Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

# CI-DeceptiveTest: An Execution-Paired Benchmark for Test Assurance in AI-Assisted Development

**Qining Li**

castelu@qq.com

Zhejiang University of Science and Technology

**Article**

**Keywords:** software testing, continuous integration, large language models, test oracles, mutation testing, benchmark datasets

**Posted Date:** November 13th, 2026

**DOI:** https://doi.org/10.21203/rs.3.rs-11393646/v1

**License:** This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

**Additional Declarations:** No competing interests reported.

# CI-DeceptiveTest: An Execution-Paired Benchmark for Test Assurance in AI-Assisted Development

Qining Li*
Zhejiang University of Science and Technology
castelu@qq.com

**Abstract**

AI-assisted development weakens the meaning of a green continuous-integration (CI) check: generated tests may execute realistic code and pass while replacing semantic oracles with tautologies or coarse smoke checks. We introduce CI-DeceptiveTest, a Python execution-paired benchmark and Gatekeeper protocol for detecting such low-assurance tests. Its 600 pairs originate from real tests in 12 open-source repositories; each source-control test is paired with a candidate produced by controlled oracle weakening, yielding 1200 label-free static-binary views. Construction combines deterministic validation, execution evidence, a 20% manually annotated slice, random inspection of 30% of AI-assisted records. The evidence is GPT-5.5-first: GPT-5.5 supports construction, review, and full-dataset validation. Labels are transformation-defined, not model-assigned. On the full benchmark, transparent static rules achieve 0.928 low-assurance recall but a 0.458 false-block rate on accepted source-control tests. `mimo-v2.5-pro` is retained as an auxiliary external cross-check; it reaches 0.885 macro F1 and reduces false blocking to 0.048. Bounded mutation analysis over 42 pre-specified source/candidate pairs further supports the paired construction. The central stress result is that weak-oracle detection remains hard: plausible coarse assertions retain execution while losing semantic constraint. The benchmark enables CI gatekeeper evaluation in a controlled, reproducible setting.

**Keywords:** software testing; continuous integration; large language models; test oracles; mutation testing; benchmark datasets.

## 1 Introduction

AI-assisted programming is moving from a separate prototyping activity into ordinary pull-request work. Developers increasingly ask assistants and coding agents to implement changes, repair failures, and add tests within the same loop. Informal "vibe-coding" workflows make the shift visible: describe the desired behavior, accept generated code, ask for tests, and iterate until CI turns green. This workflow lowers the marginal cost of producing tests. It also shifts the trust problem faced by CI. A passing generated test is not, by itself, strong evidence that the change is better tested, because the cheapest way to satisfy a testing request may be to produce test-shaped code that checks too little.

The failure mode is subtle precisely because the test still runs. A regression test whose source-control oracle checks an exact value, an exception condition, a structured object, or a domain invariant may be weakened into a variant that keeps the setup, calls the same function, and covers the same code. What is lost is the behavioral constraint: the oracle becomes `assert result is not None`, a length check, a type check, a prefix check, or an assertion that is true by construction. Such a test is not necessarily malicious, and it may still be useful as a smoke

*Corresponding author: Qining Li, castelu@qq.com.

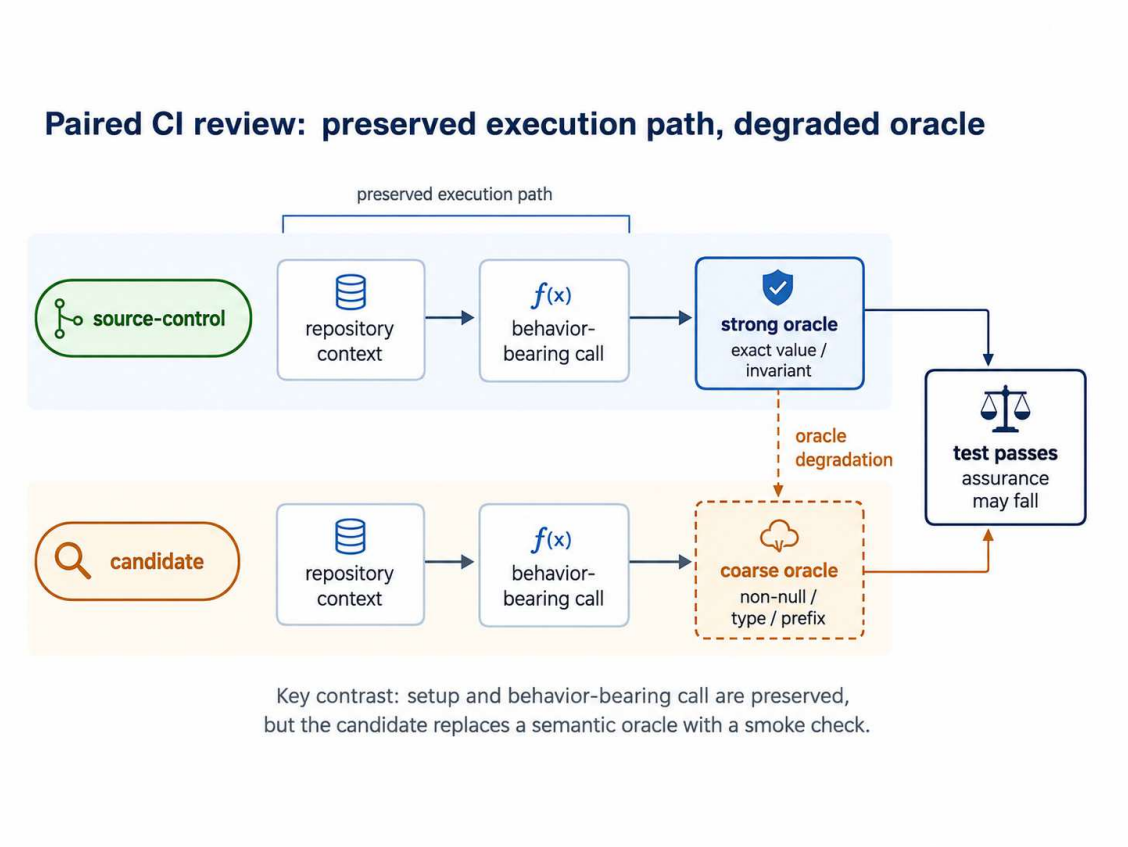



Figure 1: Oracle degradation in paired CI review. The candidate preserves the execution path of the source-control test but weakens the oracle, so passing execution no longer implies the same behavioral assurance.

test. It is, however, weaker than the source-control oracle it resembles. We call these *low-assurance tests*. The term is operational: it describes a paired benchmark relation, does not imply malicious intent, and does not assign developer motivation. Figure 1 illustrates the paired contrast: the candidate can preserve the execution path while replacing a semantic oracle with a coarse smoke check.

This paper studies the CI review problem downstream of test generation. Prior work asks whether LLMs can generate tests, improve coverage, repair tests, or kill mutants [22, 31, 13, 21, 19, 5]. CI maintainers face a different question once a test already passes: should this passing test count as assurance for the change? We therefore do not ask whether a model can write a test from scratch. We ask whether a gatekeeper can audit an execution-passing test and decide whether it still constrains behavior.

We introduce CI-DeceptiveTest, a Python execution-paired benchmark and Gatekeeper protocol for detecting low-assurance tests in LLM-assisted development. CI-DeceptiveTest is a real-test-derived controlled counterfactual benchmark. It contains 600 paired cases that begin with real open-source Python tests; each case retains the accepted source-control test and adds a counterfactual candidate produced by controlled oracle weakening. The benchmark exposes these pairs as 1200 static-binary input views whose labels are hidden from detectors. The paired design is the central methodological choice: a detector must distinguish a meaningful test from a plausible weak alternative drawn from the same repository context, while source-control false blocking is scored explicitly.

The paper makes three contributions.

1. **A real-test-derived counterfactual benchmark for test assurance.** CI-DeceptiveTest

starts from accepted open-source tests and derives controlled weak-oracle counterfactuals rather than hand-writing toy examples. Its labels are transformation-defined, execution-backed, built and validated through a GPT-5.5-assisted workflow with human-supervised QA, and paired with source-control oracles rather than assigned by a model judge.

