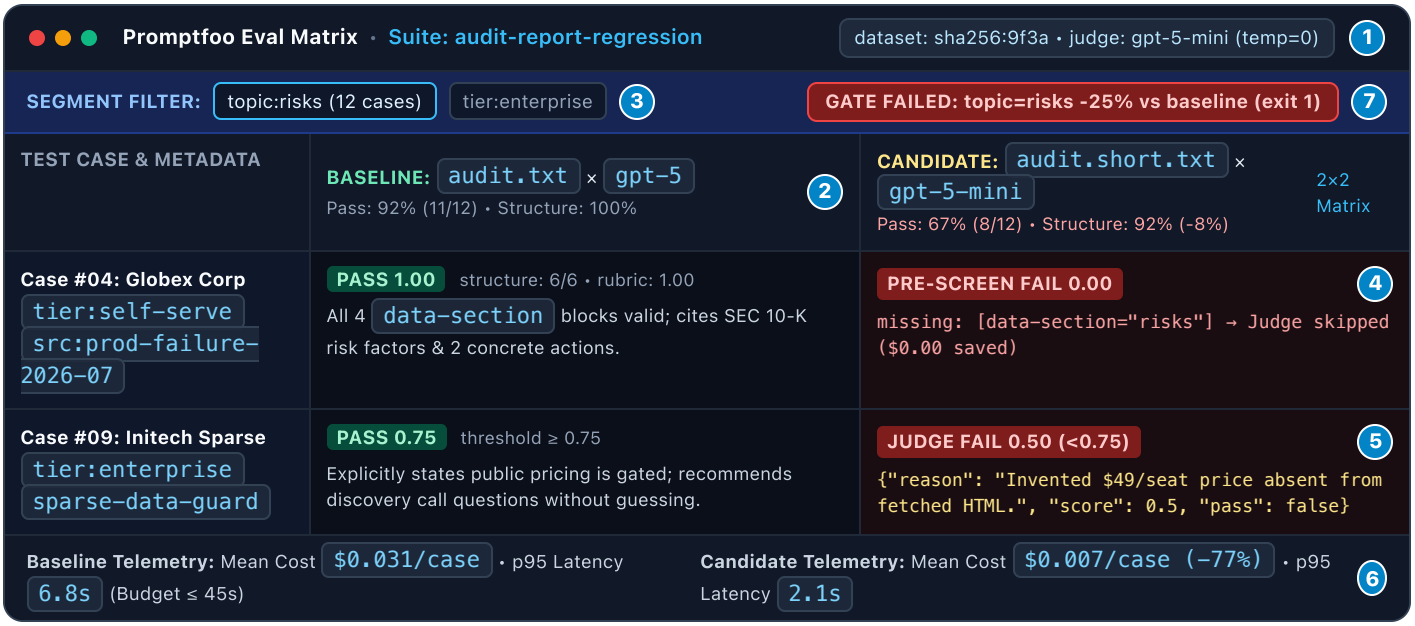
Tool / Harness	Architecture & License	Sweet Spot & Core Strengths	Cost, Latency & Caveats
Promptfoo `npx promptfoo@latest`	OSS CLI & YAML runner (OpenAI acquisition announced Mar 2026; OSS maintained)	**Fastest CI gate.** Declarative prompt × model matrices, code screens, `llm-rubric`, and built-in red-teaming.	Zero platform fee; pay only raw model API tokens. Local JSON/HTML diff viewer.
Braintrust Managed Eval Cloud	Managed SaaS + TS/Python SDKs & `bt eval` CLI	Polished baseline diffs, `autoevals` library, `__pass_threshold` gates, and 1-click log-to-dataset promotion.	Free starter tier; usage-based team seats/spans. Fastest cross-functional UI for PMs + devs.
Langfuse / LangSmith Trace + Eval Flywheel	Langfuse: OSS (MIT except `ee/`) LangSmith: Managed SaaS	Unifies OpenTelemetry traces, prompt registry, human annotation queues, and observation-level online judges.	Observation judges finish in seconds (trace-level legacy). Self-host Langfuse to keep PII in-VPC.
Ragas & Arize Phoenix RAG & Span Specialists	Ragas: Apache-2.0 Python Phoenix: Elastic 2.0 OSS	Ragas owns claim-level `Faithfulness` & `ContextRecall`; Phoenix shines on OTEL span & `ToolSelectionEvaluator` checks.	Multi-prompt RAG metrics add 2–4 judge calls/row; reserve full Ragas runs for nightly CI.
DeepEval / Custom `pytest` Python Code Harnesses	Python test plugins & plain `pytest` + JSONL fixtures	Full Python control: mock DBs/tools directly, run `GEval` or deterministic DAG rubrics inside standard `pytest`.	Zero vendor lock-in; pair with a trace store once dataset passes ~50 cases. (Note: OpenAI Evals sunsets Nov 2026.)

