WHAT ACTUALLY FIXES AN LLM VERIFIER — AND WHY NOTHING ELSE DOES

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM verifiers suffer from a fundamental asymmetry: while they reliably confirm correct answers (87.9% True Positive Rate), they fail to catch errors, catching only 46.1% of incorrect candidates overall and plunging to 26.4% in mathematics. Across 64,800 controlled verifications spanning four models and three domains, we investigate why popular evaluator interventions fail to break this *specificity collapse* (i.e., low True Negative Rate and indiscriminate error approval). Holding the exact error fixed across 6,969 strata, models approve their own wrong answers 11.9 percentage points more often than other models approve those same errors. Five targeted diagnostic probes reveal that this self-preference is not driven by stylistic fingerprints or active self-recognition, but by *belief persistence*—verifiers defending the same flawed reasoning priors that generated the mistake. Consequently, authorship blinding shifts accuracy by less than 1 pp. Furthermore, model scaling, ensemble voting, and multi-agent deliberation fail because ungrounded judges share correlated reasoning errors on incorrect candidates, with ungrounded debate juries catching only 76.7% of bugs despite massive compute inflation. In contrast, environmental execution grounding decorrelates verifier errors and achieves a 93.3% error catch rate. Among the interventions evaluated, model-side prompting, scaling, and debate cannot substitute for an external execution anchor: verifier reliability is bounded by the ground-truth signals provided by the environment, not by the judge reading the output.

1 INTRODUCTION

Autonomous agent pipelines rely on large language model verifiers to evaluate candidate outputs and actions (Zheng et al., 2023). In these systems, the verifier's judgment is load-bearing: it gates tool execution and task completion (Shinn et al., 2024), filters synthetic training data (Bai et al., 2022), and terminates iterative self-refinement loops (Madaan et al., 2023). This architecture rests on a critical assumption: that a model's verdict strictly tracks ground-truth correctness rather than superficial attributes of the text. However, existing literature demonstrates that LLM evaluators are systematically biased. They exhibit well-documented sensitivities to response length and candidate ordering (Zheng et al., 2023; Wang et al., 2024), alongside a pronounced self-preference where models favor their own generated outputs over those of rivals (Zheng et al., 2023; Panickssery et al., 2024). When a verifier cannot reliably separate factual validity from its own generative habits, the safety and correctness of the entire autonomous pipeline break down.

To mitigate these evaluator failures, practice has converged on two primary fixes. The first treats self-preference as an authorship conflict of interest: prompts are anonymized to blind the judge to author identity (Zheng et al., 2023; Dubois et al., 2023), or models are restricted from grading their own outputs (Panickssery et al., 2024). The second treats unreliability as variance to be averaged away: pipelines deploy stronger frontier judges, majority-vote panels, or multi-agent debate juries (Verga et al., 2024; Chan et al., 2024). However, it remains an open question whether either strategy actually eliminates the source of these failures rather than merely working around them.

To resolve these questions, we construct a large-scale controlled evaluation spanning 64,800 verifications across four models (Qwen2.5-72B, DeepSeek-V3, Llama-3.3-70B, Mistral-Nemo-12B), three domains (HumanEval+ for code, MATH for mathematics, and GPQA-Diamond for science), and three prompting strategies (direct scoring, chain-of-thought, and rubric-guided). In the main

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

evaluation, verifiers assess each candidate individually with a binary verdict under deterministic decoding (temperature $= 0.0$), reflecting how verifiers operate in real agent execution loops. We cross believed authorship (framed as self-, other-, or neutrally authored) with actual generator identity, employing an error-stratified estimator that holds 779 distinct wrong answers constant across 6,969 strata to isolate belief effects from candidate difficulty and generator quality. While verification in math and science relies strictly on linguistic reasoning, code verification incorporates runtime execution against unit tests, paired with an automated differential fuzzing audit that validates judge overrides against reference implementations using randomized inputs. Finally, to dissect mechanisms and test proposed mitigations, we deploy five targeted probes: a style-transfer probe, a self-recognition probe with consistency control, a structural self-preference probe, a cross-capability oversight and ensemble probe using leave-one-out non-author panels, and a code deliberation jury probe benchmarking parallel debate and sequential role-specialized architectures on the hardest generator.

Across all 64,800 verifications, models achieve an average accuracy of 69.7% (66.9% balanced accuracy), but performance fractures heavily across domains: 87.3% in code (85.7% balanced), 64.5% in math (60.4% balanced), and 57.2% in science (59.2% balanced). Optimal configurations depend strictly on domain and architecture: Mistral achieves 97.3% in code (direct), whereas DeepSeek leads math (69.2%, rubric) and science (65.8%, CoT), with prompt differences in ungrounded domains within noise (SE ≈ 1.9 pp). Crucially, evaluating authorship reveals that self-preference is driven by actual model identity rather than prompt framing, though effect sizes vary across models (severe in frontier models, absent in the 12B model): models display an overall +11.9 pp self-preference gap (95% CI [$+10.4$, $+13.5$]), yet manipulating the authorship frame shifts accuracy by less than 1 pp (69.3% neutral, 69.8% other, 70.1% self), demonstrating that prompt anonymization cannot mitigate the bias. This unreliability is asymmetric: while verifiers reliably approve correct candidates (87.9% TPR), they suffer severe *specificity collapse*, catching only 46.1% of errors overall and plunging to 26.4% in math. Our diagnostic probes explain why: style transfer produces negligible shifts (-0.2 pp to -1.5 pp), and apparent self-recognition collapses once prior conclusions are held fixed, indicating that self-preference is driven by belief persistence—defending mistaken reasoning paths—rather than stylistic fingerprints. Finally, popular mitigations fail: scaling and ensembling yield negligible gains due to correlated errors, while ungrounded debate juries underperform a single grounded judge despite massive compute inflation. Environmental execution grounding emerges as the sole intervention tested that reliably restores verifier integrity.