2. **A Gatekeeper protocol.** We define a policy-separated CI workflow in which detectors emit structured evidence and a fixed PASS/WARN/FAIL policy maps that evidence to review outcomes. The LLM is a detector, not the merge authority, and untrusted natural-language context cannot rewrite the gate policy.

3. **Baseline evidence and failure analysis.** We compare a transparent static-rule floor, GPT-5.5 full-dataset validation, a follow-up `mimo-v2.5-pro` auxiliary judge, Gatekeeper policy projections, bounded mutation diagnostics, and supervised AST-feature stress tests. The results identify weak oracles as the central unresolved slice and show why false blocking must be measured alongside low-assurance recall.

The findings are intentionally framed around the benchmark and protocol rather than model competition. Static rules reach high low-assurance recall but falsely block many source-control tests. GPT-5.5 is the primary model-assisted construction and validation path; the follow-up `mimo-v2.5-pro` auxiliary judge greatly reduces false blocking but misses more low-assurance candidates, especially weak oracles. Gatekeeper policy projections show that useful CI behavior is not a single threshold: projects must choose between conservative blocking and audit-sensitive warnings. CI-DeceptiveTest is therefore best read as a benchmark and protocol for making test assurance measurable in AI-assisted development. It does not claim that the current detector can replace maintainer judgment. It provides a controlled way to study when CI should trust a passing test and when it should ask for stronger evidence.

## 2 Background and Related Work

### 2.1 LLM Test Generation Has Shifted the Bottleneck

LLMs are now a standard object of study for software testing. Surveys identify test preparation, test repair, and program repair as central software engineering applications of LLMs [25], and recent roadmap work treats LLM-based testing as an active testing research frontier rather than a narrow code-generation application [2]. Early and recent systems improve compilability, assertion correctness, focal-context selection, search guidance, and project-aware repair: TestPilot evaluates LLM unit-test generation for JavaScript packages, ChatTester and ChatUniTest iterate over generation and repair, and CodaMOSA combines search-based testing with LLM-provided seeds when coverage stalls [22, 31, 3, 13]. Benchmarks such as TestGenEval, MultiFileTest, TestEval, and ULT/PLT move evaluation from toy functions toward project context, targeted coverage, and contamination-aware test generation [8, 27, 26, 7].

This literature establishes the premise for CI-DeceptiveTest: models can produce tests that look operationally useful. The remaining CI question is not whether a test can be produced, but whether a produced and passing test should increase confidence in the change. A generated test may pass while its oracle is absent, circular, or too weak to guard a regression. CI-DeceptiveTest therefore studies the audit problem downstream of generation.

### 2.2 Assurance Requires More Than Execution

Coverage and execution are necessary signals, but they are incomplete proxies for behavioral assurance. Classical automated test generation already exposed the split between executing code and checking behavior: EvoSuite and Pynguin generate executable tests for Java and Python, but their usefulness still depends on generated assertions and fault-revealing power [6, 17, 18].

Mutation testing provides a stronger adequacy signal by asking whether tests kill injected faults rather than merely execute code [9]. Recent LLM work builds on the same lesson. CoverUp steers LLM generation toward uncovered code regions [21]; MuTAP and Meta's ACH use mutation feedback to focus generation on fault-revealing behavior [19, 5]; and EvalPlus shows that stronger tests can expose bugs missed by weaker benchmark suites [14]. Industrial and issue-driven systems make the same point from practice: TestGen-LLM filters candidate tests through build, reliability, and measurable-improvement checks, LIBRO connects bug reports to reproducing tests, and HarnessLLM studies richer harnesses and invariants [1, 12, 16].

CI-DeceptiveTest follows this assurance-oriented view but evaluates a different artifact. It does not optimize a generator or claim to improve a test suite. It asks whether a gatekeeper can detect when a passing test has lost oracle strength relative to a paired source-control test.

### 2.3 Smell Signals Are Necessary but Not Sufficient

The test-smell literature identifies local patterns that reduce test quality, including weak checks, redundant assertions, and exception-handling smells [23]. These signals matter because many low-assurance tests are syntactically visible: no assertion, tautological assertion, non-null-only checks, type-only checks, length or shape checks, and partial string checks. A transparent CI baseline should expose such patterns.

The technical gap is therefore not the absence of individual smell signals or mutation metrics. Existing smell detectors can flag suspicious local patterns, and mutation testing can measure fault-revealing strength when execution budget is available. What they do not directly provide is a paired CI-review benchmark in which each weak candidate is matched to a source-control oracle, source-control false-block risk is a first-class metric, detector inputs are label-free, and PASS/WARN/FAIL policy projections are reproducible from the provided scoring materials.

### 2.4 Real-Test Benchmarks Provide the Right Substrate

Software-engineering evaluation has repeatedly moved toward version-controlled, execution-backed artifacts because toy examples are too easy to overfit. Defects4J and BugsInPy package real bugs and tests for controlled studies in Java and Python [11, 28]; SWE-bench turns resolved GitHub issues and pull requests into repository-level agent tasks [10]. CI-DeceptiveTest follows the same artifact-grounded instinct but asks a different question. It targets oracle-degradation detection rather than bug-fixing success or ecosystem prevalence estimation. It starts from real accepted tests and constructs paired counterfactuals so that a detector can be scored on oracle degradation while false blocking of the source-control side remains visible.

### 2.5 Gatekeepers Must Also Respect Agent Boundaries

LLM-based software-engineering agents combine models with repository context, tools, and external state [15]. A CI gatekeeper built with such components inherits an additional trust boundary: pull-request prose, issue comments, test logs, and natural-language tool outputs are untrusted inputs. OWASP identifies prompt injection as a leading LLM-application risk [20], and NCSC emphasizes that current LLMs do not enforce a reliable instruction/data boundary inside a prompt [24]. InjecAgent and AgentDojo study indirect injection and tool-using agents [32, 4]; and tool-result parsing work shows why structured outputs matter for agent robustness [30]. CI-DeceptiveTest is not a prompt-injection benchmark. However, its Gatekeeper protocol adopts the same engineering stance: untrusted natural language must not override structured test-assurance policy. Table 1 summarizes how this focus differs from adjacent testing, benchmark, and agent-security work.

Table 1: Positioning CI-DeceptiveTest among adjacent software-engineering research lines.

| Research line | Typical artifact | CI-DeceptiveTest focus |
|---|---|---|
| LLM test generation | Generated tests, repair loops, generation benchmarks | Audits already passing tests rather than generating new ones. |
| Coverage and mutation assurance | Coverage/mutation-guided generation and fault-revealing suites | Uses the same assurance motivation, but labels oracle degradation in paired CI-review rows. |
| Test-smell detection | Static patterns that may weaken tests | Treats smell signals as baselines while scoring false blocking of source-control tests. |
| Real-bug and repository benchmarks | Version-controlled bugs, issues, and execution harnesses | Reuses real-test artifacts, but measures controlled oracle degradation rather than bug-fixing success or prevalence estimation. |
| Agent security | Prompt-injection and tool-use attack tasks | Informs the Gatekeeper boundary; not the static-binary task measured here. |

# 3 CI-DeceptiveTest Dataset

## 3.1 Task Definition

CI-DeceptiveTest is a real-test-derived controlled counterfactual benchmark for low-assurance tests. Each input is a label-free test block with neutral metadata such as repository and path. Each output is binary: `source_control_static` if the row is the accepted source-control test, or `low_assurance` if the row is a counterfactual candidate whose oracle has been weakened. Benchmark role, transformation type, subtype, and answer labels are excluded from detector inputs and used only for scoring.

The paired construction is the benchmark's core design choice. Each accepted case starts from a real open-source Python test with an identifiable oracle and an executable task context. We then derive one controlled counterfactual candidate that preserves the behavior-bearing calls, setup, fixtures, helper use, and relevant state transitions, but removes or weakens the oracle. This turns a vague smell judgment into a controlled comparison. The detector is not asked whether a piece of code contains a suspicious pattern in isolation; it is asked whether a passing test still carries the behavioral assurance of the source-control test it resembles.