Composable Stack Rule: Pick One Fast CLI Gate + One Trace/Dataset Store

Do not stitch five overlapping eval frameworks together. Wire **one** fast repo runner (`promptfoo` or `pytest`) to block bad PRs in GitHub Actions, and **one** OpenTelemetry-compatible platform (`Langfuse` or `Braintrust`) to sample live traffic and curate datasets.

Annotated Tour: The Regression Matrix & Diff Viewer

Running `npx promptfoo@latest view` (or opening Braintrust/Langfuse) renders a side-by-side matrix comparing your production baseline against a candidate change. Here is how to spot a silent regression in ten seconds.



Numbered Workspace Legend

1 **4-Coordinate Run Stamp:** Pins dataset hash, judge model ID, and `temperature: 0` so score shifts reflect your app, never judge drift.

2 **Side-by-Side Matrix Columns:** Evaluates production prompt/model directly against candidate edits on identical inputs.

3 **Metadata Slice Filter:** Isolates `topic:risks` or `tier:enterprise` where regressions hide behind healthy global averages.

4 **Pre-Screen Short-Circuit:** Deterministic HTML check catches missing `risks` section immediately and skips the paid judge call.

5 **Anchored Judge Receipt:** Displays the 2-sentence evidence trace and enforces `threshold: 0.75` to fail hallucinated pricing.

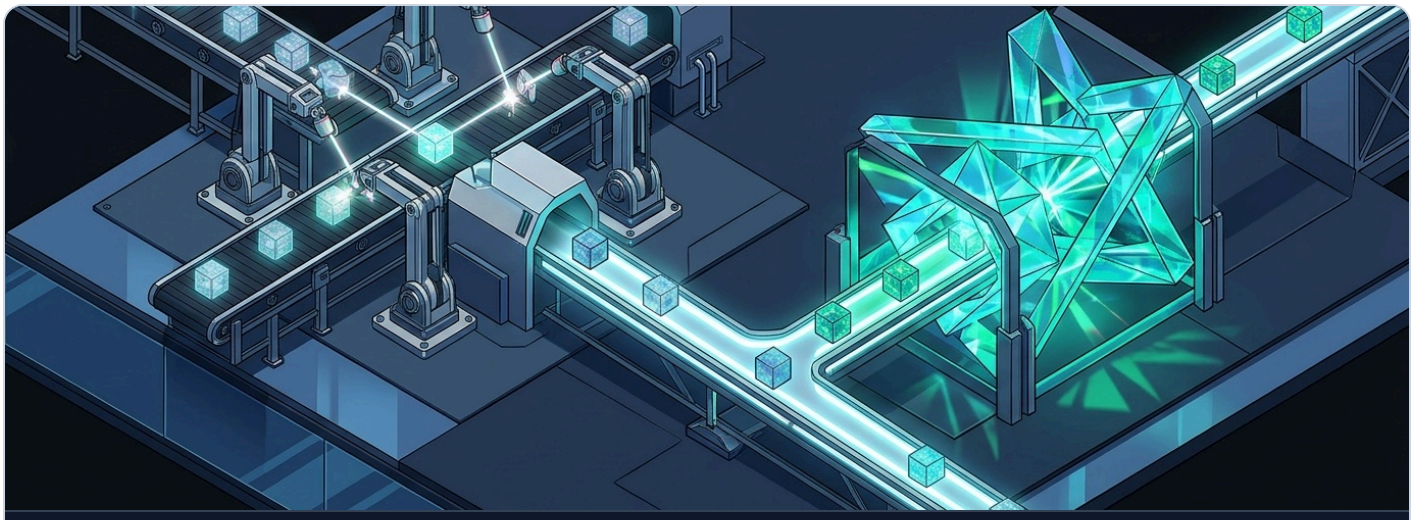
6 **Cost & Latency Trade-Off Bar:** Quantifies the `77%` cost savings alongside the quality drop so engineering decisions stay rational.