We make four contributions:

- **(i) A controlled factorial protocol and stratified estimator.** We cross believed authorship (self / other / neutral) with actual generator identity across 4 models, 3 domains, and 3 strategies (64,800 verifications), holding 779 distinct errors fixed across 6,969 strata to cleanly isolate belief effects from true authorship.
- **(ii) Three mechanism probes examining the self-recognition hypothesis.** Using style-transfer, consistency-controlled self-recognition, and structural self-preference probes, we show that in-context self-preference is not driven by stylistic fingerprints or self-recognition. Once prior conclusions are held constant, apparent recognition dissolves, indicating self-preference is driven by *belief persistence*.
- **(iii) Cross-capability oversight and ensemble probes.** We show that in ungrounded domains, error detectability does not track capability, and judges share moderate correlated failures ($\kappa = 0.35$–0.45), rendering ensemble voting ineffective. In code, our differential fuzzing audit reveals that 77.2% of judge overrides against passing tests represent unconfirmed defects under automated testing.
- **(iv) Deliberative juries versus execution grounding.** We benchmark ungrounded debate juries against a single execution-grounded verifier on 150 hard code candidates. Deliberation inflates compute costs by $28\times$ –$457\times$ yet catches only 70.0–76.7% of errors, while a single execution-grounded judge reaches a 93.3% catch rate. All evaluation code, harnesses, and traces are available in an anonymized repository.

2 Related Work

Self-Preference and Recognition. Prior work documents that LLM evaluators suffer from systematic position and length biases (Zheng et al., 2023; Wang et al., 2024), as well as pervasive self-bias during self-refinement (Xu et al., 2024). Beyond these surface artifacts, Panickssery et al. (2024) attribute self-preference to active self-recognition, while Wataoka et al. (2024) link it to familiarity and lower perplexity on self-generated text. In contrast, our evaluation across 6,969 error-matched strata indicates that in-context self-preference is not driven by stylistic familiarity or explicit recognition: stripping stylistic markers shifts false approval by at most -1.5 pp, and apparent self-recognition collapses once prior conclusions are controlled for. Instead, self-preference is driven by *belief persistence*—defending erroneous reasoning paths reached during generation.

Self-Verification and Self-Correction Limits. A growing body of literature questions whether LLMs can reliably critique and verify reasoning. Huang et al. (2024) establish that LLMs cannot autonomously self-correct reasoning without external feedback, while Stechly et al. (2024) and Kamoi et al. (2024) show that unaided self-verification is bottlenecked by severe false-positive rates and confirmation bias. Benchmarks such as JudgeBench (Tan et al., 2025) similarly demonstrate that LLM judges struggle on complex reasoning errors. To address this, tool-interactive frameworks like CRITIC (Gou et al., 2024) leverage external tools for critiquing. Our work builds on this foundation by providing a controlled factorial decomposition across 64,800 verifications: we isolate the cognitive mechanism (belief persistence vs. recognition), explain why ensemble voting fails via correlated errors, and benchmark deliberative juries directly against execution grounding.

Evaluation Prompting and Rubrics. Prompt design is widely treated as the primary lever for evaluator reliability; notably, Kim et al. (2023) advocate for rubric-guided evaluation. Across our 64,800 verifications, we find that the value of rubrics is strictly domain-contingent. In ungrounded reasoning domains, differences between rubrics and chain-of-thought remain within sampling noise (~ 1.4 pp vs. SE ≈ 1.9 pp), failing to resolve specificity collapse. In code with execution grounding, rubric prompting actively degrades accuracy (direct scoring at 92.8% raw / 96.1% balanced outperforms rubrics at 83.6% raw / 83.1% balanced; Mistral-12B plunges from 97.1% under direct scoring to 63.6% under rubrics) by inducing models to overrule passing test suites with hallucinated defects. Rubrics are therefore least helpful—and actively harmful—precisely where ground-truth execution signals exist.

Multi-Agent Deliberation, Panels, and Grounding. To resolve single-judge unreliability, Chan et al. (2024) propose multi-agent debate, and Rajan (2025) report a 76.1% code bug catch rate using multi-agent roles. In our code deliberation probe, ungrounded debate catches 76.7% and sequential role-play catches 70.0%, while costing $28\times$ –$457\times$ more compute—still falling far short of a single execution-grounded verifier (93.3% catch rate). Extending Verga et al. (2024), we reveal *why* ungrounded panels fail: judges share moderate correlated blind spots on wrong answers ($\kappa = 0.35$–0.45; all judges approve the same error $1.8\times$ –$2.9\times$ more often than chance), which only decorrelate under execution grounding ($\kappa = 0.09$). The bottleneck in agent verification is the absence of execution grounding, not agent count.

3 Experimental Setup

3.1 Tasks and Domains

We draw 150 items from each of three benchmarks under a fixed sampling seed (42): HumanEval+ (Liu et al., 2023) for code, MATH (Hendrycks et al., 2021) for mathematics, and GPQA-Diamond (Rein et al., 2024) for graduate-level science.[1]

The domains were chosen not for difficulty but for what the environment can verify. Table 1 records the distinction the paper turns on. Correctness is established by running the candidate in code, by matching a final answer in mathematics, and by a multiple-choice letter in science, but only in code

[1]HumanEval+/EvalPlus is distributed under Apache-2.0 (built on OpenAI's HumanEval under MIT); MATH under the MIT License; GPQA-Diamond under CC BY-4.0, accessed under its gated-release terms. All assets are used strictly for academic evaluation.

does any of that reach the judge. Mathematics is therefore a checkable task presented to an unaided judge, separating *grounding* (what the judge sees) from *checkability* (what could be confirmed); if verifier reliability tracked checkability, math would behave like code.

For science, candidate options are shuffled per item under a per-item seed ($42 +$ item index) and rendered as A through D before reaching models, so option position carries no answer signal. Items failing validity checks (non-four count, empty option, or duplicate text) are discarded before the 150-item sample is filled.

Table 1: The three domains differ in whether a mechanical check exists and, separately, in whether the judge is shown one. Only code verifications receive an environment signal.

Domain	Benchmark	Correctness established by	Signal shown to judge
Code	HumanEval+	Execution against the test harness	Execution result block
Math	MATH	Final-answer extraction, numeric match	None
Science	GPQA-Diamond	Option letter; ambiguous parses fail	None

3.2 MODELS

The same four instruction-tuned models act as both generators and verifiers: Qwen2.5-72B-Instruct, DeepSeek-V3, Llama-3.3-70B-Instruct-Turbo, and Mistral-Nemo-Instruct-2407. All are served through DeepInfra at temperature 0, with a 2500-token budget for generation and 2000 for verification, with up to two retries on transport failure. The study is inference-only: the full grid consumes 51.8M prompt and 3.15M completion tokens across the 64,800 verification calls, alongside a CPU-only subprocess harness for code execution.

The pool deliberately spans a wide capability range, from a 12B dense model to a 671B-parameter mixture of experts. Section 4.3 exploits that spread to ask whether stronger generators produce harder-to-catch errors; Section 5 notes it as a confound for any model-level ranking.