This construction deliberately sits between two common alternatives. It is not a purely synthetic smell corpus, because every pair is anchored in a real source-control test and repository idiom. It is a controlled detector-evaluation benchmark rather than a prevalence-estimation study: the candidate side is derived by a documented transformation so that the source and candidate share repository context, setup, and behavior-bearing calls. The benefit is counterfactual control. A detector's decision can be attributed more directly to oracle assurance rather than unrelated differences between tests. This is the same reproducibility motivation behind real-bug and repository benchmarks, but applied to test-assurance degradation rather than bug repair or coverage generation [11, 28, 10, 8]. Table 2 summarizes the evaluated benchmark instance, including scale, source-control evidence, mutation diagnostics, and artifact boundaries.

## 3.2 Construction Protocol

The construction protocol is deliberately more conservative than a bulk AST rewrite. For each source test, we identify the behavioral oracle, apply one controlled weakening, and reject the

Table 2: CI-DeceptiveTest benchmark profile.

| Dimension | Evaluated benchmark |
|---|---|
| Task | Static-binary low-assurance detection over label-free Python unit-test views from real-test-derived counterfactual pairs. |
| Scale | 600 paired cases, 1200 static-binary rows, 12 repositories, 125 unique source rows; every pair starts from a real open-source source-control test. |
| Primary transformations | 200 no-assertion, 200 tautological-assertion, and 200 weak-oracle cases. |
| Weak-oracle strata | Field existence, partial string/prefix, length, non-null, shape, and type checks, with 33–34 cases per subtype. |
| Execution evidence | Candidate rows compile and pass; source-control evidence is recorded for every pair. |
| Source-control boundary | 585 row-exact source-control pass pairs and 15 official task-context control pass pairs. |
| Mutation diagnostics | Bounded mutation analysis: 42 pre-specified measured pairs (32 degraded, 10 neutral), with 57 complete source/candidate mutation-pair summaries available across all completed runs. |
| Artifact availability | Supplementary Data 1 provides scoring metadata and selected numerical results; raw artifacts are available on reasonable request subject to third-party permissions and non-redistribution conditions. |

candidate if the transformation drops the system-under-test interaction, keeps the strong oracle as a standalone computation, changes decorators or framework context in a way that affects execution, or mixes multiple primary weakening types. Manual review is used where framework helpers, fixtures, class state, mocks, warnings, or compact structural assertions make syntax-only rewriting unsafe. The objective is not to maximize row count; it is to preserve the pairwise contrast that makes the CI review question meaningful.

The protocol also treats weak tests as an assurance relation rather than a style judgment. A compact type check or shape check can be legitimate when it is the intended oracle. The same check becomes low-assurance in this benchmark when it replaces a stronger paired source-control oracle while keeping the rest of the test plausible. This relation is why pair membership and source-control false-block rate are central to the task: without the source side, a detector can mistake aggressive style preference for useful CI assurance. Figure 2 gives the construction workflow and its admission gates.

Rows are admitted only after deterministic checks and execution evidence. The candidate must parse, compile, and pass. The source-control row must have recorded passing evidence. Pair metadata must preserve the one-source/one- candidate view and a single primary transformation. All main reported tables are computed from one fixed evaluation snapshot, and row edits after snapshot selection are limited to documented replacement repair after QA rejection.

The construction process used a GPT-5.5-assisted workflow under user-supervised review, with deterministic validators enforcing syntax, pairing, execution, and metadata constraints. GPT-5.5 was also used for full-dataset validation and cross-checking before auxiliary model checks were run. Human curation and deterministic validation control final inclusion: LLM output is not used as the ground-truth label. This boundary matters because the benchmark later evaluates auxiliary LLM judges against transformation-defined labels rather than against model-assigned labels.

The review record is stronger than an uninspected generated construction. The study included manual annotation for a 20% slice and random inspection of 30% of AI-assisted construction records. A fixed-seed 60-row review sample covering all primary transformations, all weak-oracle subtypes, and 11 repositories found 60 accepted rows, 0 rejected rows, and 0 replacements. We treat these records as construction QA and admission consistency evidence rather than independent inter-rater or maintainer-severity validation.

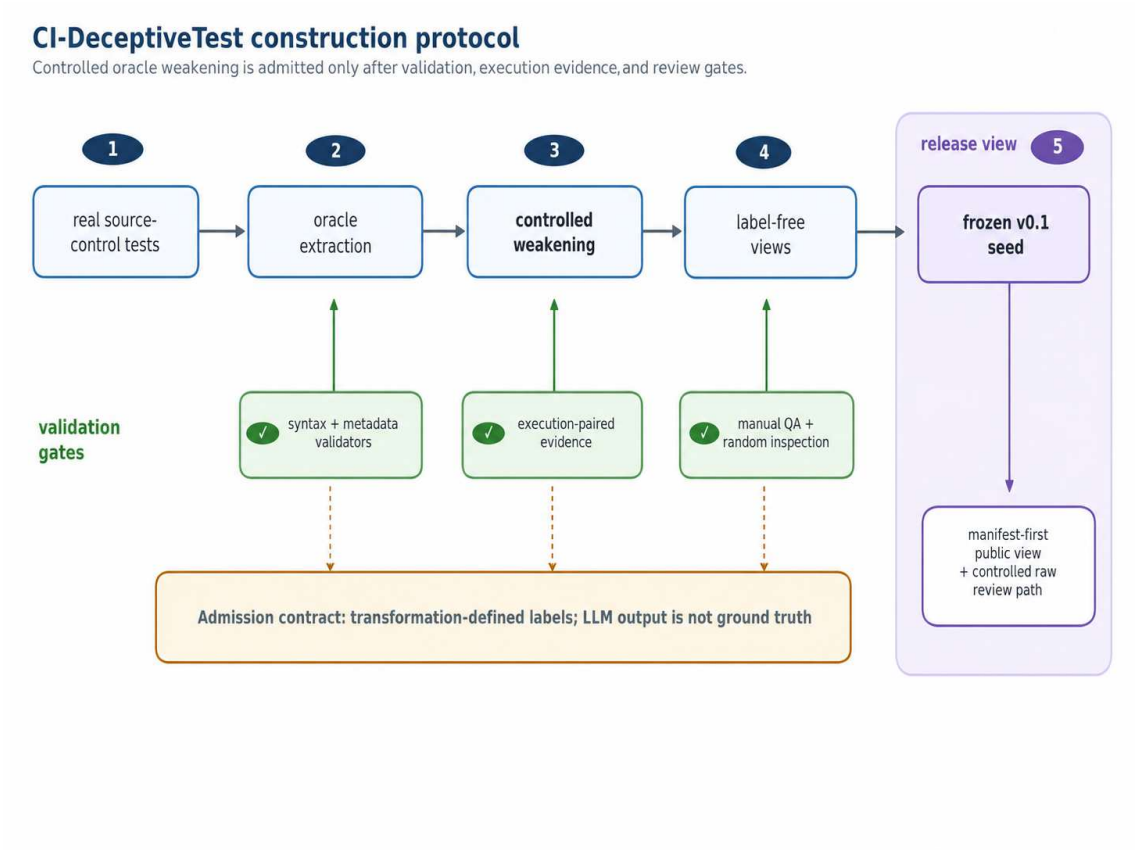



Figure 2: CI-DeceptiveTest construction protocol. Candidate rows are admitted only when a controlled oracle weakening preserves the paired execution context and passes deterministic QA checks.

### 3.3 What the Label Means

The label is transformation-defined rather than a human-consensus severity label. A low-assurance row is weaker than its paired source-control oracle under the protocol; it does not mean that the test is useless in every deployment, nor does it assign blame or intent. A source-control row, in turn, marks the accepted oracle used for paired comparison, not complete semantic adequacy. This is why the benchmark reports source-control false block rate: a detector that rejects compact but legitimate source-control tests is not useful as a CI gate.

### 3.4 Artifact Views

Supplementary Data 1 contains the derived benchmark metadata: pair indexes, view IDs, source metadata, code hashes, answer keys, schemas, scoring examples, and selected numerical results. It also includes scripts to verify the metadata package and recompute selected metrics from saved predictions. Raw source and candidate test code, execution logs, and LLM request bodies follow the access conditions in the Data availability statement. The supplement supports scoring and integrity checks; rerunning detectors on the full test inputs requires access to the underlying raw artifacts.