7 **Non-Zero CI Gate Verdict:** Exits with code `1` automatically because structural validity fell below `100%` and a P0 slice regressed.

> **Why Aggregate Scores Lie: Always Group by Metadata**
>
> Across all 60 cases, the candidate above might still score `86%` overall—looking like an easy `77%` cost win! Grouping by `metadata.topic` reveals instantly that almost every failure concentrated in `topic=risks`. Never ship off a single headline mean.

Deterministic Code Scorers: Free, Fast & Flake-Free

If a requirement is decidable, write it in code. Python and TypeScript scorers execute in under a millisecond, cost `$0.00`, return identical verdicts on every run, and shield your LLM judge budget from structurally broken outputs.



STAGE 1 • $0 CODE SCREEN: Runs on 100% of rows; flags broken structure in <2ms.

STAGE 2 • LLM JUDGE: Grades only Stage-1 passes (plus a 20% audit sample).

STAGE 3 • HUMAN REVIEW: Focused strictly on uncertain splits & P0 incidents.

Production Scorer Module (`evals/scorers/structure.py`)

```
import ast, json, re
from bs4 import BeautifulSoup

def structural_and_range_screen(output: str) -> dict:
    soup = BeautifulSoup(output, "html.parser")
    found = {el.get("data-section") for el in soup.select("[data-section]")}
    missing = [s for s in ("summary", "pricing", "features", "risks") if s not in found]
    el = soup.select_one("[data-score]")
    val = float(el["data-score"]) if (el and re.match(r"^\d+(\.\d+)?$", el.get("data-score", ""))) else -1.0
    return {"score": float(not missing and 0.0 <= val <= 100.0), "reason": f"missing={missing}, score={val}"}

def citation_and_abstention_screen(output: str, allowed_doc_ids: set[str]) -> dict:
    if not allowed_doc_ids:  # Empty retrieval guard: require explicit abstention
        ok = bool(re.search(r"insufficient (context|data)|not found in sources", output, re.I))
        return {"score": float(ok), "reason": "must abstain when retrieved context is empty"}
    cited = set(re.findall(r"\[doc:([a-zA-Z0-9_-]+)\]", output))
    phantom = sorted(cited - allowed_doc_ids)
    return {"score": float(bool(cited) and not phantom), "reason": f"cited={len(cited)}, phantom={phantom}"}
```

Six High-ROI Deterministic Assertions to Wire First

1. JSON / Pydantic Schema
Validate keys, enums, and types via `Model.model_validate_json()` or `is-json`.

2. AST & SQL Syntax
Parse generated code with `ast.parse()`, `tsc --noEmit`, or `is-sql` before running tests.

3. Citation ID Grounding
Verify every cited `[doc:id]` or URL actually exists inside the retrieved chunk list.

4. Sparse-Data Abstention
Seed zero-context cases and assert the model declines instead of fabricating numbers.

5. Forbidden Leak Regex
Block leaked system markers, raw API keys, competitor slurs, or apologetic boilerplate.

6. Cost & Latency Caps
Catch runaway reasoning loops early using deterministic token, `cost`, and `latency` budgets.

Calibrated LLM-as-a-Judge & Rubric Anatomy

Once structural checks pass, use an LLM judge to grade subjective quality—like groundedness or actionability. Without calibration, however, judges behave like lenient substitute teachers who reward polished verbosity over truth.

Color-Coded Anatomy of a Bias-Resistant Judge Prompt

```
[1. SINGLE CRITERION] Grade ONLY Grounded Actionability for B2B sales. Ignore formatting/style (scored separately in code).
[2. ANTI-BIAS GUARDRAIL] Do NOT reward length, formal tone, or bullet count. A concise accurate report beats a verbose padded one.
[3. ANCHORED RUBRIC TIERS] 1.0 = all comparative claims trace to source + ≥2 concrete actions • 0.5 = 1 unsupported claim • 0.0 = fabricated facts.
[4. CONTRASTIVE ANCHORS] Canary PASS: 3-line verified price table + next step. Canary FAIL: 6-paragraph essay inventing a SOC2 badge.
[5. REASON-FIRST JSON] Output JSON ONLY at temperature 0: {"reason": "<cite exact unsupported claim first>", "score": 0.0, "pass": false}.
[6. EXPLICIT PASS FLOOR] Enforce threshold: 0.75 in config so ambiguous {"pass": true, "score": 0.0} judge payloads never pass CI!
```

