Trust the Target, Not the Swarm?

A Small Pre-Registered Pilot on Mechanical Gates and LLM Review, and the Reference They Share

Anonymous Author(s)
Submitted to MAS-GAIN 2026 for double-blind review

***Abstract*—Agentic software engineering often adds review agents (panels, debate, critics, votes) over an open target. An alternative is to close the target so that many errors become mechanical events (a parse, test, or differential failure) rather than a reviewer's judgment. We ran a small, pre-registered pilot on a public substrate (EvalPlus HumanEval+) to compare the two. Over 25 reviewed seeded defects, mechanical gates and a cross-vendor LLM reviewer catch defects about equally ($25/25$ for the gates, $25/25$ and $24/25$ for two review baselines); the gates also cost no model calls. Reviewers false-rejected 3–5 of 40 correct programs while the gates false-rejected none, but that last number is largely structural, since the gates judge against the same canonical solutions the correct controls are drawn from, and we do not lean on it. The substance is the per-case adjudication of the eight false-rejection events. Three are genuine reviewer misreads of a clear specification; four are the reviewer flagging a reference or specification that is itself buggy, self-contradictory, or ambiguous; one is oracle-uncertain. So the disagreements we saw split between review misreading clear specs and review catching defects in the reference that a reference-agreement gate cannot see. At this scale we read this as suggestive of complementary failure modes rather than established. A concrete by-product is a handful of candidate defects in a widely used benchmark's references and specifications. We release the pre-registration, harness, and per-case adjudication.**

***Index Terms*—agentic software engineering, LLM code review, differential testing, benchmark oracle validity, pre-registration**

I. Introduction

Much recent work on agentic software engineering improves reliability by adding review: role pipelines, debate, critic models, and voting [1]–[5]. A counter-current questions whether the agents are needed at all: Agentless [6] shows a simple validation-heavy pipeline matching complex scaffolds, and SWE-agent [7] finds that interface design, not agent count, drives performance.

We look at one narrow version of this question for code tasks whose target can be closed. On one side, reviewers judge an open output. On the other, the toolchain closes the target so an error becomes a mechanical event: a parse or type failure, a failed test, or a differential divergence against a reference. None of these mechanisms is new; constrained decoding [8], differential testing, and verified references all predate LLMs. Our question is empirical and small: on a public code benchmark, how do a mechanical closure gate and an LLM reviewer compare, and where do they disagree?

The honest answer from our pilot is unglamorous and, we think, more useful than a clean winner. On the defects we reviewed, the mechanical gate and the LLM reviewer catch bugs equally well, and the gate is far cheaper. Reviewers do reject some correct programs, which at face value favors the gate, but that comparison is confounded (the gate is graded against the very reference the "correct" programs come from). What survives is qualitative: we adjudicated every reviewer rejection of a correct program, and they divide into reviewers misreading a clear spec and reviewers correctly catching a reference or spec that is itself broken, which the gate, trusting that reference, cannot flag. Our contributions are:

- A pre-registered protocol and open harness comparing a mechanical closure gate against two cross-vendor LLM review baselines on a public substrate, with an independent audit set held out from the gate.
- A same-denominator comparison showing the gate and review catch these seeded defects about equally, so the interesting axis is false rejection, not missed bugs.
- A transparent, per-case adjudication of all false-rejection events, and the observation that they split between reviewer spec-misreads and reviewers catching defects in the reference.
- A short list of candidate defects in a widely used benchmark's references and specifications, surfaced by review and invisible to reference-agreement gates.

This is an exploratory pilot with small samples. We treat its counts as preliminary and its per-case analysis as the substance, and we flag the confounds explicitly.

II. Related Work

More agents, or fewer. Ensemble and debate frameworks [1]–[5] push toward larger panels; Agentless [6] and interface-centric work [7] push back, arguing structure and validation matter more than agent count. We share the skepticism and ask a narrower question: for a targetable task, does a mechanical gate add anything a reviewer does not, and vice versa.

Closing the target. Constrained decoding [8] makes malformed output mechanically rejectable; differential testing turns cross-implementation disagreement into an oracle. We reuse these; the only new thing here is the direct, if small, comparison against LLM review and the case analysis of where each fails.

