

How to choose an AI benchmark

Choose an AI benchmark by defining the decision first, then checking task fit, grader validity, data access, saturation, and provenance.

Updated October 15, 2026

A useful AI benchmark measures a capability that matters to your decision, under conditions close enough to your intended use that the result changes what you do.

Start with the decision

Write the decision before looking at a leaderboard. “Choose a support model that follows refund policy while using tools” is a decision. “Find the smartest model” is not. The first statement names a workflow, a failure cost, and behavior you can observe. The second hides all three.

NIST's draft practices for automated language-model evaluations put this in measurement terms: define the evaluation objective, the construct to measure, and the intended use of the result. That order prevents a familiar mistake—choosing a popular score and inventing a reason for it afterward.

Turn the decision into two lists. The **must catch** list names failures that would block deployment: unauthorized purchases, fabricated citations, broken code, or unsafe tool calls. The **nice to know** list names secondary differences such as style or small latency gains. Choose benchmarks for the first list before spending budget on the second.

Match the task, not the label

Two benchmarks filed under “agents” can measure different things. One may test web navigation in a fixed browser. Another may test recovery after an API timeout. Read the unit of work, allowed tools, stopping condition, and reference answer. A task is relevant only if success depends on behavior your system will need.

The grader matters just as much. Exact match is cheap and repeatable but can reject valid variants. Unit tests are strong for executable behavior, provided the tests cannot be weakened. A simulator can verify state changes without touching a live service. Human and model judges can handle open-ended work, but their rubric, order effects, and repeatability deserve inspection.

Compare, for example, [Benchmark Tool Errors](#) with [Benchmark Hundred Step](#). Both concern agents, but the first targets fault recovery while the second targets reliability across dependent steps. A buyer worried about duplicate charges should start with error recovery, not a generic long-task score.

Check data and access

Find out whether the task data is open, gated, available by request, or unavailable. Then read the licence. “Public on GitHub” does not automatically mean permission to copy, modify, or

redistribute. Open data makes local inspection and reproduction easier; a hidden test split can reduce direct memorization but asks you to trust the operator's controls.

Look for dates and construction details. When were tasks collected? Were they written by experts, adapted from another dataset, generated, or taken from real activity? Is there a contamination cutoff? Can you inspect a development split without seeing the scored split? The answers affect both validity and your ability to debug failures.

Read saturation and uncertainty together

A benchmark is less informative near its ceiling because several models can cluster within a narrow band. Do not treat an 89.4 score as meaningfully above 89.1 without uncertainty estimates. Prefer per-task results and confidence intervals over a single decimal.

Low saturation is not proof of quality. A broken or ambiguous benchmark can also produce low scores. Use saturation as a filter after task fit and grader validity, not before. If every serious candidate is near the ceiling, build a harder private slice or choose a maintained benchmark with refreshed tasks.

Trace the score

Ask who ran the reported top score. A lab self-report is useful evidence, but an independent run tells you more about reproducibility. A reproduced result with public configuration, model version, prompt, seed, tool environment, and raw outputs is stronger still. If the source does not identify these details, mark provenance unknown rather than filling gaps with assumptions.

Stanford's HELM is a useful reference for transparent reporting because it exposes scenarios, prompts, metrics, and reproducible framework details. You do not need to adopt its exact suite to copy the habit: make each score traceable to a run.

Make a shortlist

Use this order:

1. Reject tasks that do not match the intended decision.
2. Reject graders that cannot distinguish the failures that matter.
3. Check whether the data licence and access terms permit your run.
4. Prefer benchmarks with room below the ceiling and a credible refresh plan.
5. Prefer independently reproduced scores and complete run metadata.
6. Estimate cost on the actual task count, including retries and judge calls.

The [benchmark chooser](#) can narrow the catalog by category and saturation. It is a shortlist, not a verdict. Read the primary source for every finalist, run at least one by hand, and add a small private set drawn from your own workflow. That last step catches the gap between a clean public task and the messy system you actually plan to ship.

Frequently asked questions

Should I use the benchmark with the biggest leaderboard?

No. Use the benchmark whose tasks and grader match the decision you need to make.

Is one benchmark enough?

Usually not. Pair a broad capability check with a task-specific evaluation and a small set of your own cases.

Sources

[NIST Practices for Automated Benchmark Evaluations of Language Models](https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.800-2.ipd.pdf) — <https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.800-2.ipd.pdf>

[Stanford HELM Capabilities](https://crfm.stanford.edu/helm/capabilities/v1.9.0/) — <https://crfm.stanford.edu/helm/capabilities/v1.9.0/>