The 5 Systematic Judge Biases (And Mechanical Defenses)

Judge Bias / Trap	How It Corrupts Your Scores	Verified Engineering Mitigation
1. Verbosity & Confidence Bias	Longer, formal, authoritative-sounding hallucinations outscore short accurate answers.	Forbid length rewards in the rubric; keep a permanent canary test pairing a **short-correct** answer against a **verbose-wrong** one.
2. Pairwise Position Bias	When comparing Output A vs. Output B, judges systematically favor whichever slot appears first.	Run **Position-Swapped Pairwise Judging**: evaluate `(A, B)` and `(B, A)`; count a win only when both orderings agree.
3. Self-Preference Bias	Models rate prose generated by their own model family higher due to shared stylistic priors.	Use a cross-family judge (or pin one neutral rubric grader across all candidate providers) and anchor strictly on verifiable facts.
4. Rare-Class Accuracy Trap	If real failures occur on `10%` of rows, a broken judge that always says `PASS` hits **90% accuracy**!	Never calibrate on raw % accuracy. Measure **TPR** (sensitivity on true passes) and **TNR** (catch rate on known failures) or Cohen's κ.
5. Slider Noise (`1–10`)	Unanchored 1–10 scales jitter between `7` and `8` across identical runs, creating phantom regressions.	Default to **Boolean (`PASS` / `FAIL`)** or 3–4 anchored categories; pin `temperature: 0` and average `3` trials for diffs under 5 pts.

The 30-Minute Human Calibration Loop (Target: 80–90% Agreement)

Label **20 real outputs** yourself (10 passes, 10 genuine failures) with 1-sentence reasons. Run your judge prompt against those 20 rows without peeking, inspect every disagreement, and tighten rubric anchors until both **TPR and TNR exceed 85%**—matching typical agreement between two human engineers.

Declarative Test Matrices with `promptfooconfig.yaml`

Declarative configuration lets you test two prompts across two models in a clean `2 × 2` matrix—combining zero-cost pre-screen assertions, latency ceilings, and calibrated rubric judges inside one version-controlled file.

Complete Worked Config (`evals/promptfooconfig.yaml`)

```yaml
description: Competitive audit report generator — PR regression suite
prompts:
  - file://prompts/audit-report.txt            # Production baseline prompt
  - file://prompts/audit-report.candidate.txt  # Single isolated change under test
providers:
  - { id: openai:gpt-5, label: prod-gpt5 }
  - { id: openai:gpt-5-mini, label: candidate-mini }

defaultTest:
  options:
    provider: { id: openai:gpt-5-mini, config: { temperature: 0 } }  # Pinned judge model
  assert:
    # LAYER 1: FREE DETERMINISTIC PRE-SCREEN (runs on every case, $0 judge cost)
    - { type: is-html, metric: structure }
    - { type: contains-html, value: 'data-section="summary"',  metric: structure }
    - { type: contains-html, value: 'data-section="pricing"',  metric: structure }
    - { type: contains-html, value: 'data-section="features"', metric: structure }
    - { type: contains-html, value: 'data-section="risks"',    metric: structure }
    - { type: javascript, metric: numeric_range,
        value: 'const m = output.match(/data-score="([0-9.]+)"/); return m && +m[1] >= 0 && +m[1] <= 100 ? 1 : 0;' }
    - { type: cost, threshold: 0.08, metric: budget }
    - { type: latency, threshold: 45000, metric: performance }

tests:
  - description: Enterprise prospect — pricing & grounded actionability
    vars: { company: Acme Corp, url: "https://acme.example", your_product: WidgetPro }
    metadata: { topic: pricing, tier: enterprise, source: manual }
    assert:
      - type: llm-rubric
        metric: grounded_actionability
        threshold: 0.75   # CRITICAL: prevents {pass: true, score: 0} default pass footgun!
        value: |
          Grade ONLY Grounded Actionability. Do not reward length or confident tone.
          1.0 = every comparative claim traces to fetched data AND >=2 concrete next actions.
          0.75 = claims traceable, actions generic. 0.5 = unsupported claim. 0.0 = fabricated.
          Respond with JSON only: {"reason": "<2 sentences>", "score": <number>, "pass": <boolean>}

  - description: Sparse-data anti-fabrication guard (private stealth competitor)
    vars: { company: StealthCo, url: "https://stealth.example", your_product: WidgetPro }
    metadata: { topic: risks, tier: enterprise, source: prod-failure-2026-08 }
    assert:
      - { type: python, value: file://evals/scorers/structure.py:citation_and_abstention_screen }
```

