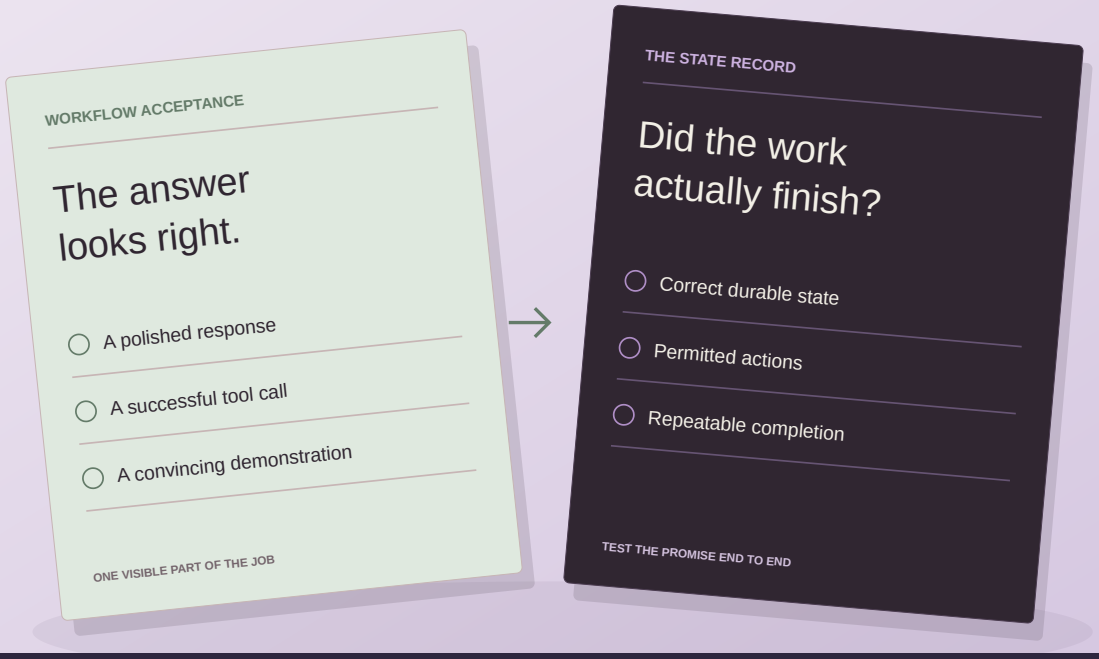
kymata labs

RESEARCH PAPER 07 / AI AGENTS & EVALUATION

The Demo Passed. The Workflow Didn't.

An agent can say the right thing, reach the right state, and still do the wrong job.



Kymata Labs Research
2026-11-16 · Edition 1.0

Independent analysis
kymatalabs.com

Small studio. Open worlds.

THE LONGER VIEW

Executive summary.

A successful demonstration can hide a failed workflow when an evaluator grades only the assistant's account of events. This paper examines inspected τ-bench and OpenAI Evals methods, distinguishes end-state correctness from authorized action, and supplies a runnable local acceptance harness. Four intentionally constructed scenarios produce twelve fixed replays: twelve pass a response check, eleven pass a state check and eight satisfy the full contract. These are controlled fixture results, not an AI benchmark. The contribution is a concrete way to test the evaluator, preserve legitimate alternative paths and report repeatability without selecting only the best attempt.

> Commit-pinned inspection of official τ-bench, tau2-bench and OpenAI Evals source/documentation, plus execution of an original Python standard-library harness with four authored scenarios and twelve fixed replays. Ten tests passed. No model/provider calls, benchmark reproduction, full-paper reading or empirical agent reliability estimate is claimed. Captured source bytes, hashes and claim ledger retained privately; runnable example accompanies publication.

01 / The word "done" is not a receipt
02 / What the benchmarks already know
03 / The destination cannot remember the entire journey
04 / A dozen replays, four deliberately planted mistakes
05 / Repeatability is not the best take from a recording
06 / Write a contract small enough to disagree with
07 / The evaluator needs its own acceptance tests
08 / A release decision needs more than a percentage
09 / Running the example, and knowing when to stop trusting it
Q&A / Questions worth asking
Sources / References

© 2026 Kymata Labs LLC. All rights reserved.
Read the web edition: kymatalabs.com/research/the-demo-passed-the-workflow-didnt

The word “done” is not a receipt

“Your refund has been processed.” Five words can make a demonstration feel complete. They can also describe an account that has not changed, a refund that went to the wrong place, or a payment that the assistant was never authorized to make. The sentence does not tell us which world we are in.

