Macrodeep Research

Inside ORCHESTRATE — Grading Agentic Tool-Use in GTM-Bench

Evaluation · Jul 2026 · 14 min read

Macrodeep Research Team

Abstract

Second in a series on GTM-Bench. The design note laid out the seven-category taxonomy; this piece goes deep on the flagship category — how an agent's multi-step tool use is scored against a deterministic end state, and the failure modes the grader is built to catch. No model scores here; this is methodology.

Contributions

- Why ORCHESTRATE is the category that separates an agent from a text generator
- A deterministic mock-MCP sandbox: fixed initial state, pure-function tools, recorded trajectory
- An end-state assertion language with positive `assert` and negative `forbid` clauses
- Five failure modes execution grading catches that prose grading waves through

Why ORCHESTRATE carries the benchmark

The other GTM-Bench categories test knowledge and judgment: can a model resolve an entity, qualify an account, expand a market, pick the right next action. Those are real capabilities, but they are single-shot — the model produces an answer and the answer is checked.

ORCHESTRATE is different in kind. It tests whether a model can act — take a goal stated in plain language, decompose it, call the right tools in the right order with the right arguments, react to what the tools return, and leave the world in the correct final state. This is the capability that separates an agent from a text generator, and it is the one that matters most for real GTM work, where the job is not “describe what should happen to this lead” but “make it happen across the CRM and the sequencer.”

It is also the hardest category to fake. A model can pattern-match its way to a plausible-sounding cold email or a reasonable-looking qualification. It cannot pattern-match its way to a correct end state in a sandbox that only rewards the state actually reached. Either the record is in the right stage and the contact is in the right sequence, or it isn't. That binary is the whole point — it is GTM-Bench's equivalent of SWE-bench's “did the test suite go green.”

The environment: a deterministic mock-MCP sandbox

ORCHESTRATE runs inside a controlled environment that presents tools over the same interface a production agent sees — a tool schema, typed arguments, structured returns — but every tool is backed by an in-memory fixture rather than a live system. Nothing touches a real CRM, nothing depends on network conditions, and every call is recorded.

The design constraint driving everything here is determinism. A benchmark is only trustworthy if re-running a submission reproduces its score. That rules out live integrations, which fail, rate-limit, and mutate underneath you. Instead, each instance ships with a fixed initial world state, and the sandbox evolves that state purely as a function of the model's tool calls. Run the same trajectory twice, get the same final state twice, get the same score twice.

A representative slice of the tool surface exposed to the model:

Each tool is a pure function over sandbox state: `crm.update_record` mutates the in-memory record and returns it; `sequencer.enroll` appends to an enrollment list; `data.lookup` reads from a fixed fixture. Because the fixtures are frozen per instance, the tools are side-effect-free with respect to anything outside the container — which is exactly what lets an external party run the harness safely on their own machine against their own model.

The trajectory: what the harness records

Every tool call the model makes is logged in order, with its arguments and its return. A completed ORCHESTRATE run produces a trajectory like this (abridged):

The trajectory is not just an audit log — it is what makes two different grading philosophies both possible from a single run. The harness can grade the `final_state` (did the model get there) and, separately, inspect the `trajectory` (how efficiently, and whether it did anything it shouldn't have).

The oracle: an end-state assertion language

Grading does not look at the model's prose. It looks at the world the model left behind. Each ORCHESTRATE instance carries an oracle expressed as a list of assertions against final sandbox state:

Two clauses do the work. `assert` lists the conditions that must hold for the instance to pass — the positive end state. `forbid` lists states that must not hold — the negative constraints. The `forbid` clause is what stops a model from brute-forcing a pass by taking every action it can think of: enrolling the contact in three sequences to make sure one is right will trip a forbid check. Correct means reaching the required state and only the required state.

An instance passes ORCHESTRATE only if every `assert` holds and no `forbid` is violated. Partial credit is not awarded at the instance level — this mirrors SWE-bench's all-or-nothing patch grading and keeps the category honestly hard.

End-state versus trajectory: the central design decision

The most-argued question in the ORCHESTRATE design is whether to grade the destination or the journey. End-state-only grading is clean and hard to dispute — the model reached the right world state or it didn't; the path is irrelevant. Trajectory-aware grading is more faithful to what “good” means in production: a model that reaches the correct end state in three tool calls is genuinely better than one that gets there after twelve redundant calls, three failed guesses, and a lucky recovery.

GTM-Bench refuses to collapse the two into one number: the headline score is end-state only, and trajectory efficiency is reported as a separate diagnostic.

The headline ORCHESTRATE score is end-state only — the un-arguable pass/fail — so the leaderboard rests on the metric nobody can contest. Trajectory efficiency is reported separately: tool-call count relative to the reference solution, redundant-call rate, and forbid-adjacent near-misses. A model can top the pass-rate board and still show a poor efficiency profile, and both facts are visible.

Failure modes the grader is built to catch

The confident narrator. A model that says “I've updated the record and enrolled the contact” without actually calling the tools. Under prose grading this reads as success; under end-state grading it fails flatly — the sandbox state is unchanged. This single failure mode is the strongest argument for execution grading in the first place.

The over-actor. A model that reaches the target state but also fires a dozen unrequested actions to hedge. Caught by `forbid` clauses and surfaced in the efficiency diagnostic. The wrong-argument success. Right sequence, wrong contact; right field, wrong value. Caught because assertions check specific values, not just that a call was made.

The order-dependent failure. A model that does the right things in an order that breaks a dependency — enrolling before the record exists. Caught because the sandbox evolves state step by step and a call against absent state returns an error the model must handle. The give-up. A model that stalls, loops, or exits without completing. Caught by a step budget per instance and a clean “incomplete” verdict rather than a hung grader.

Each of these is invisible to a benchmark that grades what the model says. All of them are visible to one that grades what the model did.

Why this is the category to watch

If GTM-Bench has a single number worth defending under technical due diligence, it is the ORCHESTRATE pass rate — because it is the one that cannot be reached by fluent text alone. It requires a model to plan, to call real tool interfaces correctly, to react to returns, and to leave a deterministic environment in exactly the right state and no other. That is the actual shape of agentic GTM work, and it is the capability the rest of the field currently measures with demos. The design here is deliberately conservative, and conservative is the point: a flagship category earns its authority by being the one nobody can argue their way past.

References

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

2. Yao et al. tau-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. 2024.

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