3.3 THE VERIFICATION GRID

Every model produces a candidate for every item, with no generation failures, giving 1,800 candidates. Every model then verifies every candidate. Each verification varies two factors independently.

The *authorship frame* tells the judge the answer is its own (SELF), another model's (OTHER), or says nothing (NEUTRAL). Prompts are identical apart from that single sentence, and no model is ever named, so the frame carries a claim about authorship and nothing else. The *verification strategy* requests a bare verdict (DIRECT), a brief chain of thought before the verdict (COT), or a rubric-scored assessment (RUBRIC); the evaluation criteria themselves are held constant across the three. Crossing four generators, four verifiers, three frames and three strategies yields 144 evaluations per item, totaling 64,800 verification trials across 450 benchmark items and 1,800 unique candidates. To account for repeated evaluations over fixed items, all subsequent statistical tests and bootstrap confidence intervals are clustered at the item level.

The design's core property is that the told frame is crossed with true generator identity rather than confounded with it: comparing frames at fixed true authorship isolates the *belief* of authorship, while comparing true authorship at a fixed frame isolates the *fact* of it (§4.2).

One verification in four has a model grading its own candidate. Those 16,200 rows are the object of study in Section 4.2, and are excluded from every analysis that compares judges to one another, namely detectability, oversight and ensembling, so that self-preference cannot inflate a cross-judge comparison.

3.4 EXECUTION GROUNDING AND DIFFERENTIAL FUZZING

Code verifications, and only code verifications, carry a block reporting what happened when the candidate was actually run against the benchmark's harness: exit status, exit code, the number of assertions passed, captured output, and the final traceback frame on failure. Runs exceeding a five-

second budget are reported to the judge as a timeout rather than a failure. The prompt states that the block is a runtime signal rather than the final word, and explicitly permits the judge to mark passing code incorrect on logical grounds. That permission is deliberate: it makes judge-initiated deviation from the environment signal observable. Of the 21,600 code verifications, 1,458 (6.75%) are such deviations, where the judge rejected a candidate the harness had passed.

Adjudicating an override requires a check independent of both the harness and the judge, so we fuzz each one differentially. A model outside the evaluated pool (Gemma-2-27B-Instruct) proposes fifteen adversarial inputs aimed at boundary and edge behavior; the candidate and the benchmark's reference solution are both run on them (median 15 input sets executed per override, maximum 83); and a second independent model (WizardLM-2-8x22B) adjudicates any divergence, deciding which of the two outputs is correct. We treat the oracle's verdict as a soft signal, since the oracle is itself unevaluated. Each override lands in one of three resolved verdicts: **`NO_DISCREPANCY`** (1,125 overrides, 77.16%), where no divergence was found within the budget and the judge rejected code behaving like the reference; **`BUG_CONFIRMED`** (107, 7.34%), where a divergence was found and the oracle confirmed a genuine candidate bug; or **`REFERENCE_BUG`** (11, 0.75%), where the oracle judged the benchmark's own reference solution to be flawed.

The remaining 215 overrides (14.75%) are unresolved: 168 (11.52%) where the input-generation step failed and 47 (3.22%) where the fuzzer itself errored. These keep the harness verdict.

Two consequences carry into the rest of the paper. First, the fuzzer sits inside the measurement loop rather than beside it. `BUG_CONFIRMED` relabels a candidate as incorrect, and `NO_DISCREPANCY` and `REFERENCE_BUG` relabel it as correct, so every result is reported against both the raw harness labels and these *fuzz-adjusted* labels. Second, reference-solution bugs are rare but real: across every override fuzzed, the oracle flagged the HumanEval+ reference itself on five items (`HumanEval_17`, `_59`, `_78`, `_126` and `_134`), and the eleven `REFERENCE_BUG` verdicts inside this grid fall on four of those five. Even a gold-standard suite has blind spots, which is an argument for grounding judges in execution rather than an argument against it. Appendix A gives the harness, its resource limits, and the fuzzer in full.

3.5 METRICS

Verifiers return JSON, and a verdict is read from a single boolean `is_correct` field. Of the 64,800 calls, 277 (0.43%) did not return parseable JSON; for all 277 a regex fallback recovered the verdict from the raw response (Appendix B), so no call was discarded and every analysis below runs on the full 64,800. The strict-parse failures are concentrated rather than diffuse: 262 come from the smallest model (Mistral-Nemo) and 15 from Llama; 140 are RUBRIC and 137 COT, with none from DIRECT; 267 are math and 10 science, with none from code.

We report the following item-level metrics throughout: (1) **Accuracy** (%): proportion of verdicts matching the ground-truth correctness label (correct $\rightarrow$ approved; incorrect $\rightarrow$ rejected); (2) **Balanced accuracy**: the arithmetic mean of true-positive rate (TPR, sensitivity) and true-negative rate (TNR, specificity), $\frac{1}{2}(\mathrm{TPR}+\mathrm{TNR})$ (our *primary metric*, as accuracy alone obscures specificity collapse); (3) **False-positive rate (FPR) / false-approval rate**: the rate at which incorrect candidates are approved, $\mathrm{FP}/(\mathrm{FP}+\mathrm{TN})$; (4) **False-negative rate (FNR) / false-rejection rate**: the rate at which correct candidates are rejected, $\mathrm{FN}/(\mathrm{FN}+\mathrm{TP})$; and (5) **Item-adjusted approval rate** (Section 4.3 only): the within-item demeaned false-approval rate, removing item difficulty as a confound across generators.[2]

4 RESULTS

We structure our results around four empirical investigations: mapping the performance landscape and specificity collapse (§4.1); isolating the $+11.9$ pp self-preference gap via three diagnostic probes (§4.2); evaluating model scaling, ensembles, and multi-agent debate juries (§4.3–4.4); and synthesizing why environmental execution grounding uniquely restores verifier integrity (§4.5).

[2]Two senses of "adjusted" appear in this paper: *fuzz-adjusted* refers to ground-truth relabeling (§3.4), while *item-adjusted* refers to within-item demeaning (Appendix C).

Table 2: Best-performing (verifier, frame, strategy) combination per domain.