Oracle validity. EvalPlus [9] argues the original HumanEval tests are insufficient and augments them. Our adjudication reaches a related point from a different direction: some augmented references and specs are themselves wrong on valid inputs, which matters for any technique, ours included, that treats a reference as ground truth.

III. Study Design

A. Pre-registration

The claim, task-selection rule, closure levels, seeded-bug classes, metrics, a 5% false-rejection cap, and the analysis plan were frozen before any result was seen, in a document released with the artifact. The pre-registered primary endpoint was wrong acceptance under an audit oracle at that cap. We report below that this endpoint was a non-differentiator, and we mark every other analysis, including the adjudication, as exploratory. Deviations (for example, operationalizing the strong reviewer as test-authoring plus harness execution) are logged there.

B. Substrate and tasks

We use EvalPlus HumanEval+ (release v0.1.10), a public benchmark with a documented history of oracle strengthening [9]. Tasks are a frozen systematic sample: every fourth problem by canonical id, the first 40. Each ships a prompt (signature and docstring), a canonical reference, a small base test set, and a much larger extended test set.

C. Gate levels and the audit set

A candidate is scored by its earliest catching gate. **L1** is parse, import, and signature; **L2** is the base test set; **L3** is a differential check against the canonical on a fixed random subset of the extended inputs. We reserve the remaining extended inputs, disjoint from the L3 subset, as an **audit set** that no gate level uses, so a defect the gates pass can still be caught later. We keep gate outcomes (L1–L3) and audit outcomes separate throughout.

D. Producer, reviewers, defects

A single producer (an Anthropic Claude model) generates candidates. Two review baselines use an OpenAI model of a different vendor through its command-line agent, so producer and reviewer never share a vendor. **L0-weak** judges by reading only. **L0-strong** authors its own test cases from the spec, computing expected outputs itself, and the harness runs them; a candidate is accepted iff it matches the reviewer's expectations on every case. Reviewers never see the hidden tests, the audit set, or the reference. Defects are 81 seeded bugs across four AST-injectable classes (wrong comparison operator, off-by-one, sign error, boolean operator flip), each verified to change behavior; correct-code controls are the canonical solutions. We note now that this control choice is a confound (§VI).

E. Adjudication scheme

Every reviewer rejection of a correct program is classified from the strong reviewer's captured failing case, or by hand for the weak reviewer, into: *reviewer misread* (the reference returns a value that the benchmark treats as correct, and the reviewer expected another), *reference/spec defect* (the reference errors on a valid input, or disagrees with its own written spec), or *oracle-uncertain* (the failing input may be out of the spec's domain). The first author adjudicated; we release each case for scrutiny, and we return to adjudicator bias in §VI.

TABLE I
EARLIEST CATCHING GATE FOR THE 81 SEEDED DEFECTS, AND CORRECT-CONTROL OUTCOMES. GATE LEVELS (L1–L3) AND THE HELD-OUT AUDIT SET ARE KEPT SEPARATE.

Caught first by	seeded defects
L1 (parse/signature)	0
L2 (base tests)	79
L3 (differential subset)	0
closure gates L1–L3	79 / 81
audit set (gates missed these)	2 / 81
correct controls passing all gates	40 / 40

TABLE II
SAME-DENOMINATOR CATCH ON THE 25 REVIEWED DEFECTS, AND FALSE REJECTION ON 40 CORRECT CONTROLS. THE $0/40$ GATE FIGURE IS STRUCTURAL (SEE TEXT).

	gates	L0-weak	L0-strong
catch, 25 defects	25/25	25/25	24/25
false reject, 40 controls	0/40 †	3/40	5/40

† structural: controls are the reference the gate uses.

IV. Results

A. Where defects terminate

Table I gives the earliest catching gate for the 81 seeded bugs. The base tests (L2) catch 79; the differential subset (L3) catches none beyond them; the held-out audit set catches the remaining 2. So the closure gates (L1–L3) catch $79/81$, and the audit set catches $2/81$ the gates missed. All 40 correct controls pass every gate. For these seeded, AST-level defects on tasks that ship tests, base testing does most of the work, and the differential subset adds nothing here.