## 4 Gatekeeper Protocol

CI-DeceptiveTest treats the gatekeeper as a CI protocol rather than a free-form agent. The design is *framework-first, agent-inside*: detectors may use static analysis or an LLM judge,

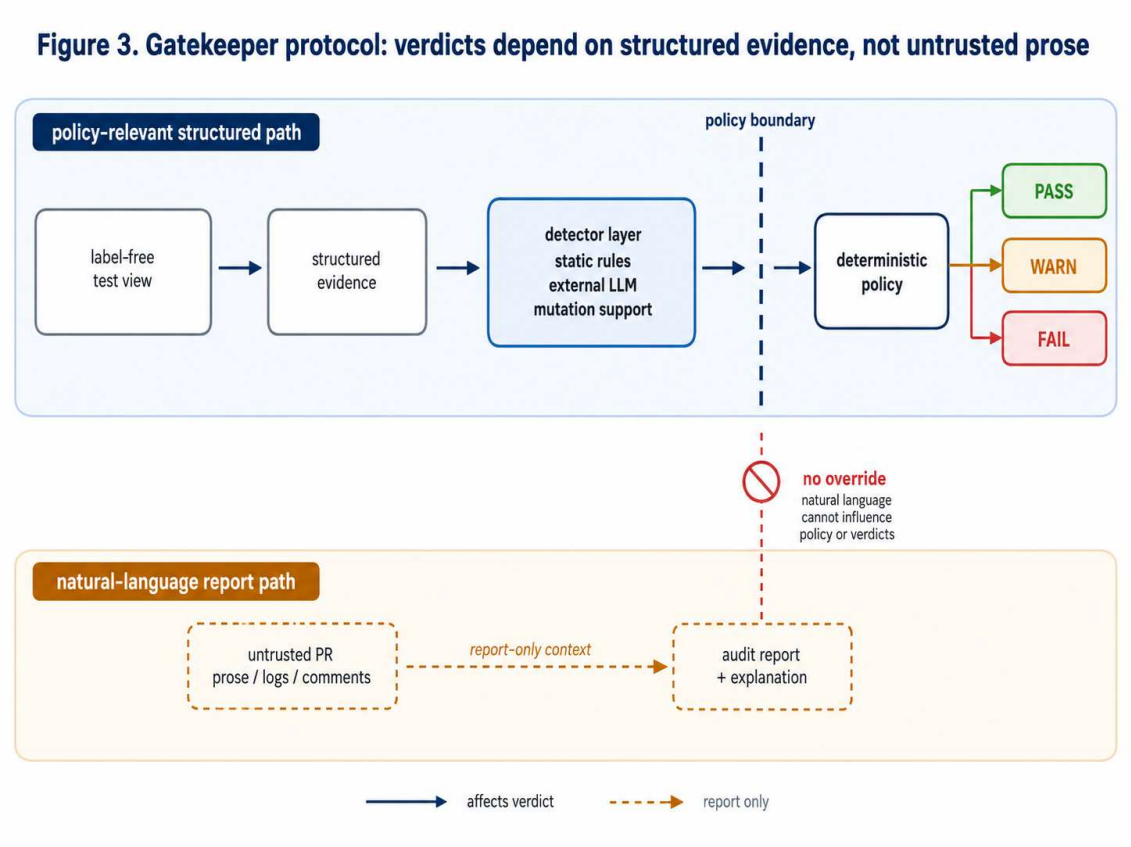



Figure 3: Gatekeeper protocol. Solid arrows denote structured evidence that can affect CI verdicts; dashed arrows denote natural-language context that may appear in explanations but cannot override the deterministic policy boundary.

but the policy that maps evidence to a CI verdict is deterministic and encoded outside untrusted natural language. The LLM is never allowed to rewrite the gate policy, reinterpret the benchmark label space, or become the merge authority.

### 4.1 Separating Evidence, Prediction, and Verdict

The protocol separates three layers that are often blurred in agentic CI. The evidence layer records static structure, execution status, source-control provenance, parse status, and optional mutation evidence. The detector layer consumes the allowed label-free view and emits structured predictions. The verdict layer maps detector outputs to PASS, WARN, or FAIL. This separation is the security and audit boundary: pull-request prose, test logs, generated comments, and natural-language tool outputs may appear in a report, but they must not override structured detector evidence or policy. Figure 3 makes this separation explicit: only structured evidence reaches the policy engine, while untrusted natural language remains report-only context. Table 3 gives the corresponding static-binary contract for inputs, detector outputs, audit records, and verdicts.

### 4.2 Detectors

We evaluate three detector families that play different roles in the protocol. Static Rule is a transparent AST floor that flags missing assertions, tautologies, and weak-oracle-shaped checks such as non-null, type-only, length/shape-only, field-existence, and partial string checks. GPT-5.5 is the primary model-assisted construction and validation model for the benchmark. The

Table 3: Gatekeeper contract for the static-binary workflow.

| Layer | Policy-relevant record | Boundary |
|---|---|---|
| Input | Label-free test view, repository metadata, and artifact hashes. | No answer label, benchmark role, primary transformation, or subtype is exposed to a detector. |
| Detector | Static Rule and/or LLM judge outputs under a JSON/schema contract. | Malformed or missing model outputs are recorded as parse failures and scored by the documented fallback. |
| Audit record | `view_id`, detector booleans, matched rules, parse status, evidence level, model version, and artifact hash. | Reports state the evidence level and do not upgrade static evidence into mutation or security evidence. |
| Verdict | PASS if no detector flags the row; WARN if detectors disagree; FAIL if both flag the row. | The LLM rationale never changes this mapping; projects may choose whether WARN is review-only or blocking. |

follow-up `mimo-v2.5-pro` LLM judge is an auxiliary external check that uses the same model-agnostic task interface: the request pack contains the label-free test view and asks the model to return a structured binary label. The reported auxiliary LLM results are external-judge evidence over the GPT-5.5-assisted benchmark, not a cross-provider generality claim. AST Feature Logistic is a supervised code-shape stress test; it is not a proposed deployment detector, but it measures how much of the benchmark can be solved through local syntax features.

### 4.3 Policy Projections

The PASS/WARN/FAIL verdict is human-facing, while the paper reports two binary projections for benchmark scoring. In `fail_only`, only FAIL is scored as low assurance; this models conservative merge blocking. In `warn_as_low`, WARN and FAIL are scored as low assurance; this models audit-sensitive review. These policies are not universal thresholds. They make the central CI trade-off explicit: lower false blocks versus fewer missed low-assurance tests.

### 4.4 Operational CI Example

In a concrete CI run, a pull request adds or edits a Python test. The Gatekeeper hashes the label-free test view, runs the selected detectors, validates structured outputs, and applies the fixed verdict policy. If both detectors flag the row, the required CI check can emit a blocking FAIL. If only one detector flags the row, the check emits WARN and records the disagreement for review. If neither detector flags the row, it emits PASS. The important constraint is that the LLM rationale never changes this mapping; untrusted PR text, logs, and tool output cannot rewrite policy.

## 5 Evaluation Methodology

### 5.1 Research Questions

The evaluation follows the benchmark's central tension: a useful gatekeeper must catch tests that no longer constrain behavior without turning every compact source-control test into a false alarm. We therefore evaluate both sides of the CI decision. Low-assurance recall measures whether weak candidates are caught; source-control false block rate measures whether the gate would punish adequate project tests.

**RQ1:** How far do transparent static smell rules go, and how much do they overblock source-control tests?

**RQ2:** Does a full-dataset LLM judge improve the false-block trade-off, and where does it fail?

**RQ3:** Which low-assurance transformations remain difficult, especially among weak oracles?

**RQ4:** What do Gatekeeper policy projections, bounded mutation diagnostics, and supervised AST-feature stress tests reveal about operating points, evidence strength, and shortcut risk?

All main reported results are computed from a fixed evaluation snapshot. The snapshot is selected before scoring the reported detectors, so later QA activity cannot change the rows used in the main tables.