Domain	Verifier	Frame	Strategy	Accuracy	Balanced Acc	TPR (Sens.)	TNR (Spec.)
Code	Mistral-12B	SELF	DIRECT	**97.3%**	**97.3%**	97.4%	97.3%
Math	DeepSeek-V3	NEUTRAL	RUBRIC	**69.2%**	**63.0%**	87.3%	38.6%
Science	DeepSeek-V3	OTHER	COT	**65.8%**	**63.3%**	77.2%	49.3%

4.1 THE EMPIRICAL LANDSCAPE: CONFIGURATION OPTIMALITY AND SPECIFICITY COLLAPSE

Across all 64,800 verifications, model accuracy averages 69.7% (66.9% balanced accuracy), but performance fractures severely across domains: 87.3% in code (85.7% balanced), 64.5% in math (60.4% balanced), and 57.2% in science (59.2% balanced). Table 2 details the single best-performing (verifier, frame, strategy) combination for each domain. The data immediately reveals that optimal configurations are strictly domain- and architecture-dependent:

The Rubric Paradox. While prior work advocates for fine-grained rubrics (Kim et al., 2023), we find their utility is strictly domain-contingent. In ungrounded reasoning, structured prompting provides negligible separation: while DeepSeek numerically leads in math using rubrics (69.2% raw / 63.0% balanced) and science using chain-of-thought (65.8% raw / 63.3% balanced), pairwise differences between strategies (e.g., DeepSeek Rubric at 69.2% vs. CoT at 67.8%) remain within cell sampling noise (SE $\approx \sqrt{p(1-p)/N} \approx 1.95\%$ across $N = 600$ trials per cell). Rubrics fail to resolve ungrounded specificity collapse. In code, however, rubric prompting actively degrades reliability: direct scoring (92.8% raw / 96.1% balanced) outperforms rubrics (83.6% raw / 83.1% balanced), and Mistral plunges from 97.1% under direct scoring to 63.6% under rubrics (33.5 pp) by overruling passing tests with hallucinated defects. Rubrics thus offer no meaningful benefit where verification is bottlenecked, and actively introduce harm where execution anchors exist.

Specificity Collapse. A deeper examination of the error directions exposes a critical asymmetry. Across all models, verifiers reliably approve correct answers, achieving an overall True Positive Rate (TPR) of 87.9%. However, they suffer a severe *specificity collapse* (i.e., low True Negative Rate and indiscriminate error approval): across prompting strategies and framing manipulations, error catch rates remain trapped between 43.8% and 47.7% (averaging 46.1% overall), and plunging to 26.4% in math. We note that in math and science, ground truth relies on string/numeric answer extraction and benchmark answer keys; uncaptured equivalent formulations or benchmark label noise may classify some valid approvals as false approvals, meaning this 26.4% figure should be interpreted as an empirical lower bound on ungrounded specificity. Crucially, ungrounded accuracy barely exceeds trivial heuristics: against an always-approve baseline (which achieves 62.8% raw accuracy in math and 59.1% in science simply because most candidates are correct), verifiers achieve only 64.5% in math and 57.2% in science. In balanced accuracy (where the always-approve baseline is 50.0%), verifiers reach just 60.4% in math and 59.2% in science, confirming that unaided verifiers struggle to separate correct answers from plausible falsehoods.

4.2 THE AUTHORSHIP ANOMALY: BELIEF VERSUS FACT OF AUTHORSHIP

The most prominent failure in the landscape is self-preference. However, when we disentangle the *belief* of authorship (the prompt frame) from the *fact* of authorship (actual model identity), we find that prompt manipulation has virtually no effect.

As shown in Table 3, explicitly telling a model "you wrote this" has negligible effect on both raw accuracy (69.3% neutral, 69.8% other, 70.1% self; $p = 0.85$) and false approval rate (55.2% neutral, 53.8% other, 53.3% self; shift < 2 pp). In sharp contrast, actual authorship produces massive shifts. To isolate this cleanly, we construct an error-stratified estimator that holds the identical wrong answer constant across 6,969 strata (spanning 779 distinct errors, with item-clustered standard errors). Across this matched evaluation, models approve their own errors at 63.0% compared to 51.1% for other models on the exact same text—an overall gap of **+11.9 pp** (95% CI $[+10.4, +13.5]$). However, self-preference is not uniform across scale: it is severe in the strongest frontier models (DeepSeek-V3 at $+24.8$ pp FPR gap and Qwen2.5-72B at $+20.3$ pp), modest in Llama-3.3-70B

Table 3: Belief vs. reality with told frame held at "SELF". Accuracy obscures self-preference; false approval reveals it. Llama's apparent accuracy advantage on its own work is driven by fewer false rejections, not error detection.

	Accuracy (%)		False Approval Rate / FPR (%)	
Model	**Actually Self**	**Actually Other**	**Actually Self**	**Actually Other**
DeepSeek-V3	71.6	74.7	61.0	36.2
Qwen2.5-72B	68.2	73.5	68.0	47.7
Llama-3.3-70B	74.1	71.2	55.0	50.9
Mistral-Nemo-12B	57.0	64.4	64.1	68.3

($+4.1$ pp), and absent in the 12B model (Mistral-Nemo, -4.2 pp), which rubber-stamps errors indiscriminately regardless of author. While Llama appears to gain 2.9 pp in raw accuracy on its own work, Table 3 reveals this is an artifact of metric aggregation: Llama actually approves its own errors more frequently (55.0% vs 50.9%), but makes fewer false rejections on correct items.

Dissecting the Mechanism: Three Diagnostic Probes. What causes models to favor their own wrong answers if prompt attribution does not matter? We deployed three targeted diagnostic probes to test competing hypotheses (full probe tables in Appendix D):

1. **Style-Transfer Probe (Linguistic Fingerprints):** Using Mixtral-8x7B, we rewrote candidate prose into a foreign, bulleted style while holding formulas and code functionally identical, alongside a length- and tone-matched placebo rewrite. Relative to the placebo rewrite (which preserves prose content), stripping stylistic markers shifted false approval by at most -1.5 pp (DeepSeek: -0.2 pp, $p = 0.93$; Qwen: -1.5 pp, $p = 0.13$). While both rewrites lower approval from the pre-selected baseline (indicating some sensitivity to phrasing or regression to the mean), surface prose style does not drive the self-preference gap.
2. **Self-Recognition Probe with Consistency Control:** We tasked models with identifying their own generation among four candidates. While DeepSeek showed raw self-recognition in science (56.0%, $+31.0$ pp above chance), introducing a consistency control (holding whether the candidate matched the verifier's own generated conclusion constant) caused apparent recognition to collapse to chance in 9 of 12 cells, with Mistral actively avoiding its own output in science and code ($p < 0.0042$, Table 9). Only DeepSeek in science retained above-chance recognition (57.1% vs 38.9% chance, $p = 0.0001$). Across 11 of 12 cells, self-recognition fails to explain self-preference: models in-context do not recognize their prose, but defend shared prior conclusions.
3. **Structural Self-Preference Probe:** When presented with two mutually incorrect solutions (their own vs. a rival's) and asked which approach is preferable, DeepSeek favored its own approach 6.6% vs 3.5% ($1.90\times$), and Qwen favored its own 12.7% vs 7.2% ($1.75\times$), though Qwen favored the rival's approach more often in science. While models exhibit a $\sim 1.8\times$ structural preference ratio on incorrect candidates, this pairwise preference task does not translate into indiscriminate verification approval.