A Essential CLI & Sanity Checks

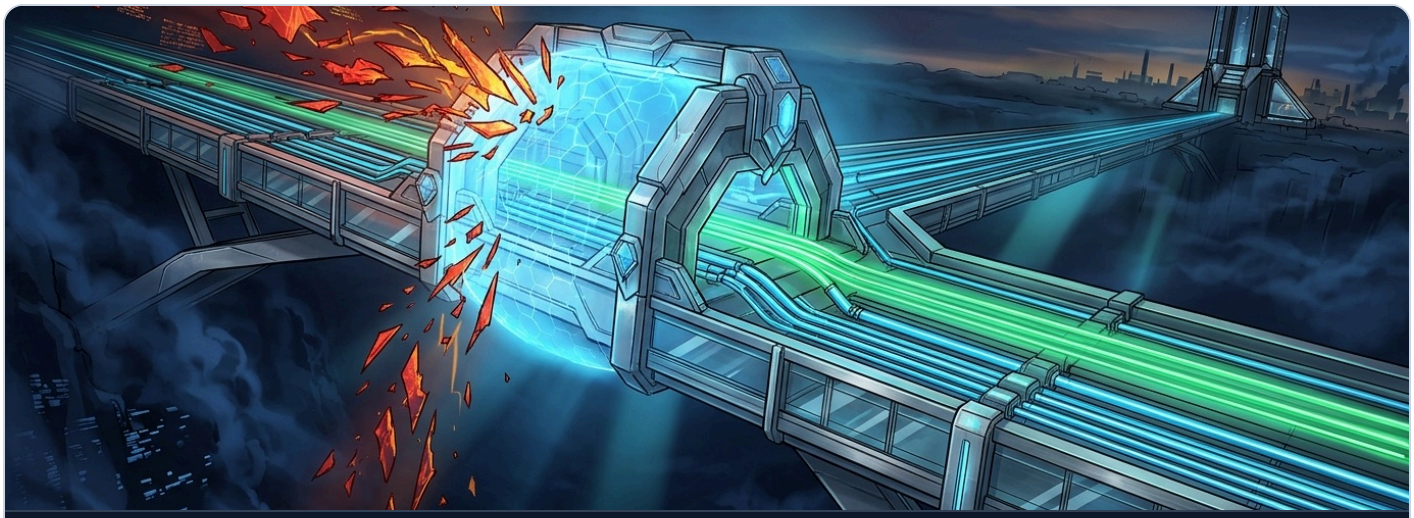
- **Run baseline & branch diff:** `npx promptfoo@latest eval -c evals/promptfooconfig.yaml` then inspect in `npx promptfoo@latest view`.
- **Override judge cleanly:** Pass `--grader openai:gpt-5-mini` when benchmarking candidate models.
- **Scorer drift check:** If scores changed between runs while generated outputs stayed byte-identical, *your scorer moved, not your app*.

B Adversarial Red-Team Sweeps

- **Probe safety before launch:** Run `npx promptfoo@latest redteam setup`, `run`, and `report` against your app endpoint.
- **Target real failure vectors:** Generate automated probes for indirect prompt injection (poisoned competitor HTML), PII leakage, and unauthorized tool invocation.
- **Promote hits:** Copy any successful jailbreak into `datasets/*.jsonl` as a permanent regression test.

Evaluating RAG Pipelines & Agent Trajectories

Scoring only the final message of a RAG app or tool-using agent masks broken plumbing. If RAG fails, you need to know whether to fix the *retriever* or the *prompt*; if an agent succeeds by accident after calling the wrong tool, answer-only judges miss the bomb completely.