### 5.2 Methods

**Static Rule** is the transparent full-dataset floor. It parses Python test code and applies auditable AST rules for no assertion, tautological assertion, and weak-oracle-shaped checks. The baseline is intentionally simple: it represents the kind of rule set a maintainer could inspect, not the strongest possible static analyzer.

**GPT-5.5 validation and auxiliary LLM judge checks** are separated by role. GPT-5.5 is the primary model-assisted workflow used for construction, review, and full-dataset validation. The detector-facing request pack is model-agnostic and keeps the answer key outside the prompt. After the GPT-5.5 validation pass, we run `mimo-v2.5-pro` on all 1200 rows as an auxiliary external judge. This auxiliary run scores an external judge against a GPT-5.5-assisted benchmark; it does not rank foundation models or establish cross-provider generality. The provided request pack can be rerun with other providers.

**Gatekeeper auxiliary-judge+Static** combines Static Rule and `mimo-v2.5-pro` auxiliary predictions into PASS/WARN/FAIL verdicts, then scores two binary projections: fail-only and warn-as-low.

**Mutation and AST diagnostics** are supporting analyses rather than headline full-dataset detectors. The bounded mutation analysis reports source-first mutation evidence and a paired mutation-threshold diagnostic only for measured pairs. The pre-specified mutation subset contains 42 complete source/candidate mutation pairs; additional completed measurements bring the available mutation record to 57 complete mutation-pair summaries. We report these measurements as supplementary evidence, not as a full-dataset mutation benchmark. AST Feature Logistic is a supervised non-LLM stress test using local code-shape features under pair-grouped, repository-held-out, and leave-primary-transformation-out splits. It tests shortcut risk, not semantic oracle understanding.

**Security-facing checks** are supplementary boundary diagnostics rather than RQ4 primary evidence. The policy-isolation stress protocol checks that synthetic untrusted PR, log, and tool-output payloads do not change a fixed verdict function. Separate LLM-judge input-sensitivity probes inject adversarial comments or string literals into test code. These analyses support the policy-boundary claim in this benchmark setting; end-to-end multi-turn CI agent security with secrets, repository writes, and merge permissions is a separate deployment study.

### 5.3 Metrics and Statistical Reporting

All main metrics are computed from the same binary confusion matrix, with `low_assurance` as the positive class. A true positive denotes a `low_assurance` row predicted as low assurance; a false positive is a source-control row predicted as low assurance; a false negative is a low-assurance row predicted as source control; and a true negative is a source-control row predicted

as source control. Macro F1 is the unweighted mean of the two per-class F1 scores. Source-control false block rate is $\mathrm{FP}/(\mathrm{FP}+\mathrm{TN})$, and is central because an over-conservative gate can look effective while blocking many adequate tests.

Full-dataset results over 1200 rows are treated as census-style deterministic evaluations of the fixed benchmark snapshot. For paired method comparisons, we report exact two-sided sign tests over rows where methods differ in correctness. For bounded 300-row analyses, bootstrap resampling never regenerates rows, reruns the LLM, or changes the answer key; it resamples fixed predictions only.

Transport errors, missing outputs, malformed JSON, and labels outside the allowed schema are parse failures and are assigned `source_control_static` before metric computation. This fallback is conservative for false blocking but can reduce measured low-assurance recall. HTTP 429 retries affect only output collection; after collection, scoring uses persisted outputs and answer keys.

### 5.4 Operational Scope

The GPT-5.5 workflow is the primary model-assisted construction and validation path. The auxiliary `mimo-v2.5-pro` run completed 1200 rows with 471,900 total tokens, 0 transport errors, 6 parse failures, and a timestamp span of 75.6 minutes. Using the MiMo pay-as-you-go prices observed on 2026-06-14 for `mimo-v2.5-pro` [29], this corresponds to an estimated CNY 1.65 domestic run or USD 0.24 overseas run. This is a pay-as-you-go equivalent estimate for the auxiliary check, not a Token Plan bill, and it is not a controlled CI latency benchmark. A practical deployment would use a tiered CI policy: static checks on every PR, LLM judging for high-risk or WARN cases, and mutation only as targeted follow-up.

## 6 Results

Across the fixed full-dataset evaluation, the central result is not that one detector wins. Low-assurance test review exposes two failure modes that ordinary CI tends to collapse together. Transparent smell rules catch many weakened tests, but they also reject many adequate source-control tests. GPT-5.5 is the primary model-assisted construction and validation path; the follow-up `mimo-v2.5-pro` auxiliary judge is more selective and sharply reduces false blocking, but it accepts a non-trivial fraction of subtle low-assurance candidates. The Gatekeeper result is therefore an operating-point story: CI policy must decide where to spend review effort, not merely choose a single universal threshold.

### 6.1 Static Rules and LLM Judging Expose Opposite Failure Modes

Table 4 gives the main full-dataset results. Static Rule reaches 0.928 low-assurance recall, but the same rule set has a 0.458 source-control false block rate: it would block 275 of the 600 source-control rows, or 45.8 false blocks per 100 source-control checks. The classic no-assertion-or-tautology rule family is less aggressive, with 0.663 recall and 0.193 FBR. The additional weak-oracle surface rules buy recall, but the payment is overblocking. This makes the static baseline valuable as an auditable floor, not as a standalone merge gate.

The GPT-5.5-first model-assisted workflow is followed by a `mimo-v2.5-pro` auxiliary judge to test whether the same label-free interface yields a lower false-block operating point. On all 1200 rows, the auxiliary judge reaches 0.885 macro F1 and reduces FBR from 0.458 to 0.048, while low-assurance recall falls from 0.928 to 0.820. The paired disagreement is large enough to matter operationally: both methods are correct on 772 rows, both are wrong on 27 rows, Static Rule is uniquely correct on 110 rows, and the auxiliary judge is uniquely correct on 291 rows. An exact sign test over the 401 discordant rows gives $p = 5.70 \times 10^{-20}$. The takeaway is not that the LLM judge is a perfect gate. It is that transparent static evidence and contextual LLM

Table 4: Full-dataset detector and policy results on 1200 static-binary rows. FBR is source-control false block rate.

| Method or policy | Macro F1 | Low rec. | Low prec. | FBR | CI implication |
|---|---|---|---|---|---|
| Static Rule | 0.725 | 0.928 | 0.669 | 0.458 | High-recall transparent floor; unsafe as a blocking gate. |
| GPT-5.5-assisted + `mimo-v2.5-pro` auxiliary judge | 0.885 | 0.820 | 0.944 | 0.048 | Low false-block auxiliary judge over GPT-5.5-validated benchmark. |
| Gatekeeper `fail_only` | 0.877 | 0.775 | – | 0.018 | Conservative blocking: few false blocks, more missed weak tests. |
| Gatekeeper `warn_as_low` | 0.728 | 0.973 | – | 0.488 | Audit-sensitive review: catches most weak tests, creates high review load. |

Table 5: LLM-judge performance by transformation slice and selected weak-oracle subtype.

| Slice | Rows | Macro F1 | Low rec. | FBR | Diagnostic reading |
|---|---|---|---|---|---|
| `no_assertion` | 400 | 0.902 | 0.845 | 0.040 | Often visible, but not always recognized as low assurance. |
| `tautological_assertion` | 400 | 0.967 | 0.985 | 0.050 | Mostly solved by the current judge. |
| `weak_oracle` | 400 | 0.782 | 0.630 | 0.055 | Central unresolved slice. |
| `partial_string_or_prefix_only` | 68 | 0.665 | 0.412 | 0.029 | Most subtle observed weak-oracle miss pattern. |
| `length_only` | 66 | 0.735 | 0.576 | 0.091 | Coarse structural check often accepted as plausible. |
| `type_only` | 66 | 0.863 | 0.818 | 0.091 | Easier, but still not a semantic oracle. |

evidence make different mistakes. Once source-control false-block risk is measured explicitly, the meaning of "better" changes.