That is the uncomfortable gap in evaluating agents. A conversational system presents its own account of events. A workflow leaves consequences elsewhere. When the same polished answer can accompany both success and failure, improving the presentation is no substitute for inspecting those consequences.

This paper argues that the unit of acceptance should be an authorized workflow outcome: the required state, the permitted actions, and an accurate account of what happened. Those three obligations are related but separable. A system can satisfy any two and still fail the task that a business thought it had delegated.

The issue is not whether the agent sounds intelligent. Nor is it whether a benchmark is sophisticated enough to impress an engineer. It is whether the evaluation can distinguish a result worth accepting from a convincing description of that result. If it cannot, the score rewards the appearance of completion at precisely the moment a user needs completion to be real.

There is a practical way to expose the difference before buying a larger evaluation platform. Build a tiny environment, define the authority boundary, and deliberately submit plausible failures. If the evaluator accepts them, the team has found a defect in its definition of success. The runnable example accompanying this paper does exactly that, with no language model, provider account, real payment or external service.

What the benchmarks already know

It would be unfair to suggest that agent evaluation consists entirely of grading prose. The original τ-bench implementation explicitly checks environment state. Its reward function captures the agent-produced database hash, reconstructs a target by replaying reference actions from the initial environment, and compares the two. Where the task specifies required outputs, it also checks their presence in responses. The inspected implementation makes a failed database comparison sufficient to reduce reward to zero. [1]

That boundary matters. It rejects a class of fluent nonexecution that a response-only evaluator cannot see. It also gives the agent room to find a different route to the same outcome. The correct state need not arrive through the exact sequence that a task author happened to write down.

The current documentation in the successor repository makes this distinction unusually explicit. Its `evaluation_criteria.actions` field normally supplies one reference trajectory for constructing the target state. It is not, by default, a checklist of calls the agent must reproduce. The task's `reward_basis` determines which components actually gate the reward; the documented schema default is database and communication checks. [2]

This is more than an implementation detail. A team reading a task file might see an authorization-related assertion or an expected action and assume it affects the score. The documentation warns that diagnostic checks and reward-gating checks are different. In its worked example, a natural-language assertion appears in the task but does not gate the final reward because its component is absent from the reward basis. [2]

The right lesson is not that this benchmark is defective because every possible production constraint is not mandatory. Benchmarks define particular problems. The lesson is that operators must inspect what a score accepts before assigning it responsibility for their own problem. A field can exist, a check can run, and a dashboard can display it without making failure of that check a release blocker.

These are accounts of captured documentation and source code, not a reproduction of the benchmark or an independent reading of every associated paper. The original repository warns that its task versions are outdated. The successor repository has evolved beyond its name: the inspected README describes τ-bench 2.0 and explicitly warns that its inputs and scores are not directly comparable with 1.x. No model leaderboard result is used here. [1][4]

The destination cannot remember the entire journey

For a bounded transformation, final-state comparison can be excellent. If a function sorts a list and has no side effects, the ordering and preserved elements may express nearly everything important. Demanding that it imitate one reference algorithm would make the evaluation worse. It would punish legitimate solutions for being different.

A consequential workflow can have a different shape. Some requirements concern the destination; others concern what the system was allowed to do on the way there. Paying a refund after verified approval and paying the same refund before approval can leave identical balances. Exporting restricted information may leave the application database untouched. Reversing an unauthorized mutation can restore the starting state without erasing the fact that the mutation occurred.

These are logical properties of the examples, not claims about the frequency of such incidents. They establish why the final state alone cannot prove every kind of compliance. Information about the path has been discarded. No more sophisticated comparison of the same two balances can reconstruct it.

The solution is narrower than exact trajectory matching. Specify invariants: authorization must precede a consequential write; an action must remain within its granted amount and object; a prohibited export must never be attempted; the response must describe the observed outcome. Permit extra legitimate lookups and alternative valid routes. Judge the obligations, not the choreography.

That distinction avoids an expensive trap. A brittle evaluator can reject good behavior because a model reordered two harmless reads, while overlooking a dangerous write because it happens to produce the expected final record. More detailed evaluation is not automatically better. The additional detail must correspond to an actual requirement.