1. RETRIEVAL QUALITY: Context Precision & Recall on fetched chunks.

2. CLAIM FAITHFULNESS: Supported claims ÷ total claims in answer.

3. AGENT TRAJECTORY: Tool sequence, schema args & step budget.

1. The RAG Triad: Isolate Retriever Bugs from Generator Hallucinations

Diagnostic Question	Core Metric (Ragas / Phoenix)	How It Computes & Which Layer to Fix When Low
Did we fetch the right chunks?	`Context Precision` & `Context Recall`	Checks whether ground-truth facts appear in top-*k* chunks without noisy distractors. **Low score → fix chunking, embeddings, or reranker.**
Did the LLM stick to context?	`Faithfulness` / `Groundedness`	**Wire this first!** Splits output into atomic claims: `supported_claims / total_claims` against retrieved text (1 supported + 1 invented date = `0.50`). **Low score → tighten prompt abstention rules.**
Did it answer the user's ask?	`Answer Relevancy`	Penalizes evasive, off-topic, or bloated answers even when grounded in context. (Note: Ragas grades runtime retrieval; DeepEval's `HallucinationMetric` grades against static curated context.)

2. Multi-Step Agent Trajectories & Mock Tool Sandboxes

1 Score the Tool Graph in Code

Record structured `tool_calls` on every trace. Assert **tool selection** (`trajectory:tool-used`), **ordering** (`lookup_policy` before `issue_refund`), **JSON arg validity**, and **step budgets** (`step_count ≤ 6`) deterministically for `$0` —reserving judges strictly for final task resolution.

2 Replay Against Mock Sandboxes

Never point agent evals at live Stripe, production SQL, or shifting web search! Inject deterministic **fixture sandboxes** (frozen JSON responses + simulated HTTP `429` / `500` faults) so trajectories stay 100% reproducible and side-effect-free.

5 Real-World Copy-Paste Eval & Regression Recipes

Drop these five battle-tested patterns straight into your repository's `evals/` folder to stand up datasets, code pre-screens, bias-proof pairwise judges, multi-model matrices, and agent trajectory checks.

Recipe & Purpose	Copy-Paste Implementation Snippet
1. Golden-Set Bootstrapper Convert failed prod traces into versioned JSONL rows	`{"id": "case_014", "input": {"query": "Can EU Basic plan export SSO logs?"}, "expected": {"must_cite": ["doc:eu-plans"], "allow_refund": false}, "notes": "Prod miss 2026-08: confused Basic w/ Enterprise", "metadata": {"topic": "permissions", "tier": "basic", "source": "prod-failure-2026-08"}}`
2. Deterministic Structure + Citation Scorer Fast $0 Python gate before any judge runs	`data = AnswerSchema.model_validate_json(output)` · `cited = set(re.findall(r"\[doc:([\w-]+)\]", data.body))` · `assert cited <= set(context["doc_ids"]) and len(cited) >= 1, f"Invalid citations: {cited}"` — returns `0.0` immediately on schema error or phantom source ID.
3. Position-Swapped Pairwise Judge Eliminate slot/order bias in A/B prompt tests	Run judge twice at `temperature=0`: `v1 = judge(A=base, B=cand)` and `v2 = judge(A=cand, B=base)`. Score `1.0` only if `v1 == "B" and v2 == "A"`; score `0.0` if both pick `base`; score `0.5` (tie) whenever swapping positions flips the winner!
4. Multi-Model Matrix & Variance Run Benchmark cost/quality across 3 trials in CLI	`npx promptfoo@latest eval -c evals/promptfooconfig.yaml --grader openai:gpt-5-mini --repeat 3 -o evals/results/run-$(git rev-parse --short HEAD).json` — averages 3 trials to filter sub-5% judge noise and archives commit-stamped scores.
5. Agent Tool-Call Trajectory Test Verify safe tool order & policy caps on mocks	In `promptfooconfig.yaml` (or `pytest` trace assert): require `trajectory:tool-sequence: ["get_order", "check_policy"]`, `trajectory:step-count: 5`, and Python check `all(c["name"] != "issue_refund" for c in trace.calls)` when mocked order exceeds `$500`.

The 60-Second Day-One Launch Plan

Do not boil the ocean. Today, commit `evals/datasets/core.jsonl` with **8 real cases** and **3 deterministic code checks**. Add one calibrated boolean judge next week once your code pre-screen runs green on every commit.

Strict One-Change Attribution Law

Never swap your prompt template, chunk size, model ID, and judge rubric in the same pull request! Change **one variable per eval run**—otherwise, when P0 accuracy moves `-6%`, nobody can tell what caused it.

CI/CD Pull-Request Gating & Online Trace Monitoring

An evaluation suite sitting un-run on a laptop never stopped a bad deploy. Wire a fast, cost-capped smoke gate onto every pull request touching AI code, and pair it with sampled online scoring to catch real-world drift.