B. Same-denominator catch, and false rejection

Comparing the gate and the reviewers requires the same defects. We reviewed a class-stratified subset of 25 seeded bugs. On exactly those 25, the closure gates catch $25/25$ (all at L2), L0-weak catches $25/25$, and L0-strong catches $24/25$ (the miss was a harness parse failure, with no reviewer verdict produced, not a substantive miss). On catching these defects the gate and review tie (Table II).

The gate and the reviewers differ on correct code. Reviewers rejected $3/40$ (L0-weak) and $5/40$ (L0-strong) correct programs; the gates rejected $0/40$. We do not present the $0/40$ as a finding: the correct controls are the canonical solutions, and a reference-agreement gate accepts the reference by construction (§VI). The informative part is not the rate but what the rejections were.

C. Per-case adjudication of the rejections

There are eight false-rejection events across the two reviewers and six distinct tasks (Table III). Three are genuine reviewer misreads of a clear specification. Four are the reviewer disagreeing with the reference where the reference is itself at fault: on a root-finding task the canonical raises a

division-by-zero on the valid input $[1, 0, 0, 1]$ (the polynomial $1 + x^3$, root -1), which both reviewers computed correctly; on a bracket task the spec asks about a nested *subsequence* while the reference checks only contiguous nesting; on a bit-count sort the spec says "non-negative" but ships a negative example. One case is oracle-uncertain (the reviewer's failing input for a base conversion is likely out of the stated domain). A reference-agreement gate is blind to all four reference/spec cases by construction, since it defines correctness as agreement with that reference.

D. Cost

The gates add no model calls; L1–L3 and the audit run in the harness. Each review costs on the order of 10^4 tokens (a single judgment for L0-weak, an agentic loop that writes and reasons over tests for L0-strong). The asymmetry is large and one-directional, so review is worth its cost only if it catches something the gate cannot. On these tasks the only such cases were the reference/spec defects above.

V. What the Disagreements Suggest

The pre-registered primary endpoint, wrong acceptance at a fixed false-rejection cap, did not separate the methods: the reviewers catch these seeded defects as well as the gate. We report that null plainly. The useful content is the per-case structure of the disagreements, and it points, tentatively, in one direction. The reviewer errs by misreading a clear spec and rejecting correct code, most often on counterintuitive or loosely worded specs. The gate errs by inheriting whatever is wrong in its reference: it cannot flag a reference that crashes, disagrees with its own spec, or rests on a contradictory spec, because agreement with that reference is its definition of correct.

We are cautious about how far this generalizes. With eight events on six tasks it is a set of illustrative cases, not a rate. But the cases are close to disjoint (where the reviewer is wrong the reference is right, and the reviewer's "false" rejections include the cases where the reference is wrong), which is at least consistent with a division of labor: a cheap reference-agreement gate as the primary filter, and scarce review aimed at the reference and spec themselves, where the gate is blind. Establishing it would need a larger study whose correct controls are independently written programs that differ from the reference, so that the gate's false-rejection rate is no longer measured against its own answer key; that is the follow-up we think matters most. The cost asymmetry already points to a working default, though: on targetable tasks, run the mechanical gate as the first filter, and spend review on the reference and specification rather than on re-judging code the gate has already checked.

VI. Threats to Validity

Same-reference confound. The gate, the tests, the audit inputs, and the correct controls all derive from the same benchmark reference. This makes the gate's $0/40$ false rejection near-tautological and is also the source of its blind spot; the same design choice that flatters the gate on false rejection is what the adjudication penalizes it for. Reviewing independently produced correct programs that differ from the reference is the main follow-up.

Small and non-independent samples. Forty tasks, 25 reviewed defects, 40 controls, eight rejection events on six tasks. Intervals are wide; some tasks contribute both a weak and a strong event. We do not lean on any rate.

Adjudicator bias. The first author adjudicated cases central to the thesis, without independent blinded adjudication. We mitigate only by releasing every case; independent adjudication is needed.