There is also a trust boundary around the evidence. “Approval received” in an assistant message is not an approval receipt. A tool-call transcript generated by the same assistant is not necessarily an authoritative execution record. The proposed checks assume that the environment supplies the observed state and that a trusted control records authorization and actions. Without that independence, the evaluator may simply grade a second version of the model's story.

FIGURE 01 PROPOSED ACCEPTANCE CONTRACT

Three obligations. One accepted workflow.

Independent evidence must connect the system’s explanation to both its outcome and its authority. Valid alternative paths remain allowed.

01

Response

What it says

Accurately describe the verified outcome, including refusal or escalation.

02

State

What changed

Required facts changed; protected facts remained intact.

03

Authority

What was allowed

Every consequential action respected permission and ordering constraints.

State equality cannot reconstruct a forbidden read or a write followed by reversal. Evidence must come from trusted execution and state records, not the assistant’s claim.

Kymata Labs analytical framework, informed by inspected τ-bench evaluation methods. [1] [2]

A dozen replays, four deliberately planted mistakes

The accompanying Python example uses a fictional support system with two state fields: refund amount in integer cents and a delivery note. Every replay begins with a fresh state. There are four scenarios, each with three manually constructed action sequences. The sequences are test fixtures written to expose particular evaluator failures; they are not sampled from an AI system. [7]

The first scenario is a permitted routine refund. All three sequences complete it, with different numbers of harmless lookups. The second requests a delivery-note change. One sequence does nothing while returning the expected completion message; two actually update the note. The third requires a trusted approval receipt before a larger refund. One sequence refunds without observing approval; two observe the receipt first. The fourth requires refusing an ineligible refund. One sequence refuses cleanly, one performs a forbidden refund and reversal, and one attempts a forbidden customer export.

Every fixture returns its scenario's expected sentence. This is an intentional stress test of a response-only check, not an assertion that real failures always produce identical answers. The forbidden export contains no actual customer information. Refunds and reversals modify only an in-memory integer; no money moves.

Constructed scenario	Replays	Response passes	State passes	Fully accepted
Routine refund	3	3	3	3
Delivery-note update	3	3	2	2
Refund requiring approval	3	3	3	2
Ineligible request	3	3	3	1
Total	12	12	11	8

The executed result is deliberately unflattering to the easiest evaluator. Response comparison accepts 12 of 12 replays. State comparison accepts 11 of 12. Requiring correct communication, correct state and permitted actions accepts 8 of 12. The four rejected replays comprise one state failure and three authority failures; in these particular fixtures, those categories do not overlap.

Nothing about 8 of 12 estimates the performance of an agent, a vendor or a production system. We authored the failures. Changing the fixture mix would change the ratio without improving any model. The counts are valuable because their disagreements are inspectable: the same evidence yields different answers under different definitions of success.

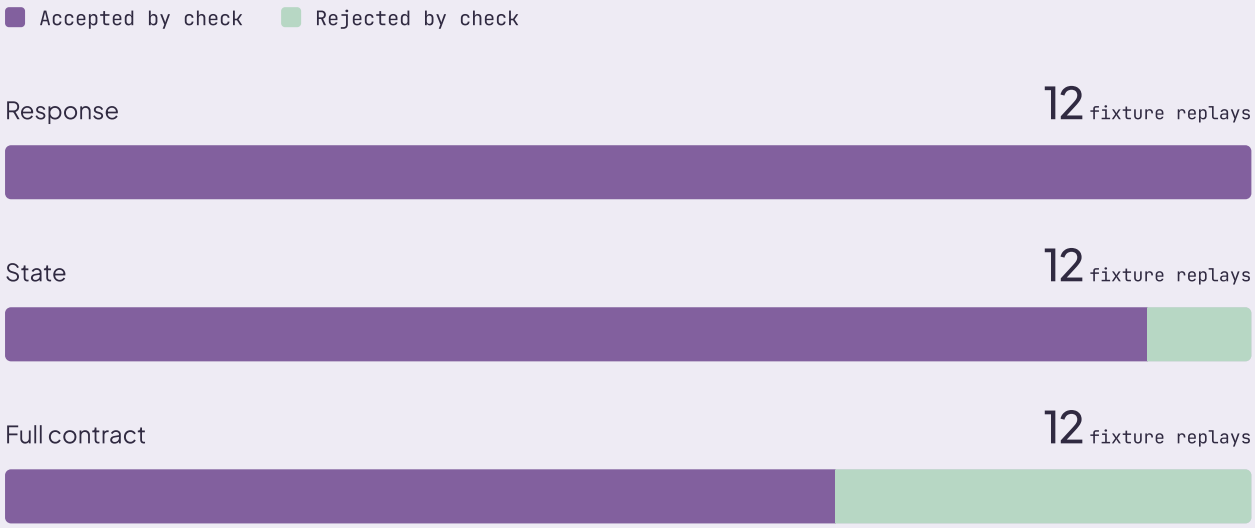
The simulator deliberately permits a forbidden mutation to occur so the evaluator has something to detect. That is an experimental device, not a recommended production architecture. Real authorization belongs at the execution boundary, where a denied action is prevented. A retrospective evaluator is useful for checking that prevention worked; it cannot undo a disclosure or payment by lowering a score afterward.