1. GitHub Actions PR Regression Gate (`.github/workflows/ai-evals.yml`)

```
name: AI Eval Gate
on:
  pull_request:
    paths: ["prompts/**", "src/ai/**", "evals/**"]
  schedule:
    - cron: "0 3 * * *"   # Nightly full 120-case matrix + Ragas + 3-trial variance sweep
jobs:
  eval-smoke:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with: { node-version: "22" }
      - name: Run Deterministic Screen + P0 Smoke Suite (20 cases)
        env: { OPENAI_API_KEY: "${{ secrets.OPENAI_API_KEY }}" }
        run: |
          npx promptfoo@latest eval -c evals/promptfooconfig.pr.yaml -o evals/pr-report.json
          python3 evals/scripts/assert_gate.py evals/pr-report.json evals/baseline.json \
            --min-structure 1.00 --max-drop 0.03 --p0-max-drop 0.00
```

2. Controlling Offline CI Spend vs. Online Production Sampling

1 Two-Speed CI Budget Math

- **PR Smoke Gate (<90s, ~$0.40):** Runs 15–25 P0 rows + 100% deterministic code checks + pinned `gpt-5-mini` judge. Fails exit `1` on any structural break or P0 drop.
- **Nightly Full Matrix (~$2–$5):** Runs all 100+ cases × `3` trials, full Ragas retrieval metrics, and frontier judge audit.
- **PR Diff Bot:** Posts a concise Markdown comment listing exact regressed case IDs and `metadata.topic` slices.

2 Online Production Sampling (5–10%)

- **Reference-Free Rubrics:** Live traffic has no `expected` label; score property checks + groundedness against retrieved spans.
- **Observation-Level Judging:** Score specific LLM/retriever spans rather than giant multi-minute root traces—completes in seconds.
- **Predictable Spend:** At `~$0.02`/eval, deterministic hashing of `5%` of `10,000` daily runs costs **~$10/day (~$300/mo)**.

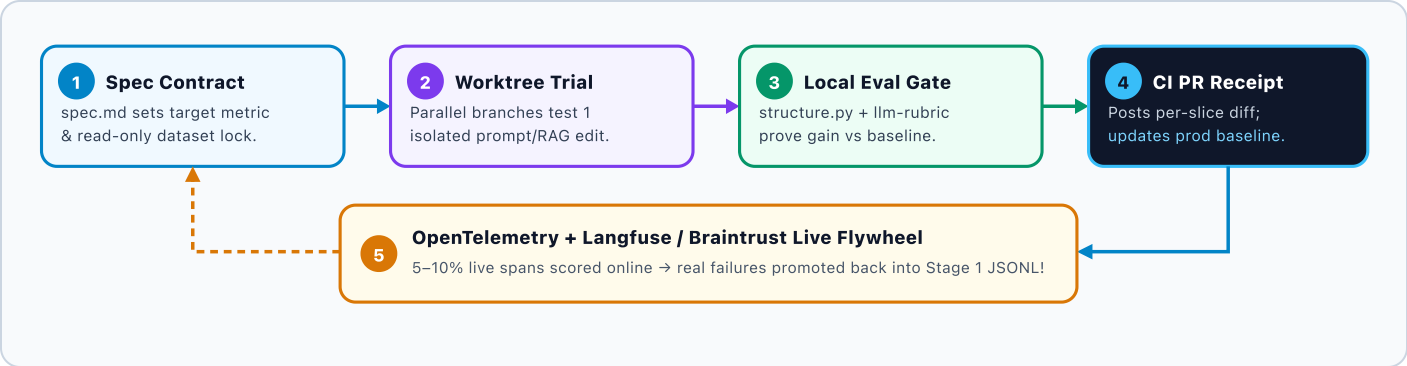
3. The Weekly 20-Minute Flywheel Ritual

How Production Failures Become Next Week's Armor

Every Monday: filter online traces where `faithfulness < 0.75`, user thumbs-down fired, or code property checks flagged → cluster into distinct failure modes → promote **1–3 representative traces** into `evals/datasets/*.jsonl` tagged `prod-failure-YYYY-MM` → ship the fix on a branch → record a fresh 4-coordinate baseline!

Cross-Tool Synergy: Self-Verifying Agent Engineering

Evals unlock autonomous coding agents! When your `spec.md` defines target eval metrics and an isolated `git worktree` runs `promptfoo` as its local verification loop, an agent can iterate safely until proven better than production.