The auxiliary run is not a model leaderboard. It is included to show that the GPT-5.5-assisted benchmark can be scored by a separate model-facing judge interface while keeping answer keys outside the prompt. Parse failures do not explain the operating point: `mimo-v2.5-pro` has 6 parse failures and a worst-case macro-F1 swing of 0.004.

### 6.2 Weak Oracles Are the Central Stress Case

The main error pattern is not the easy case of a missing assertion. Table 5 shows that the GPT-5.5-first workflow followed by `mimo-v2.5-pro` auxiliary judging performs best on tautological assertions and worst on weak oracles. In the weak-oracle slice, low-assurance recall is only 0.630. The hardest observed subtype is partial-string/prefix-only, with recall 0.412; length-only rows have recall 0.576. These subtype comparisons are exploratory fixed-slice diagnostics, not definitive subtype hardness rankings, because each subtype currently has only 33–34 low-assurance rows. Weak-oracle recall is the benchmark stress signal: when a detector accepts a plausible coarse oracle, it may also accept a test that no longer checks the source-control value relation.

Disagreement inspection explains why this slice matters. Static rules often catch syntactic no-assertion and prefix-only candidates that the LLM accepts. The LLM, in turn, preserves many compact but legitimate source-control tests that static rules overblock. Both methods are strained when a source-control test itself uses a narrow but meaningful structural oracle. This is why source-control false-block risk is a first-class metric: without it, a detector can look strong by rejecting too many adequate tests.

### 6.3 Gatekeeper Policies Make the Trade-off Explicit

The PASS/WARN/FAIL layer converts detector disagreement into an auditable CI decision surface. In `fail_only` mode, the Gatekeeper keeps FBR at 0.018 but misses more low-assurance rows. In `warn_as_low` mode, low-assurance recall rises to 0.973 but FBR rises to 0.488. The verdict counts make the trade-off concrete: FAIL is the blocking CI outcome and WARN is a review outcome. Across all rows, the Gatekeeper emits 16 low-assurance PASS rows, 119 low-assurance WARN rows, 465 low-assurance FAIL rows, 11 source-control FAIL rows, and 282 source-control WARN rows. The large `warn_as_low` false-block rate should therefore be read as a review-load projection, not automatically as a hard merge-block projection.

This is the practical value of exposing policy separately from detection. Different CI settings can choose different operating points without changing the detector evidence: a release-gating setting can route WARN to review, whereas a high-throughput setting can block only agreement FAILs. The policy-cost analysis over fixed confusion matrices reaches the same conclusion: balanced review favors the LLM-only operating point, reviewer-scarce CI favors `fail_only`, and release-gate profiles that heavily penalize missed low-assurance tests favor `warn_as_low`. These are not measured human costs or deployment prescriptions. They make the false blocks versus missed low-assurance tests trade-off explicit.

### 6.4 Supporting Diagnostics Bound the Claim

Table 6 reports diagnostics that sharpen the interpretation of the main results without expanding the scope of the claim. The bounded mutation analysis supports the oracle-weakening interpretation on measured rows: 42 pre-specified pairs have complete source/candidate mutation measurements, 32 candidates have lower mutation scores, and the measured degradation rate is 0.762. Additional completed mutation measurements bring the total completed source/candidate mutation record to 57 measured pairs, with 42 degraded candidates and 15 neutral candidates. This addresses the scale of the mutation evidence more directly than the pre-specified subset alone, but it remains secondary evidence, not a full mutation benchmark. Unmeasured rows would require source-first mutation execution before candidate mutation can be interpreted.

The AST Feature Logistic stress test shows the complementary risk: detectors can exploit benchmark shape. It reaches 0.947 macro F1 under pair-grouped cross validation and 0.935 under leave-one-repo-out, but drops to 0.812 when the primary transformation family is held out. The largest feature weights are exactly the shallow signals a benchmark should expose: assertion-count differences, expression-statement residue, literal true assertions, and equality comparisons. This is not evidence of semantic oracle understanding; it is a shortcut diagnostic that future benchmark versions should make harder.

Repository-level checks reduce, but do not eliminate, source-mix concern. The largest repository contributes 0.140 of rows. Removing any single repository changes the `mimo-v2.5-pro` auxiliary macro F1 by at most 0.005, low-assurance recall by at most 0.009, and FBR by at most 0.009. This is a source-stratified diagnostic, not evidence of non-Python generality.

## 7 Discussion and Threats to Validity

### 7.1 Implication: CI Needs Assurance Gates, Not Just Test Counts

The main implication is methodological. In AI-saturated development, producing test-shaped code is becoming cheap; establishing that the test constrains behavior remains expensive. A CI system that rewards passing tests, line coverage, or the mere presence of an assertion can be misled by low-assurance tests that continue to execute realistic code while weakening the oracle.

Table 6: Supporting diagnostics and claim boundaries.

| Diagnostic | Main observation | Boundary |
|---|---|---|
| Mutation analysis | 42 pre-specified measured pairs; 32 degraded; 57 complete mutation-pair summaries across all completed runs, with 42 degraded and 15 neutral. | Secondary diagnostic, not a full mutation benchmark over 600 pairs. |
| Weak-oracle measured subset | 16 of 23 measured weak-oracle candidates degraded relative to source controls. | Subtype cells range from 1–8 pairs; not a full weak-oracle mutation benchmark. |
| AST Feature Logistic | Macro F1 0.947 pair-grouped, 0.935 leave-one-repo-out, 0.812 leave-primary-transform-out. | Stress test for code-shape shortcuts, not semantic oracle understanding. |
| LLM input sensitivity | Injected comments alter 15/60 previously correct rows (ASR 0.250); hardening lowers ASR to 0.183; comment sanitization lowers ASR to 0.083; string-literal injection remains ASR 0.250. | Bounded input-sensitivity probe, not a deployed CI-agent security result. |

CI-DeceptiveTest makes that failure mode measurable by pairing every weakened candidate with a source-control oracle and by reporting false blocking on the source-control side.

This framing also explains why a single detector is unlikely to be enough. Static rules are transparent and high recall, but overblock compact legitimate tests. LLM judges are more context-sensitive, but they accept many weak oracles. Mutation testing is stronger evidence when it is available, but it is too expensive to treat as a first-line check for every row. A practical Gatekeeper should therefore be staged: cheap static checks first, LLM judging for ambiguous tests, WARN routing for weak-oracle-shaped cases, and targeted mutation or reviewer follow-up only when the additional evidence is worth the cost.

### 7.2 Why Weak Oracles Are the Hard Case

Weak oracles are the benchmark's stress signal because they sit closest to ordinary review ambiguity. No-assertion and tautological tests matter, but they are often syntactically visible. A non-null, type, length, shape, field-existence, or prefix check may be a useful smoke test in one context and a low-assurance replacement in another. The paired protocol turns that ambiguity into a testable question: does the candidate preserve the behavioral assurance of the source-control oracle it resembles?

The weak-oracle result should be read as a stress finding rather than a solved task. The current subtype comparisons identify where this benchmark is hardest, but they are fixed-slice diagnostics rather than definitive subtype rankings. The design lesson is already useful: benchmark growth should prioritize plausible weak assertions and paired dynamic evidence, not merely add more obvious no-assertion cases.

### 7.3 Scope of Current Claims

The current claims are intentionally scoped. CI-DeceptiveTest is a Python execution-paired benchmark constructed, reviewed, and validated through a GPT-5.5-assisted workflow with deterministic validation. The reported `mimo-v2.5-pro` judge is a follow-up auxiliary external check over the same label-free request pack, not the dataset construction authority. Labels are transformation-defined, execution-backed, and manually reviewed through the construction QA process, including a 20% manually annotated slice and 30% random inspection of AI-assisted construction records. The independent review packet and scoring rubric are protocol-ready artifact materials. Mutation evidence is used as bounded support: the pre-specified mutation subset contains 42 complete paired measurements, and all completed mutation measurements contain 57 complete source/candidate mutation-pair summaries.

These choices define what the result is good for: a reusable benchmark and protocol for studying low-assurance test detection in AI-assisted Python unit test review. Broader language coverage, provider coverage, maintainer studies, larger mutation overlays, and deployed CI-agent evaluations are natural extensions of the same paired protocol rather than prerequisites for the current benchmark contribution.