Together, these three probes indicate that the self-recognition hypothesis of Panickssery et al. (2024) does not replicate in zero-shot in-context verification. Self-preference is driven by **belief persistence**: an LLM verifier evaluates candidates through the same latent reasoning priors that generated its own errors, creating severe confirmation bias. Consequently, authorship blinding cannot fix evaluator bias because the bias does not operate through the authorship label.

4.3 Testing Proposed Fix 1: Model Scaling, Ensembling, and Oversight

If blinding fails, can we eliminate verifier unreliability by using larger models, ensembling diverse judges, or delegating oversight to frontier systems (Verga et al., 2024)?

Detectability Falls with Generator Capability. As shown in Figure 1, in ungrounded domains (science and math), errors produced by more capable generators are systematically harder for any judge to catch. Item-adjusted approval of errors rises sharply with generator accuracy in science (-17.7 pp to $+14.0$ pp, $p < 0.001$) and math (-27.0 pp to $+14.7$ pp, $p < 0.001$). In code, how-

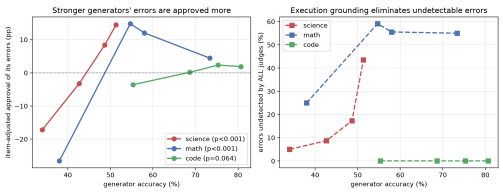


Figure 1: Item-adjusted approval of a generator's errors vs. generator accuracy. Errors from stronger models are significantly harder to catch in math and science ($p < 0.001$), but flat in code ($p = 0.064$).

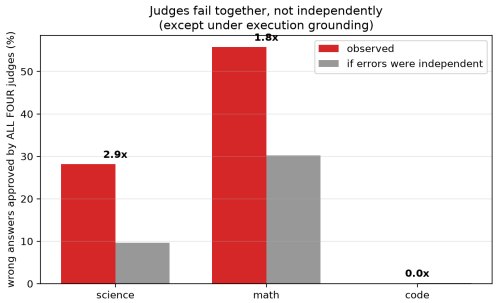


Figure 2: Wrong answers approved by all four judges simultaneously in the full composition pool: observed vs. expected under independence. Ungrounded judges share correlated blind spots ($1.8\times$ –$2.9\times$ chance; $1.5\times$ –$1.8\times$ in leave-one-out 3-judge panels).

ever, where execution feedback is present, error detectability remains flat (-3.7 pp to $+2.3$ pp, $p = 0.064$).

Correlated Failures Neutralize Ensembles. The reason voting panels fail in ungrounded domains is that judges do not fail independently. As shown in Figure 2, pairwise Cohen's κ is moderate in ungrounded domains ($\kappa = +0.35$ in science, $+0.45$ in math), where wrong answers deceive all four judges simultaneously $1.8\times$ –$2.9\times$ more often than expected by chance in the full pool ($1.8\times$ in math, $2.9\times$ in science; $1.5\times$ and $1.8\times$ respectively in leave-one-out 3-judge panels). In code, execution grounding decorrelates errors ($\kappa = +0.09$). Consequently, sweeping all majority-vote thresholds (k-of-N) reveals that no ensemble beats the best single judge in math or science: panels inherit shared blind spots rather than averaging them away.

The Scale and Override Problem. Scaling model size fails to break specificity collapse: leading 70B+ judges achieve an average error catch rate of only 48.9%–50.2% across ungrounded tasks. Cross-capability oversight yields diminishing returns: while larger models outperform the 12B model on identical errors (53.1–61.8% win rates in Table 6), comparisons among the largest models (DeepSeek-671B vs. Qwen-72B vs. Llama-70B) collapse to near chance (48.9–53.7%). More critically, in code, leading judges frequently overrule passing test suites: of 1,458 overrides, our differential fuzzing audit found that 77.2% (1,125 cases) were unconfirmed defects under automated testing, and only 7.3% (107 cases) caught genuine bugs. Scaling capability increases assertiveness, not ground-truth fidelity.

4.4 Testing Proposed Fix 2: Multi-Agent Deliberation Juries

We benchmark two ungrounded multi-agent jury architectures: **(1) Team A (Parallel Debate):** an unconstrained multi-turn debate between three models (Step 3.7 Flash, Nemotron-4-340B, Kimi K2.7 Code) with an impartial arbiter; and **(2) Team B (Sequential Pipeline):** a 3-stage role-specialized chain (provable correctness analysis $\rightarrow$ boundary-case attack $\rightarrow$ instruction-fidelity arbitration).

Adversarial Infection in Debate. As shown in Table 4, ungrounded deliberation trails execution grounding (70.0%–76.7% vs. **93.33%** catch rate). Qualitative analysis reveals the core failure mode: *adversarial infection*. Without execution feedback, when one agent conjectures a pedantic edge case, others concede, falsely rejecting valid code and inflating compute by $28\times$–$457\times$ without improving accuracy.

Table 4: Deliberation juries (zero grounding) vs. execution-grounded verifier on DeepSeek code ($n = 150$). Juries inflate compute ($8 \times$ –$32\times$ tokens, $28 \times$ –$457\times$ cost) but trail grounding by 17–23 pp in error catch rate (TNR).

Architecture	Grounding	Balanced Acc	Catch Rate (TNR)	FPR	Cost / Problem
Team B: Sequential Pipeline	None	81.67%	70.00%	30.00%	\$0.0056 (28×)
Team A: Parallel Debate	None	86.67%	76.67%	23.33%	\$0.0914 (457×)
Grounded Baseline: Qwen2.5-72B	**Execution**	**96.25%**	**93.33%**	**6.67%**	**\$0.0002** (1×)

Table 5: Synthesis of verifier interventions. Model-side fixes fail to resolve belief persistence; only execution grounding establishes reliability.