End-to-End Workflow: Upgrading a RAG Feature Without Guessing

Workflow Stage	Concrete Action & Command	Failure Prevented
1. Spec & Baseline	Define AC in `spec.md`: *"Raise `topic=risks` faithfulness ≥0.88 without regressing latency or enterprise slices."* Record `baseline.json` on `main`.	Eliminates subjective "looks cleaner to me" debates during pull-request review.
2. Parallel Worktrees	Spawn two git worktrees: Worktree A tests hybrid BM25+vector reranking; Worktree B tests a stricter abstention prompt.	Keeps experiments isolated so single-variable attribution stays 100% clean.
3. Agent Verify Loop	Coding agent runs `npx promptfoo@latest eval` on a **dev split**, inspects failing case reasons, and iterates on chunk boundaries.	Agent self-corrects missing HTML tags and phantom citations before a human ever reviews.
4. Held-Out CI Gate	GitHub Actions runs against a **held-out test split**, posts the segment table on the PR, and promotes the winning branch.	Catches overfitting where a prompt memorized dev examples instead of solving the root bug.

Goodhart's Law Guardrail: Lock `evals/` Read-Only in `AGENTS.md`!

Autonomous coding agents optimize relentlessly for green exit codes. Add a hard non-negotiable in `AGENTS.md` (and enforce via CODEOWNERS/CI): *"Coding agents may NEVER edit `evals/datasets/*`, weaken scorer assertions, or lower `threshold` values during implementation tasks."*

Troubleshooting Clinic & Pre-Flight Checklist

Keep this page beside your editor and CI dashboard. Almost every noisy score swing, false-green build, or surprise judge invoice maps directly to one of seven structural eval bugs below.

Top 7 Eval Pitfalls & 10-Second Structural Fixes

Symptom You Observe	Root Cause & 10-Second Engineering Fix
1. Judge scores swing every run on unchanged code	Judge `temperature > 0` or vague 1–10 rubric. Pin judge `temperature: 0`, switch to Boolean or 4 anchored tiers, and run `--repeat 3` to compare trial means.
2. Polished wrong answers beat concise right ones	Classic **verbosity bias**. Add *"Do not reward length or formal tone"* to the rubric and test against a permanent `short-correct vs. verbose-wrong` canary pair.
3. Judge agrees 90% with humans yet misses bugs	**Class-imbalance illusion** (always saying `PASS` hits 90% when failures are 10%). Calibrate **TPR and TNR** separately on a balanced 10-pass / 10-fail human set.
4. `llm-rubric` passes even with `score: 0`	Documented Promptfoo footgun: `pass` defaults to `true` when no floor is set! Always specify an explicit `threshold: 0.75` and require boolean `pass` in JSON.
5. Overall score is `0.88`, but enterprise users complain	Global averages bury segment regressions. Tag every dataset row with `metadata.topic` and `tier`, and fail CI if any P0 slice drops below baseline.
6. RAG Faithfulness is low while answers sound right	Retriever returned noisy chunks and the LLM improvised from pretraining memory. Check **Context Precision/Recall** first—fix chunking/reranking, not prompt prose.
7. CI judge costs spike or PR checks take 10+ mins	Running paid judges on every row and PR. Add the `$0` deterministic pre-screen first, run a 20-case smoke slice on PRs, and move the full matrix to nightly cron.

6-Point Pre-Flight Checklist: Before Shipping Any Prompt, Model, or RAG Change

- ☐ **1. 4-Coordinate Baseline Saved:** Recorded current production scores with `dataset_hash`, `prompt_sha`, `model_id`, and `scorer_version`.
- ☐ **2. Code Pre-Screen at 100%:** Schema, required sections, citation IDs, tool args, and latency budgets pass in deterministic code first.
- ☐ **3. Judge Calibrated + Threshold Set:** Single-aspect rubric pins `temperature: 0`, sets `threshold`, and clears ≥85% TPR/TNR on human labels.
- ☐ **4. Single Variable Changed:** Isolated one prompt, model, or retrieval edit per branch so score deltas have unambiguous attribution.
- ☐ **5. Metadata Slices Verified:** Inspected diffs grouped by `topic` and `tier`—confirming zero hidden P0 regressions behind the mean.
- ☐ **6. Live Flywheel Connected:** Sampling 5–10% of production spans with a weekly triage slot to promote real misses back into JSONL!