### 7.4 Construct Validity

The main construct boundary is counterfactual realism. The low-assurance rows are not hand-written toy tests: they are controlled counterfactuals derived from real source-control tests. This design prioritizes label clarity and paired causal contrast over ecosystem prevalence estimation. It shows whether a detector recognizes oracle degradation under a paired real-test context, while questions about how frequently such patterns arise in pull-request histories belong to a complementary prevalence study. Other naturally occurring agent failure modes, such as flakiness, environment-dependent tests, target mocking, skip/xfail abuse, and tests that alter production code to satisfy weak assertions, can be incorporated by extending the same real-test-derived paired protocol.

Another construct threat is that source-control tests are not perfect semantic specifications. They can be compact, project-specific, or incomplete. The benchmark therefore measures degradation relative to the paired source-control oracle, not absolute semantic completeness. The benchmark includes user-supervised manual annotation and random inspection for construction QA. A follow-on reviewer study should separate obvious no-assertion cases from weak-oracle cases and include framework idioms such as fixtures, helper assertions, mocking, monkeypatching, expected warnings, and class-level setup. The current framework-idiom audit records coverage of these patterns as construction QA, while preserving transformation-defined labels.

The benchmark should therefore be compared with real-bug and repository benchmarks by purpose, not by claim type. Defects4J, BugsInPy, and SWE-bench support controlled studies over real faults or issue-resolution tasks [11, 28, 10]. CI-DeceptiveTest instead controls the test side: it preserves the accepted test's repository setting while weakening its oracle. This makes the labels cleaner for detector evaluation and keeps the source-control false-block side visible. Estimating deployed prevalence is a separate empirical question, not the role of this benchmark.

### 7.5 Internal and External Validity

Internal validity depends on label separation and artifact consistency. The label-free input view hides benchmark role, primary transformation, and subtype; answer-only fields are kept outside model prompts. Generated tables, scoring scripts, artifact checks, and consistency checks reduce transcription and join risk, but they cannot make scoring bugs impossible. Model-output failures are handled by a single fallback policy, and parse-failure impact is reported rather than silently discarded.

External validity is intentionally limited. The JavaScript adapter smoke uses six curated JavaScript pairs and 12 static binary rows; it is a protocol-portability note, not non-Python validation or cross-language generality evidence. A Java or JavaScript benchmark version would require source-control packaging, mutation tooling, framework-specific assertion idioms, and language-specific false-block analysis. Similarly, broader provider claims require rerunning the judge request pack with additional model families beyond the GPT-5.5 primary workflow and the `mimo-v2.5-pro` auxiliary check. The current auxiliary model results should be read as operating-point evidence for this benchmark, not as a foundation-model ranking.

The same scoping applies to security language. Prompt-injection guidance and agent benchmarks motivate the Gatekeeper's separation between untrusted natural language and structured policy [20, 24, 32, 4]. The current evidence supports this policy-boundary design in the benchmark setting. End-to-end robustness of a deployed multi-turn CI agent with secrets, write

permissions, or merge authority requires a separate security evaluation.

### 7.6 Ethics, Data Use, and Release Boundary

CI-DeceptiveTest is derived from open-source software tests and does not include a human-subject study. Its labels are not claims about developer motives. Synthetic prompt-injection payloads are generated for stress testing and do not contain real credentials. The main dual-use risk is that weakened oracle examples could teach users how to create tests that appear stricter than they are; this is the dual-use risk of weakened-oracle examples. We mitigate this by framing examples as low-assurance anti-patterns, separating public scoring metadata from raw upstream test text, including a dataset card and misuse guidance, and using controlled raw-text review when redistribution rights are unclear. The metadata package in Supplementary Data 1 is not a broad raw-data redistribution and does not replace upstream license review.

### 7.7 Roadmap

The benchmark expansion and power roadmap is deliberately separated from the current claims. Future versions should add a license-cleared raw-text subset, larger weak-oracle subtype strata, one non-Python execution-paired slice, cross-provider judge reruns, larger mutation overlays, and maintainer-facing review studies. Security work should become a separate CI-agent suite with multi-turn tool traces, realistic permission boundaries, and explicit secret/write/merge controls. Power analysis for model-delta claims should be reported before small differences between judges are interpreted as substantive.

## 8 Conclusion

AI-assisted development makes it easy to produce passing tests. It does not make those tests automatically assurance-bearing. CI-DeceptiveTest targets this gap with a Python execution-paired benchmark and Gatekeeper protocol for detecting low-assurance tests in LLM-assisted development. The evaluated benchmark instance contains 600 paired cases and 1200 static-binary rows derived from real open-source Python tests through controlled oracle weakening. Its contribution is a real-test-derived counterfactual measurement protocol for detector evaluation, with prevalence estimation left as a complementary empirical question.

The evidence shows why the problem deserves a dedicated benchmark. Static rules catch many low-assurance rows but falsely block too many source-control tests. GPT-5.5 is the primary construction, review, and full-dataset validation model; the follow-up `mimo-v2.5-pro` auxiliary judge reduces false blocking from 0.458 to 0.048 and reaches 0.885 macro F1, but the weak-oracle result is the central stress signal in the current benchmark. Weak-oracle detection remains the primary hard case: plausible coarse assertions are difficult because they can execute realistic behavior while dropping the source oracle's semantic constraint. The bounded mutation inventory strengthens this interpretation on measured rows, while also showing why the benchmark should not be overclaimed as a full mutation benchmark. Subtype-level weak-oracle patterns should remain exploratory until larger strata are added.

The broader lesson is that CI should evaluate the behavioral assurance that tests preserve, not merely whether tests pass. CI-DeceptiveTest provides a reproducible benchmark instance, evidence protocol, and baseline package for that question. Future work should reuse the paired protocol across languages, model providers, mutation overlays, prompt-injection threat suites, and maintainer-facing review studies before making ecosystem-wide or production CI claims.

## Declarations

**Funding.** The author received no specific funding for this work.
**Conflict of interest.** The author declares no conflict of interest.
**Data availability.** The derived CI-DeceptiveTest benchmark metadata, pair indexes, answer keys, code hashes, execution-log manifests, and selected numerical evaluation results are included in Supplementary Data 1. The underlying source and transformed test code, raw execution logs, and raw LLM request bodies are not publicly available because clearance for redistribution of third-party source material and raw logs has not been completed. These materials are available from the corresponding author, Qining Li (castelu@qq.com), on reasonable request for research or review, subject to applicable third-party permissions and non-redistribution conditions.
**Code availability.** The metadata-scoring and verification scripts are included in Supplementary Data 1. The full experiment and reproduction code is available from the corresponding author on reasonable request under the same access conditions.
**Author contributions.** Qining Li is the sole author and is responsible for the study conception, dataset construction protocol, experimental design, implementation, analysis, and manuscript preparation.

## A Dataset Construction and Review Protocol

The construction protocol follows a source-control-first rule. A candidate is accepted only when it preserves the behavior-bearing call path of a real source-control test while weakening the oracle. Rejection criteria include dropping the system-under-test interaction, retaining the original strong oracle as a standalone computation, changing framework context, or mixing multiple primary weakening types. GPT-5.5 supported candidate construction, review, and cross-checking, but model outputs were not label authorities. The final admission decision used deterministic validation, execution evidence, and human-supervised review under the fixed taxonomy.

The construction log records user-supervised manual annotation and inspection: a 20% manually annotated slice, random inspection of 30% of AI-assisted construction records, larger review files, and GPT-5.5-assisted cross-checks. The fixed-seed 60-row review sample covered all three primary transformations, all six weak-oracle subtypes, and 11 repositories. It found 60 accepted rows, 0 rejected rows, and 0 replacements. This is an admission-consistency audit. A blinded review packet and scoring rubric are retained as protocol materials in the internal experiment archive and can be requested under the stated access conditions; the reported labels remain transformation-defined.

## B Reproduction Entry Points

The available materials support three levels of reproduction:

1. Metadata and score checks using Supplementary Data 1, which includes pair indexes, answer keys, code hashes, execution-log manifests, saved predictions for selected baselines, and scoring and verification scripts.