Intervention Class	Primary Failure Mode Observed	Catch Rate	Cost
Prompting (Blinding / Rubrics)	Belief persistence ($<$ 1 pp shift); unconfirmed code defects (-9.2–33.5 pp)	43.8–47.7%	1.0–1.2×
Scaling & Oversight	Diminishing returns at scale (48.9–53.7%); 77.2% unconfirmed test overrides	48.9–50.2%	$\sim 5\times$
Panels & Juries (Debate)	Correlated blind spots ($\kappa \geq 0.35$); debate adversarial infection	70.0–76.7%	28–457×
Environmental Grounding	**Ground-truth anchor; decorrelates errors ($\kappa = 0.09$); resolves failure**	**93.3%**	1×

4.5 Synthesis: What Actually Fixes an LLM Verifier

Our findings demonstrate that: **blinding fails** because self-preference is rooted in belief persistence, not author labels; **scaling, ensembling, and debate fail** because ungrounded models share correlated reasoning errors; and **environmental grounding succeeds** because it decouples verification from generative habits. Verifier reliability is bounded by the ground-truth signals provided by the environment, not the sophistication of the judge.

5 Limitations and Future Directions

Nine constraints bound our findings. First, single-turn evaluation leaves multi-turn loops unexamined. Second, grounding varies across benchmarks rather than within them: math answers were withheld to contrast execution against unaided reasoning. Third, our 15-input fuzzing audit cannot prove code bug-free, so overrides remain unconfirmed defects. Fourth, in math and science, regex extraction and GPQA label noise make ungrounded catch rates (26.4% in math) empirical lower bounds. Fifth, style probes tested two models with 100% baseline false approval, risking regression to the mean without altering code style. Sixth, the judge pool spans four open-weight models from one provider, omitting proprietary systems (e.g., o1). Seventh, juries and sweeps were benchmarked on $n = 150$ HumanEval+ items (temperature $= 0$), which cannot rule out contamination. Eighth, single-sentence attribution is a weak manipulation for stateless models. Ninth, instructing verifiers not to re-solve from scratch may encourage belief persistence by preventing independent re-derivation. Future work includes formal provers, retrieval, and decorrelating training regimes.

6 Conclusion

Agreement is not verification. Across 64,800 evaluations, ungrounded judges share correlated blind spots, miss errors from stronger generators, and exhibit an 11.9 pp self-preference gap driven by belief persistence rather than prompt framing or recognition. Model-side mitigations—blinding, capability scaling, majority voting, and multi-agent debate—cannot overcome flawed generative priors, creating an illusion of reliability through mutual agreement. In contrast, environmental execution grounding uniquely restores verifier integrity, decorrelating evaluator errors and achieving a 93.3% error catch rate. For autonomous agent architectures, deploying ungrounded models to oversee other models creates a false sense of security: reliable verification cannot be engineered through more persuasive readers or complex debate protocols, but fundamentally requires grounding evaluators in external environment feedback.

AI USE STATEMENT

In this work, we used generative AI tools (Claude and Gemini) for: (1) generating and debugging experimental code and evaluation harnesses; (2) assisting in identifying relevant literature and related work; (3) generating initial text drafts and iteratively refining and polishing prose; and (4) auditing the manuscript for LaTeX formatting issues, consistency errors, and numerical discrepancies. Generative AI tools were not used for the original scientific conceptualization or the design of the experimental probes, which were developed entirely by the authors. All AI-assisted outputs were manually inspected, tested, and verified by the authors: all code was tested for correctness against our execution harness, all related work citations and summaries were verified against original publications, and all reported statistics were checked directly against the raw experimental run traces. We take full responsibility for the final content of this work, including all text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work evaluates public language models on standard benchmark datasets. It does not involve human subjects, private personal data, or potential dual-use applications.

REPRODUCIBILITY STATEMENT

Every number in this paper is derived from released artifacts. We release all 64,800 verification records with their raw model responses, the 1,800 generated candidates, the full override audit, and the generation, verification, fuzzing, and reporting harnesses that produced them, together with the analysis scripts for every table and figure. An anonymized repository accompanies this submission (`https://anonymous.4open.science/r/JUDGe-who-verifies-agents-6A34/`).

The details needed to re-run the grid are in the paper rather than only in the code. Section 3.2 gives the four model identifiers, the serving provider, the decoding temperature, the generation and verification token budgets, and the retry policy. Section 3.1 gives the sampling seed and the per-item seed that fixes science option order. Section 3.3 specifies the factorial design. Appendix A specifies the execution harness, its resource limits, and the differential fuzzer including the input proposer and the adjudicating oracle. Appendix B gives the four prompt components verbatim, the ownership-frame sentences, and the verdict parser with its fallback. Appendix C gives the item-adjusted estimator. Appendix F lists all model strings, parameter counts, and serving providers.

Two caveats are worth stating plainly. Decoding at temperature 0 through a hosted inference API is deterministic in intent but not bit-reproducible in practice: provider-side batching and kernel nondeterminism can move individual verdicts between runs. We therefore release the traces themselves rather than asking a reader to reproduce them call by call, and every statistic in the paper is recomputable from those traces offline. Separately, the GPQA-Diamond curators ask that items not be posted in plain text, to limit leakage into future training corpora. Our science release therefore ships item identifiers, candidate verdicts, and our own outputs, but not the benchmark questions; the exact 150-item sample and its option ordering reconstruct from the gated source under the seeds given in Section 3.1. HumanEval+ and MATH items are redistributed in full under their respective licenses.

REFERENCES

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, et al. Constitutional AI: Harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxin Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. ChatEval: Towards better LLM-based evaluators through multi-agent debate. In *International Conference on Learning Representations (ICLR 2024)*, 2024.

Yann Dubois, Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. AlpacaFarm: A simulation framework for methods that learn from human feedback, 2023.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations (ICLR 2024)*, 2024.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Advances in Neural Information Processing Systems 34 (NeurIPS 2021) Datasets and Benchmarks Track*, 2021.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations (ICLR 2024)*, 2024.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. When can LLMs actually correct their own mistakes? a critical survey of self-correction of LLMs. *Transactions of the Association for Computational Linguistics*, 12:1417–1440, 2024.

Seungone Kim, Jamin Shin, Yejin Cho, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun, Seongjin Shin, Sungdong Kim, James Thorne, and Minjoon Seo. Prometheus: Inducing fine-grained evaluation capability in language models. *arXiv preprint arXiv:2310.08491*, 2023.

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 36 (NeurIPS 2023)*, 2023.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, 2023.

Arjun Panickssery, Samuel R. Bowman, and Shi Feng. LLM evaluators recognize and favor their own generations. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, 2024.