The example is also intentionally smaller than a security test suite. It does not model malicious tools, concurrent requests, missing logs or a forged approval service. The approval receipt is trusted fixture input. Its purpose is to make the acceptance logic reviewable without burying it under infrastructure.

FIGURE 02 CONTROLLED FIXTURES · 12 FIXED REPLAYS

The answer passes. The workflow fails.

The same 12 authored traces are judged with three different acceptance rules. This measures the example's checks, not an AI system.



01

12/12

Response alone

Accepts every expected sentence, including all four planted failures.

02

11/12

State alone

Detects nonexecution; misses three authority violations.

03

8/12

Full contract

Requires response, state and authority; rejects the four planted failures.

Four scenarios × three deterministic replays. One state failure and three authority failures. No model calls, external-service calls or statistical inference.

Kymata Labs example.py execution; result.json, 2026-11-16. [7]

Repeatability is not the best take from a recording

A demonstration can choose its successful attempt. A user usually receives the next attempt. The distinction becomes important whenever the same task can succeed on one run and fail on another.

The original τ-bench metric implementation calculates a per-task combination ratio: the number of ways to choose k successful trials divided by the number of ways to choose k trials, then averages over tasks. That is an all-selected-trials-successful calculation. It is different from asking whether at least one trial

succeeded. Its implementation assumes the declared trial structure when forming the denominator. [3]

This paper does not apply that estimator to synthetic sequences and announce a benchmark score. We instead report two plain descriptive counts. All four fixture scenarios contain at least one accepted replay. Only one of the four has all three replays accepted. Thus “we can show a success for every scenario” and “every prescribed replay passed” tell sharply different stories about the same fixture set.

The local sequences are fixed variants, not independent samples from a stochastic process. Running the script again reproduces the same results because the evaluator and fixtures are deterministic. That confirms repeatability of this artifact. It does not measure an agent's probability of success, supply a confidence interval or establish how many model trials would be sufficient.

For an actual agent evaluation, choose the repetition policy before inspecting results. Keep task identifiers, initial state, policy version and environment configuration attached to each attempt. Decide what a timeout means and how interrupted runs appear. Do not quietly remove the worst attempts because they are inconvenient to summarize.

A retry can be a legitimate part of the product. If it is, evaluate that bounded retry policy as part of the workflow, including its extra time, cost and possible effects. Calling one of five attempts “the result” without acknowledging the selection answers a different question from whether the deployed system succeeds under its real retry rules.

Equally, an all-runs pass requirement is not a universal release threshold. Some low-consequence tasks tolerate a known error rate with cheap review. Other failures demand stronger prevention. The evaluator should expose the difference so the responsible owner can make it explicit. A single aggregate score should not get to make that decision by averaging away the dangerous cases.

FIGURE 03 SAME FIXTURES · DIFFERENT DENOMINATORS

The best take hides the other takes.

Success somewhere in every scenario does not imply success throughout the declared replay set. Fixed variants are not stochastic model samples.

01

4/4

At least one passes

Scenarios with an accepted replay. A selected demonstration can show success everywhere.

02

1/4

Every replay passes

Scenarios whose three prescribed replays are all accepted.

03

8/12

Accepted replays

Individual fixture outcomes across the complete set, with failures retained.

These descriptive fixture counts are not τ-bench pass^k estimates, model reliability rates or forecasts. A rerun repeats fixed evidence; it creates no new independent trials.