2. Full experiment review using the internal archive, available from the corresponding author on reasonable request under the stated access conditions.

3. Raw-text and execution review using the paired source and candidate tests and logs, subject to applicable third-party permissions and non-redistribution conditions.

The supplement supports reproduction of selected reported scores from saved predictions. It does not include the raw test inputs needed to rerun detectors or execution experiments, and no unrestricted raw-data release is claimed.

## C Claim-to-Artifact Traceability

Each quantitative claim in the main text is linked to a generated result table, benchmark manifest, or scoring report in the internal experiment archive. Selected scoring reports and metadata are included in Supplementary Data 1; additional artifacts can be requested under the stated access conditions. The trace covers dataset profile counts, construction review, full-dataset detector metrics, weak-oracle slice analysis, policy projections, mutation diagnostics, model-output parse failures, and security-boundary checks. The manuscript reports the consolidated evidence needed for the paper's argument instead of embedding every audit table in the PDF.

## D Supplementary Analysis Tables

Additional analysis tables retained in the internal experiment archive include repository distribution, source-control evidence distribution, construction-review summary, binary task scope audit, independent-review packet audit, framework-idiom audit, static-rule ablation, false-block taxonomies, practitioner burden projections, auxiliary-judge sensitivity, parse-failure impact, repository sensitivity, weak-oracle operational risk, mutation subset representativeness, mutation run-log summaries, local CI replay smoke tests, permission traces, and LLM input-sensitivity tables. The mutation summaries record 57 complete source/candidate mutation-pair measurements, including 42 measurements in the pre-specified subset and 15 additional completed measurements. These materials can be requested under the stated access conditions. Selected mutation summaries are also included in Supplementary Data 1. These records do not turn the study into a full mutation benchmark over all 600 pairs.

## References


[1] Nadia Alshahwan, Jubin Chheda, Anastasia Finegenova, Beliz Gokkaya, Mark Harman, Inna Harper, Alexandru Marginean, Shubho Sengupta, and Eddy Wang. Automated unit test improvement using large language models at Meta. In *Proceedings of the ACM on Software Engineering*, 2024.

[2] Cristian Augusto, Antonia Bertolino, Guglielmo De Angelis, Francesca Lonetti, and Jesús Morán. Large language models for software testing: A research roadmap. *arXiv preprint arXiv:2509.25043*, 2025.

[3] Yinghao Chen, Zehao Hu, Chen Zhi, Junxiao Han, Shuiguang Deng, and Jianwei Yin. ChatUniTest: A framework for LLM-based test generation. In *Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering Demonstrations Track*, 2024.

[4] Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramer. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

[5] Christopher Foster, Abhishek Gulati, Mark Harman, Inna Harper, Ke Mao, Jillian Ritchey, Herve Robert, and Shubho Sengupta. Mutation-guided LLM-based test generation at Meta. *arXiv preprint arXiv:2501.12862*, 2025.

[6] Gordon Fraser and Andrea Arcuri. EvoSuite: Automatic test suite generation for object-oriented software. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, pages 416–419, 2011.

[7] Dong Huang, Jie M. Zhang, Mark Harman, Qianru Zhang, Mingzhe Du, and See-Kiong Ng. Benchmarking LLMs for unit test generation from real-world functions. *arXiv preprint arXiv:2508.00408*, 2025.

[8] Kush Jain, Gabriel Synnaeve, and Baptiste Roziere. TestGenEval: A real world unit test generation and test completion benchmark. In *International Conference on Learning Representations*, 2025.

[9] Yue Jia and Mark Harman. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, 37(5):649–678, 2011.

[10] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations*, 2024.

[11] René Just, Darioush Jalali, and Michael D. Ernst. Defects4J: A database of existing faults to enable controlled testing studies for java programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pages 437–440, 2014.

[12] Sungmin Kang, Juyeon Yoon, and Shin Yoo. Large language models are few-shot testers: Exploring LLM-based general bug reproduction. In *Proceedings of the IEEE/ACM International Conference on Software Engineering*, 2023.

[13] Caroline Lemieux, Jeevana Priya Inala, Shuvendu K. Lahiri, and Siddhartha Sen. CodaMOSA: Escaping coverage plateaus in test generation with pre-trained large language models. In *Proceedings of the 45th IEEE/ACM International Conference on Software Engineering*, pages 919–931, 2023.

[14] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, 2023.

[15] Junwei Liu, Kaixin Wang, Yixuan Chen, Xin Peng, Zhenpeng Chen, Lingming Zhang, and Yiling Lou. Large language model-based agents for software engineering: A survey. *ACM Transactions on Software Engineering and Methodology*, 2026.

[16] Yujian Liu, Jiabao Ji, Yang Zhang, Wenbo Guo, Tommi Jaakkola, and Shiyu Chang. HarnessLLM: Automatic testing harness generation via reinforcement learning. *arXiv preprint arXiv:2511.01104*, 2025.

[17] Stephan Lukasczyk and Gordon Fraser. Pynguin: Automated unit test generation for python. In *44th IEEE/ACM International Conference on Software Engineering: Companion Proceedings*, pages 168–172, 2022.

[18] Stephan Lukasczyk, Florian Kroiß, and Gordon Fraser. An empirical study of automated unit test generation for python. *Empirical Software Engineering*, 28(2):36, 2023.

[19] Arghavan Moradi Dakhel, Amin Nikanjam, Vahid Majdinasab, Foutse Khomh, and Michel C. Desmarais. Effective test generation using pre-trained large language models and mutation testing. *arXiv preprint arXiv:2308.16557*, 2023.

[20] OWASP Gen AI Security Project. LLM01:2025 prompt injection, 2025.

[21] Juan Altmayer Pizzorno and Emery D. Berger. CoverUp: Coverage-guided LLM-based test generation. *arXiv preprint arXiv:2403.16218*, 2024.

[22] Max Schäfer, Sarah Nadi, Aryaz Eghbali, and Frank Tip. An empirical evaluation of using large language models for automated unit test generation. *IEEE Transactions on Software Engineering*, 50(1):85–105, 2024.

[23] Test Smells. Software unit test smells, 2026.

[24] UK National Cyber Security Centre. Prompt injection is not SQL injection (it may be worse), 2025.

[25] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. Software testing with large language models: Survey, landscape, and vision. *IEEE Transactions on Software Engineering*, 2024.

[26] Wenhan Wang, Chenyuan Yang, Zhijie Wang, Yuheng Huang, Zhaoyang Chu, Da Song, Lingming Zhang, An Ran Chen, and Lei Ma. TestEval: Benchmarking large language models for test case generation. In *Findings of the Association for Computational Linguistics: NAACL*, 2025.

[27] Yibo Wang, Congying Xia, Wenting Zhao, Jiangshu Du, Chunyu Miao, Zhongfen Deng, Philip S. Yu, and Chen Xing. MultiFileTest: A multi-file-level LLM unit test generation benchmark and impact of error fixing mechanisms. *arXiv preprint arXiv:2502.06556*, 2026.

[28] Ratnadira Widyasari, Sheng Qin Sim, Camellia Lok, Haodi Qi, Jack Phan, Qijin Tay, Constance Tan, Fiona Wee, Jodie Ethelda Tan, Yuheng Yieh, Brian Goh, Ferdian Thung, Hong Jin Kang, Thong Hoang, David Lo, and Eng Lieh Ouh. BugsInPy: A database of existing bugs in python programs to enable controlled testing and debugging studies. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1556–1560, 2020.

[29] Xiaomi MiMo. Xiaomi MiMo API pricing, 2026. Accessed 2026-06-14; page updated 2026-06-12.

[30] Qiang Yu, Xinran Cheng, and Chuanyi Liu. Defense against indirect prompt injection via tool result parsing. *arXiv preprint arXiv:2601.04795*, 2026.

[31] Zhiqiang Yuan, Yiling Lou, Mingwei Liu, Shiji Ding, Kaixin Wang, Yixuan Chen, and Xin Peng. Evaluating and improving ChatGPT for unit test generation. In *Proceedings of the ACM on Software Engineering*, 2024.

[32] Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics: ACL*, 2024.

## Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- SupplementaryData1.zip