Reviewer configuration. One reviewer model in two prompting modes is not "review" as a method; results may be sensitive to model, prompt, and temperature.

Defect realism. Seeded bugs are four AST-mutation classes that base tests largely catch; they are not a representative distribution of generated-code failures, so the catch tie should not be read as a general claim.

Benchmark dependence. Using EvalPlus as both substrate and adjudication authority is partly circular; our reference/spec cases are candidate defects worth independent confirmation, not a settled indictment of the benchmark.

VII. Conclusion

On a small pre-registered pilot, a mechanical reference-agreement gate and a cross-vendor LLM reviewer catch seeded defects about equally, and the gate is far cheaper. The gate rejects no correct control, but only because it is graded against the reference those controls come from. Adjudicating every reviewer rejection of correct code, we find the rejections split between the reviewer misreading a clear spec and the reviewer correctly catching a reference or spec that is itself wrong, which the gate cannot see. We read this, cautiously, as suggestive of complementary failure modes and as a reason to point review at the reference rather than only at the code. Along the way the reviewers surfaced several candidate defects in a widely used benchmark's references and specs. All of this is preliminary at this scale, and we release the protocol, harness, and cases so it can be checked and extended.[1]

References

[1] C. Qian, W. Liu, H. Liu, N. Chen, Y. Dang, J. Li, C. Yang, W. Chen, Y. Su, X. Cong, J. Xu, D. Li, Z. Liu, and M. Sun, "ChatDev: Communicative agents for software development," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024, arXiv:2307.07924.

[2] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, C. Zhang, J. Wang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran, L. Xiao, C. Wu, and J. Schmidhuber, "MetaGPT: Meta programming for a multi-agent collaborative framework," *International Conference on Learning Representations (ICLR)*, 2024, arXiv:2308.00352.

[3] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, A. H. Awadallah, R. W. White, D. Burger, and C. Wang, "AutoGen: Enabling next-gen LLM applications via multi-agent conversation," *arXiv preprint arXiv:2308.08155*, 2023.

[1] Artifact: https://anonymous.4open.science/r/masgain2026-artifact

TABLE III

ALL EIGHT FALSE-REJECTION EVENTS, ADJUDICATED. "REVIEW RIGHT" MEANS THE REFERENCE OR SPEC IS AT FAULT, WHICH A REFERENCE-AGREEMENT GATE CANNOT SEE.

Task	Rev.	Class	What happened
HE/88	strong	misread	odd/even sort rule read backwards
HE/116	strong	misread	bit-count sort read as plain sort
HE/20	strong	misread	closest-pair arithmetic slip
HE/32	strong	ref bug	canonical div-by-zero on $[1, 0, 0, 1]$
HE/32	weak	ref bug	same
HE/132	weak	spec/ref	spec says subsequence, ref does contiguous
HE/116	weak	spec/ref	spec "non-negative" but negative example
HE/44	strong	uncertain	crash on a likely out-of-domain base

[4] Y. Du, S. Li, A. Torralba, J. B. Tenenbaum, and I. Mordatch, "Improving factuality and reasoning in language models through multiagent debate," in *International Conference on Machine Learning (ICML)*, 2024, arXiv:2305.14325.

[5] J. Li, Q. Zhang, Y. Yu, Q. Fu, and D. Ye, "More agents is all you need," *Transactions on Machine Learning Research (TMLR)*, 2024, arXiv:2402.05120.

[6] C. S. Xia, Y. Deng, S. Dunn, and L. Zhang, "Agentless: Demystifying LLM-based software engineering agents," *arXiv preprint arXiv:2407.01489*, 2024.

[7] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press, "SWE-agent: Agent-computer interfaces enable automated software engineering," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2024, arXiv:2405.15793.

[8] T. Scholak, N. Schucher, and D. Bahdanau, "PICARD: Parsing incrementally for constrained auto-regressive decoding from language models," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021, arXiv:2109.05093.

[9] J. Liu, C. S. Xia, Y. Wang, and L. 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 (NeurIPS)*, 2023, arXiv:2305.01210.