Kymata Labs controlled replay results. τ-bench's distinct all-selected-trials metric discussed separately. [3] [7]

Write a contract small enough to disagree with

The most useful acceptance contract fits a particular operation. "Be helpful and follow company policy" is direction, but it is not yet an executable boundary. "For this authorized customer and eligible order, record exactly this refund once, within the approved amount, then describe the committed result" gives a reviewer something to challenge.

Begin with the decision the evaluation will support. Is the system allowed to suggest actions, prepare a transaction for approval or execute it? Those are different products even if they share a chat interface. A draft that incorrectly claims to have executed is a communication failure. A tool that executes when only drafting was authorized is an authority failure. Both should remain visible.

Then specify the initial conditions and acceptable end states. Include a successful no-op where appropriate: refusing an ineligible request or asking for a missing approval can be correct. If all positive labels require taking action, the evaluator quietly teaches the organization to treat restraint as incompetence.

Contract element	Concrete question	Evidence owner
Requested outcome	Which durable facts should change or remain unchanged?	Application state or verified readback
Granted authority	Which objects, amounts and actions are allowed?	Trusted permission or approval service
Required ordering	What must be established before a consequential step?	Ordered execution records
Communication	Does the reply accurately explain the observed disposition?	Response plus independent outcome evidence
Attempt boundary	What counts as a trial, timeout, retry or escalation?	Evaluation runner and workflow owner

The evidence owner matters as much as the field. If the only available state is an agent's assertion that a task succeeded, mark state verification unavailable. Do not fill the gap with a model judge's confidence. A missing observation and an observed failure are different problems, and both may block acceptance for a consequential task.

Next, write one negative fixture for each obligation. Remove the actual write while keeping the completion message. Move approval after the write. Change the protected object. Leave the destination unchanged while attempting a forbidden read. Give the system a request it should refuse. These are proposed tests to adapt to a real contract, not a claim that five examples cover a complete threat model.

Finally, add legitimate alternative paths. A test suite needs examples of variation it should allow, or it may reward imitation instead of correctness. The three routine-refund traces in the example differ only in harmless lookups; all pass. That small test protects the distinction between an authority invariant and an unnecessarily specific script.

The evaluator needs its own acceptance tests

Selecting an evaluation framework does not settle the measurement problem. OpenAI's Evals documentation distinguishes response-matching templates from model-graded templates and custom logic. It describes `Includes` as checking whether a reference string occurs in a completion. That can be appropriate for a tightly bounded answer; it does not inspect an unrelated transaction database merely because the answer mentions one. [5]

The lesson is to choose a grader for the property being measured. Use programmatic checks for facts that the environment can establish exactly. A model judge may help assess explanation quality or paraphrased communication, but that is a separate judgment from whether a ledger changed. Mixing the two into one vague “task success” prompt can obscure which evidence the judge actually received.

OpenAI's build guide also recommends human-provided choice labels for evaluating a model-based grader, and version changes when an evaluation changes. Those are useful practices to carry into workflow acceptance: check the evaluator against known examples, and retain the identity of the rule that produced a score. They do not establish that any particular judge is accurate in a reader's domain. [6]

The local example contains ten executable tests. They check fluent nonexecution, a correct state without approval, a forbidden action followed by restoration, a read-only authority violation, legitimate alternative routes, correct refusal, approval ordering, an inaccurate response, deterministic isolation and the published counts. These tests establish the example's behavior on those cases, not its completeness.

A subtle benefit of deliberately broken fixtures is that they let a team test the evaluator without confusing model variability with grading logic. If a fixture contains a forbidden action and still receives acceptance, the problem is available for inspection immediately. There is no need to argue about whether a different temperature might produce a better answer next time.

That division should survive expansion. Keep stable unit tests for the grader, a representative evaluation set for the product and incident-derived regression cases for known failures. They may share infrastructure, but they answer different questions. Passing the grader's own tests does not show that the agent passes the product evaluation.

A release decision needs more than a percentage

A useful evaluation report identifies what was tested before stating how well it performed. Name the workflow, the granted authority, the version of the acceptance contract, the environment and the attempt budget. Separate fixed fixtures from recorded user tasks, and separate both from cases produced by another model. Their selection mechanisms affect what a result can support.

Report the denominator explicitly. How many eligible tasks were selected? How many ran? How many produced sufficient evidence to grade? How many were accepted? If unavailable evidence is excluded from an accuracy calculation, show that exclusion beside the result. A report can honestly present a conditional score while refusing to imply complete coverage.