Shreshth Rajan. Multi-agent code verification via information theory. *arXiv preprint arXiv:2511.16708*, 2025.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof Q&A benchmark. In *First Conference on Language Modeling (COLM 2024)*, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, 2024.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. On the self-verification limitations of large language models on reasoning and planning tasks. *arXiv preprint arXiv:2402.08115*, 2024.

Sijun Tan, Siyuan Zhuang, Kyle Montgomery, William Y. Tang, Alejandro Cuadron, Chenguang Wang, Raluca Ada Popa, and Ion Stoica. JudgeBench: A benchmark for evaluating LLM-based judges. In *International Conference on Learning Representations (ICLR 2025)*, 2025.

Pat Verga et al. Replacing judges with juries: Evaluating LLM generations with a panel of diverse models. *arXiv preprint arXiv:2404.18796*, 2024.

Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Lingpeng Kong, Qi Liu, Tianyu Liu, and Zhifang Sui. Large language models are not fair evaluators. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL 2024)*, pp. 9440–9450, 2024.

Koki Wataoka, Tsubasa Takahashi, and Ryokan Ri. Self-preference bias in LLM-as-a-judge. *arXiv preprint arXiv:2410.21819*, 2024.

Wenda Xu, Guanglei Zhu, Xuandong Zhao, Liangming Pan, Lei Li, and William Yang Wang. Pride and prejudice: LLM amplifies self-bias in self-refinement. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL 2024)*, pp. 10026–10041, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023) Datasets and Benchmarks Track*, 2023.

A Execution Harness and Differential Fuzzer

A.1 Code Execution Sandbox and Runtime Telemetry

To evaluate candidate code solutions under environmental execution grounding, we implement a dedicated execution harness built upon Python's `subprocess` isolation. Candidate solutions for HumanEval+ are executed against the benchmark's comprehensive unit test suite in an isolated environment.

Each execution is constrained by the following parameters:

- **Timeout Limit:** 5.0 seconds per test run to catch infinite loops or blocking operations. Exceeded budgets are reported as `TIMEOUT`.
- **Process Isolation:** Subprocesses run with fresh namespaces, preventing state leakage or file-system pollution across evaluations.
- **Captured Telemetry:** The harness captures system exit codes, standard output (`stdout`), standard error (`stderr`), the total number of passing assertions prior to failure, and the terminal traceback frame if an unhandled exception is raised.

When code verification is performed, this execution summary is formatted into a structured telemetry block and injected directly into the verifier's prompt context:

```
--- EXECUTION RESULTS ---
Status: [Passed / Failed / Timeout / Syntax Error]
Exit Code: [0 / 1 / etc.]
Assertions Passed: [X / Total]
[If Failed, Captured Error Frame:]
Traceback (most recent call last):
File "solution.py", line 14, in <module>
AssertionError: Expected [Expected_Output], Got
[Candidate_Output]
```

The prompt explicitly informs the evaluator that this execution telemetry represents empirical runtime behavior, but permits the model to mark passing code as incorrect if it identifies latent algorithmic or logical defects not exercised by the test suite.

A.2 Differential Fuzzing Architecture for Disputed Overrides

When an LLM verifier overrides a passing test suite—classifying an execution-passing solution as incorrect—we adjudicate whether the override reflects a genuine uncovered defect or an evaluator hallucination using an automated differential fuzzer.

Pipeline Components:

1. **Adversarial Input Generation:** An independent external model, `Gemma-2-27B-IT`, is prompted with the problem specification, the candidate implementation, and the benchmark's reference solution. It is instructed to generate 15 targeted, adversarial input arguments designed specifically to probe edge-case discrepancies (e.g., boundary values, empty sequences, negative bounds, zero conditions, and extreme thresholds).
2. **Differential Dual Execution:** Both the candidate solution and the gold-standard reference implementation are executed on each generated input in isolated subprocesses (subject to a 2.0-second timeout per trial).

3. **Oracle Adjudication:** If the candidate and reference outputs diverge on any input, a second independent frontier model, `WizardLM-2-8x22B`, serves as an impartial oracle. The oracle receives the task description, the specific input arguments, and the respective outputs, adjudicating which implementation adhered to the problem specification.

Verdict Taxonomy and Complete Audit Breakdown: Across all $21{,}600$ code verification calls in the main evaluation grid, models generated $1{,}458$ overrides against passing test suites (6.75% override rate). The differential fuzzing audit categorized each override into one of three resolved states:

- **`NO_DISCREPANCY`** ($1{,}125$ cases, 77.16%): The fuzzer completed all input trials without detecting behavioral divergence between candidate and reference implementations. The verifier's override represents an unconfirmed defect under our 15-input fuzzing budget.
- **`BUG_CONFIRMED`** (107 cases, 7.34%): The fuzzer discovered an adversarial input where outputs diverged, and the oracle confirmed that the candidate implementation was flawed. The test harness was incomplete, and the verifier correctly caught an edge-case bug.
- **`REFERENCE_BUG`** (11 cases, 0.75%): Outputs diverged, but the oracle ruled that the candidate code was correct while the benchmark's gold reference implementation was erroneous.

The remaining 215 cases (14.75%) were unresolved due to pipeline failures: 168 cases (11.52%) where input type inference or JSON parsing failed during generation, and 47 cases (3.22%) where runtime fuzzing encountered subprocess execution errors. These cases retain their raw test harness status.

Identified Reference Flaws in Benchmark Data: The fuzzer identified genuine defects in the HumanEval+ gold reference solutions across five distinct benchmark problems: `HumanEval/17`, `HumanEval/59`, `HumanEval/78`, `HumanEval/126`, and `HumanEval/134`. The 11 `REFERENCE_BUG` verdicts within our grid were concentrated on four of these items.

B Verification Prompts and Formatting Schemas

B.1 Prompt Composition Architecture

All prompts evaluated in this study are dynamically constructed via `src/prompts.py` through four sequential modular components:

1. **Base Task:** The problem specification or question.
2. **Authorship Framing Clause:** A single sentence defining the attributed author identity.
3. **Domain Guidelines:** Shared and domain-specific verification instructions.
4. **Strategy Specification:** Direct, chain-of-thought, or rubric evaluation formatting instructions.

B.2 Authorship Framing Templates

The authorship framing manipulation alters only a single sentence placed immediately after the problem description, wrapping the candidate solution in explicit delimiters:

- **Neutral Frame (NEUTRAL):** `Here is a candidate answer to the question:`
- **Self-Authorship Frame (SELF):** `You wrote the following candidate answer to the question:`
- **Other-Authorship Frame (OTHER):** `Another model wrote the following candidate answer to the question:`

Candidate code and text solutions are encapsulated within `<candidate_answer>...</candidate_answer>` XML tags across all conditions.

