Macrodeep Research

The GTM-Bench Harness — Architecture of a Reproducible Evaluation Runner

Evaluation · Jul 2026 · 16 min read

Macrodeep Research Team

Abstract

An engineering companion to the GTM-Bench research series. The design note and its follow-ups describe what the benchmark measures and why the numbers can be trusted. This piece is about the thing that produces the numbers: the harness. How it's structured, how a model plugs into it, and how someone outside runs it against their own model and gets the same answer we do.

Contributions

- A four-stage pipeline — instances -> adapter -> grader -> aggregator -> report — with each stage single-purpose
- A tiny, model-specific adapter is the only seam between the harness and any model
- Every grader is a pure function from response and oracle to verdict — deterministic and independently auditable
- The whole runner is containerized; a run is fully specified by container image, instance-set version, and adapter

The claim this post has to back up

The research series makes one promise repeatedly: GTM-Bench is reproducible — an external party can run the harness against their own model and verify the score. That promise is only as good as the runner behind it. A benchmark whose harness is a tangle of internal scripts, live API calls, and undocumented assumptions is not reproducible no matter how clean the task design looks on paper.

So this post treats the harness as the product it is. The design goal is blunt: a stranger with a model endpoint should be able to run GTM-Bench and get a number they can defend, without talking to us. Everything below serves that.

The shape of a run

A GTM-Bench run is a pipeline with four stages, each with a single responsibility:

1. Load the instance set (a versioned collection of JSON task instances). 2. Present each instance to the model through an adapter and collect the raw response or, for agentic tasks, the tool trajectory. 3. Grade

each response with a category-specific grader against the instance's hidden oracle, producing a per-instance verdict. 4. Aggregate verdicts into category scores, the composite, and the objective-only number, and emit a report.

The stages are deliberately decoupled. The thing that talks to a model knows nothing about grading; the thing that grades knows nothing about how the response was obtained. That separation is what lets the same graders score any model, and lets a new model be added without touching grading logic.

The instance: the unit everything moves

Everything in the pipeline moves one object: the instance. Its contract is fixed across categories so the pipeline stays uniform:

Two properties of this contract matter for reproducibility. The `oracle` never leaves the harness — the adapter strips it before anything is sent to a model, so a model physically cannot see the answer even if it wanted to. And the `version` travels with every instance and every verdict, so a report can never claim a score without stating exactly which set produced it.

The adapter: the only model-specific code

The adapter is the seam between the harness and a model. It is the only part of the system that knows anything model-specific, and it is deliberately tiny. Its entire job is to satisfy one interface:

For single-shot categories (ENRICH, QUALIFY, TAM, ROUTE, COMPOSE, SUMMARIZE) `response` is the model's output. For ORCHESTRATE the adapter also brokers the tool loop: it hands the model the sandbox tool schema, relays each tool call into the sandbox, returns the sandbox's result to the model, and records the full trajectory. When the model finishes or hits the step budget, the adapter returns the trajectory and final sandbox state.

Keeping all model-specific behavior inside this one thin interface is the key architectural decision. Adding a model is writing one adapter, not modifying the harness — a hosted API, a local endpoint, an open-weights model behind an inference server, each is just an adapter satisfying `run()`. Graders never change when a new model is added, because they only ever see the standardized `response`, never the model. An external party integrates their model by writing exactly one small file and touching nothing else — that is what “reproducible by someone outside” reduces to in practice.

A reference adapter ships with the harness so the interface is unambiguous, and the tool-brokering loop for ORCHESTRATE is provided rather than left as an exercise, since that loop is where an incorrect integration would most easily distort a score.

The graders: pure functions from response to verdict

Each category has a grader, and every grader is a pure function:

No grader makes a network call, reads external state, or depends on anything but its two arguments. This is what makes grading deterministic and independently auditable — anyone can read a grader, feed it a response and an oracle, and confirm the verdict by hand.

The graders divide along the line the research series draws. Objective graders (ENRICH, QUALIFY, TAM, ROUTE) are deterministic comparisons: field-exact-match with the spec's normalization rules, label match, precision/recall at k, branch match. Same inputs, same verdict, forever. The ORCHESTRATE grader evaluates the end-state assertion language (`assert` and `forbid` clauses) against the final sandbox state, and separately computes the trajectory-efficiency diagnostic from the recorded call log — still a pure function, since the sandbox state is an input to it, not a live system it reaches into.

Generative graders (COMPOSE, SUMMARIZE) invoke a judge model under a fixed rubric. These are the one place a grader is not fully deterministic, which is precisely why the research series holds them to a minority weight and excludes them from the objective-only score. The harness surfaces them separately in the report so no one has to trust them to trust the objective number.

The sandbox: determinism by construction

The ORCHESTRATE sandbox — covered in depth in its own post — is, from the harness's point of view, a fixture-backed state machine with no external dependencies. It ships inside the harness container, seeds itself from each instance's initial state, applies tool calls as pure state transitions, and exposes final state for grading.

Because it never touches a real system, two things follow that the harness relies on: a run has no side effects outside its container, and the same trajectory reproduces the same final state on any machine. Reproducibility and safety fall out of the same design choice.

Containerization and reproducibility

Following the SWE-bench discipline, the whole runner is containerized. The container pins the harness code, the graders, the sandbox, and — critically — the instance-set version. A run is therefore fully specified by three things: the container image, the instance-set version, and the adapter for the model under test. Given those, the run is reproducible on any machine.

“We got score X” is not a claim anyone should accept on faith. “Here is the container, here is the instance-set version, here is the adapter interface — run it yourself” is a claim anyone can check.

Determinism is enforced, not hoped for: graders are pure, the sandbox is fixture-backed, instance sets are version-pinned, and the only non-deterministic surface — the generative judge — is quarantined into separately-reported categories. Re-running an objective score reproduces it exactly; if it doesn't, that is a bug in the harness, not benchmark noise.

The report: what a run emits

A run produces a structured report, not just a headline:

(Score fields shown as `null` — this post documents the report structure, not results.)

The report always carries the instance-set version alongside the scores, breaks the composite into its objective-only and per-category parts, includes the ORCHESTRATE efficiency diagnostic as a first-class field, and retains per-instance verdicts so any aggregate can be traced back to the instances that produced it. A number you cannot decompose is a number you cannot trust; the report is built so every aggregate is auditable down to the instance.

What ships, and what that enables

The runnable artifact is: the containerized harness, the category graders, the ORCHESTRATE sandbox, a reference adapter, and a versioned public instance set. With those, an external engineer can integrate a model by writing one adapter, run the public set, read the graders to see exactly how they were scored, and reproduce the objective numbers bit-for-bit.

The private held-out split — the one that produces authoritative leaderboard numbers — stays unpublished for the contamination reasons covered elsewhere in the series, but the mechanism that grades it is fully open. That split is intentional and, we think, the right one: open the machinery, hold out the answer key. It lets anyone verify how GTM-Bench grades while preserving a clean set to rank against.

Why an engineering post, not just a research one

The research series argues that GTM-Bench is trustworthy. This post is the part where that argument has to survive contact with an actual engineer. A benchmark's credibility ultimately rests on someone outside being able to run it — and being able to read the runner and agree it does what it claims. Publishing the architecture, the interfaces, and the reproducibility guarantees is how a benchmark earns the word “reproducible” instead of just using it.

The leaderboard remains a separate release, run under this harness, published only when the numbers are real and someone outside can reproduce them.

References

1. Jimenez et al. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? 2024.

2. Anthropic. Model Context Protocol (MCP) specification. 2024.

3. Liang et al. Holistic Evaluation of Language Models (HELM). 2023.