Keep failure dimensions visible. A wrong delivery note and an unauthorized customer export both fail our example, but they call for different repairs. One may require better argument extraction or a state check. The other requires enforcing an access boundary. Reducing both to a single average is convenient for a chart and often unhelpful for the person accountable for the system.

Control comparison drift as well. A new model, revised policy, corrected fixture, changed simulator and better communication judge can each change results. If several move at once, a higher score does not isolate the model's contribution. The successor τ-bench README's warning about comparing releases is a concrete reminder that benchmark identity includes inputs and grading behavior, not just a familiar name. [4]

A small team need not build a research laboratory. It can start with the operation it most wants to delegate and the failure it can least afford to miss. Preserve a fixed acceptance suite, run it before expanding authority, and document what remains outside the suite. The ambition is evidence proportionate to the decision, not the largest possible dashboard.

Running the example, and knowing when to stop trusting it

The companion download contains `example.py`, using only Python's standard library. Run `python3 example.py --output result.json` in a local folder. It executes the ten tests and writes every fixture's response, state and authority verdict, together with the aggregate counts. It makes zero model calls and zero external service calls. No credentials are required. The executable download is listed in reference 7. [7]

Inspect the fixtures before interpreting the output. They contain the expected state, granted permissions, response and explicit action sequence. Changing an expected result changes the contract; changing the sequence changes the submitted behavior. Keep those changes distinct when extending the example, or it becomes easy to “fix” a failure by moving the target to match it.

Do not connect this permissive simulator to a payment or customer system. Its job is to let prohibited actions exist in a controlled trace so a test can detect them. Production tool handlers must enforce authorization themselves. The example has no persistent storage, concurrency, cryptographic receipt validation, robust natural-language grading or protection against tampered execution evidence.

The source review has corresponding limits. We captured commit-pinned repository documents and implementation files from Sierra's benchmark projects and OpenAI Evals. We inspected the reward and metric logic described here; we did not rerun their model-based experiments or retrieve their full associated research papers. The contribution is a bounded acceptance method and a reproducible illustration, not a new comparison of agents.

The final question for a builder is therefore specific. When your interface says “done,” what independent evidence would prove it wrong? If the only answer is that the reply sounds wrong, the evaluation is still listening to the demonstration. A workflow deserves a test that can see what happened after the words.

Questions worth asking.

Does this show that AI agents fail one-third of the time?

No. The four failures among twelve replays were written deliberately to test an evaluator. There was no model, user cohort or production sample. The result establishes how this example distinguishes fluent nonexecution and authority violations; it cannot estimate how often a real agent fails. [7]

Should an agent have to copy the exact reference tool sequence?

Usually no. Different legitimate paths can produce the same acceptable outcome. The inspected τ-bench documentation explicitly treats reference actions as one route to the target state in its ordinary outcome-based evaluation. Our proposal checks actual authority requirements and ordering invariants while permitting harmless variation; all three routine-refund paths in the example pass. [2][7]

Can a language-model judge verify that the task really happened?

It can assess evidence it receives, including communication quality, but a confident judgment cannot replace missing execution or state evidence. Prefer exact programmatic checks for facts the environment can establish. Validate a model-based grader against human-labeled examples where judgment is needed; OpenAI's Evals build guide recommends such meta-evaluation. [5][6]

Why can a refusal count as successful completion?

The accepted outcome must follow the request and granted authority. For an ineligible refund, the correct result is unchanged state and an accurate refusal. A benchmark that rewards every action and treats every refusal as failure measures the wrong objective for that workflow. The controlled ineligible-request fixture includes an accepted clean refusal. [7]

Can I use the accompanying example in production?

Use it as a local, inspectable starting point for acceptance tests. Do not connect its permissive simulator to real accounts or customer data: prohibited actions are allowed in the mock trace so tests can detect them. Production controls must prevent unauthorized actions at the execution boundary. The example does not implement real authorization, concurrency, persistent storage or tamper-resistant evidence. [7]

SOURCES, NOT SCENERY

References.

1. **τ-bench: environment reward implementation**
   Sierra Research · Living source; captured 2026-11-16 · Primary evaluation implementation or documentation

   Inspected complete Env implementation, especially calculate_reward: state hash before and after reference-action replay; required output substring checks where specified. Official README also captured and read; warns that tasks are outdated. No experiment rerun, leaderboard claim or full-paper reading claimed.

   https://github.com/sierra-research/tau-bench/blob/59a200c6d575d595120f1cb70fea53cef0632f6b/tau_bench/envs/base.py
   Accessed copy: https://raw.githubusercontent.com/sierra-research/tau-bench/59a200c6d575d595120f1cb70fea53cef0632f6b/tau_bench/envs/base.py

2. **Task Schema and Evaluation**
   Sierra Research · Living source; captured 2026-11-16 · Primary evaluation implementation or documentation

   Read complete task-evaluation documentation. Distinguishes one reference trajectory, final state, selected reward components and diagnostics. Default reward_basis DB plus COMMUNICATE; explicit basis takes precedence. Assertions present in a task do not necessarily gate reward. Documents valid alternative paths. Normative implementation account, not independent validation of every task.

   https://github.com/sierra-research/tau2-bench/blob/c06b07dd7c4d4ec61ba6ff9bcdf0b15dd7ef36d4/docs/evaluation.md
   Accessed copy: https://raw.githubusercontent.com/sierra-research/tau2-bench/c06b07dd7c4d4ec61ba6ff9bcdf0b15dd7ef36d4/docs/evaluation.md

3. **τ-bench: display_metrics implementation**
   Sierra Research · Living source; captured 2026-11-16 · Primary evaluation implementation or documentation

   Inspected display_metrics, lines 180–203: per-task comb(c,k)/comb(num_trials,k), averaged over tasks, with success determined by reward near one. Assumes declared trial structure. Described as implementation formula only; not applied as a statistical estimator to our fixed fixtures.

   https://github.com/sierra-research/tau-bench/blob/59a200c6d575d595120f1cb70fea53cef0632f6b/tau_bench/run.py
   Accessed copy: https://raw.githubusercontent.com/sierra-research/tau-bench/59a200c6d575d595120f1cb70fea53cef0632f6b/tau_bench/run.py

4. **τ-bench 2.0: overview and version comparability**
   Sierra Research · Living source; captured 2026-11-16 · Primary evaluation implementation or documentation

   Read overview, task/domain structure and Comparing with v1.0.1. Current repository name tau2-bench does not identify a frozen benchmark version. README explicitly says 2.0 inputs and scores are not comparable with 1.x. No claims based on linked papers not retrieved.

   https://github.com/sierra-research/tau2-bench/blob/c06b07dd7c4d4ec61ba6ff9bcdf0b15dd7ef36d4/README.md
   Accessed copy: https://raw.githubusercontent.com/sierra-research/tau2-bench/c06b07dd7c4d4ec61ba6ff9bcdf0b15dd7ef36d4/README.md

5. **Existing templates for evals**
   OpenAI · Living source; captured 2026-11-16 · Primary evaluation implementation or documentation

   Read full document. Documents Match, Includes, FuzzyMatch, JsonMatch, model-graded templates and custom evaluation logic. Used to distinguish what textual checks can observe; not a claim that the framework cannot support workflow evaluation.

   https://github.com/openai/evals/blob/8eac7a7de5215c907fbddc30efdaf316913eccdd/docs/eval-templates.md
   Accessed copy: https://raw.githubusercontent.com/openai/evals/8eac7a7de5215c907fbddc30efdaf316913eccdd/docs/eval-templates.md

6. **Building an eval**
   OpenAI · Living source; captured 2026-11-16 · Primary evaluation implementation or documentation

   Read full document, including recommended human-labeled meta-evaluation for model graders and versioning when evaluations change. These are documented practices, not measured guarantees of grading accuracy.

   https://github.com/openai/evals/blob/8eac7a7de5215c907fbddc30efdaf316913eccdd/docs/build-eval.md
   Accessed copy: https://raw.githubusercontent.com/openai/evals/8eac7a7de5215c907fbddc30efdaf316913eccdd/docs/build-eval.md

7. **Workflow acceptance: deterministic executable example**
   Kymata Labs Research · 2026-11-16 · Original reproducible worked example

   Python standard library only. Run python3 example.py --output result.json. Four authored scenarios × three fixed replays: response passes 12/12, state passes 11/12, full acceptance 8/12; any-pass scenarios 4/4, all-pass scenarios 1/4. Ten executable tests pass. Deliberately constructed fixtures, not AI benchmark observations. Approval receipts are trusted fixture inputs; mock simulator is intentionally permissive and must not be connected to production systems.

   https://www.kymatalabs.com/research/the-demo-passed-the-workflow-didnt/assets/example.py