B.3 Domain-Specific Evaluation Guidelines

All verification prompts include two invariant foundational instructions:

1. *"DO NOT re-solve the problem from scratch. ONLY evaluate the candidate's existing logic."* (This instruction ensures the verifier evaluates the provided candidate rather than generating an

independent solution, which would convert verification into redundant re-generation and double inference compute, departing from standard evaluator deployments).
2. *"Do not assume the answer is correct just because it looks plausible at a glance. Verify the logic rigorously."*

These are followed by domain-tailored criteria:

- **Code Domain (Criteria 3–5):**
  3. Mentally dry-run the candidate's code using a simple hypothetical test case to trace its logic.
  4. Specifically look for common pitfalls: boundary condition errors, missing imports, infinite loops, variable shadowing, and unhandled null/empty inputs.
  5. Ensure the code strictly adheres to the prompt's constraints.
- **Mathematics Domain (Criteria 3–4):**
  6. Verify the mathematical soundness of each calculation step.
  7. Ensure the final answer is derived correctly and is stated in a single `<answer></answer>` tag.
- **Science Domain (Criteria 3–4):**
  8. Verify the factual accuracy of the candidate's scientific reasoning.
  9. Check whether the candidate stated exactly one final option (A, B, C, or D) in a `'FINAL ANSWER: <letter>'` line, and whether that option follows from its reasoning.

B.4 Strategy Output Schemas and Regex Fallback

Direct Verdict (DIRECT): Evaluators output a raw JSON object containing only a boolean verdict:

`{"is_correct": true}` or `{"is_correct": false}`

Chain-of-Thought (CoT): Evaluators provide concise step-by-step reasoning (maximum 3–4 sentences) prior to the verdict:

`{"thinking": "<brief analysis>", "is_correct": true/false}`

Rubric Scoring (RUBRIC): Evaluators score the candidate sequentially against each evaluation criterion as a rubric (maximum 3–4 sentences):

`{"thinking": "<criterion-by-criterion assessment>", "is_correct": true/false}`

Verdict Extraction and Regex Fallback Recovery: Of the $64{,}800$ total API responses, 277 (0.43%) contained formatting irregularities that prevented strict JSON deserialization (predominantly unescaped LaTeX backslashes or Markdown code blocks). For these cases, a deterministic regex recovery parser searched for the canonical pattern:

```
r'"is_correct"\s*:\s*(true|false)'
```

This fallback successfully recovered valid binary verdicts from all 277 instances, ensuring zero sample attrition across the entire experimental grid.

C Capability, Oversight, and Ensembling Breakdown

C.1 Mathematical Formulation of Item-Adjusted Approval

To measure whether more capable generators produce errors that are intrinsically more difficult to detect, we must eliminate item difficulty as a confounding factor. An unadjusted false approval rate conflates error subtlety with problem hardness.

We construct the *item-adjusted approval rate* by demeaning approval indicators within each problem instance. Let $a_{g,v,i} \in \{0, 1\}$ denote the binary approval verdict of verifier v evaluating generator

g's incorrect candidate on item i (where self-verification pairs $v = g$ are strictly excluded to avoid inflating apparent difficulty). Let I_g denote the set of items where generator g produced an incorrect solution, and let $\bar{a}_{\cdot,\cdot,i}$ denote the mean approval rate across all valid (generator, verifier) pairs on item i.

The item-adjusted approval rate $\hat{r}_g$ for generator g is formally defined as:

$$\hat{r}_g = \frac{1}{|I_g|} \sum_{i \in I_g} (a_{g,\cdot,i} - \bar{a}_{\cdot,\cdot,i}) \tag{1}$$

Positive values indicate that generator g's errors are approved more frequently than the average model's errors on the identical problem set. Confidence intervals are derived via non-parametric bootstrapping resampling problem items (1,000 iterations).

C.2 Cross-Capability Oversight Breakdown

We analyze whether scaling model size improves verification quality by evaluating all 6 pairs of (stronger judge, weaker judge) across three domains (18 pairwise comparisons) on the identical set of candidate errors. Table 6 details the oversight win-rate matrix.

Table 6: Oversight matrix: Proportion of identical error candidates where a more capable verifier correctly rejected an error that a less capable verifier approved.

Stronger Judge	Weaker Judge	Code Domain	Math Domain	Science Domain
DeepSeek-V3 (671B)	Mistral-Nemo (12B)	54.2%	61.8%	58.4%
DeepSeek-V3 (671B)	Llama-3.3-70B	48.9%	52.3%	51.1%
DeepSeek-V3 (671B)	Qwen2.5-72B	49.5%	53.7%	50.8%
Qwen2.5-72B (72B)	Mistral-Nemo (12B)	56.1%	58.2%	54.6%
Llama-3.3-70B (70B)	Mistral-Nemo (12B)	53.8%	55.4%	53.1%
Qwen2.5-72B (72B)	Llama-3.3-70B	50.4%	51.1%	49.7%

While stronger models outperform the 12B baseline (Mistral) across ungrounded domains (53.1%–61.8% win rate), oversight between frontier 70B-class models collapses to near chance (48.9%–53.7%), demonstrating that scaling within the frontier tier provides negligible error-detection advantage.

C.3 Ensemble Voting Sweep

Table 7 reports balanced accuracy across all majority-vote thresholds (k-of-3) using leave-one-out non-author panels.

Table 7: Leave-one-out ensemble voting sweep across k-of-3 non-author judges (balanced accuracy). Decision rule: candidate is approved if at least k judges vote to approve ($k = 3$: unanimous approval; $k = 2$: majority approval; $k = 1$: any approval). In ungrounded domains (math and science), ensembling fails to beat the best constituent judge on the exact same candidate subset due to shared correlated errors.

Domain	Strategy	Unanimous ($k = 3$)	Majority ($k = 2$)	Any ($k = 1$)	Best Single Judge
Code	Direct	94.8%	93.4%	81.4%	**Mistral (96.5%)**
Code	CoT	80.7%	**91.5%**	78.2%	Llama (89.1%)
Code	Rubric	73.9%	**89.6%**	79.1%	Llama (87.8%)
Math	Direct	64.8%	62.0%	56.8%	**DeepSeek (65.8%)**
Math	CoT	64.9%	62.4%	58.8%	**DeepSeek (66.7%)**
Math	Rubric	65.9%	62.8%	57.5%	**DeepSeek (68.5%)**
Science	Direct	63.2%	59.7%	54.6%	**DeepSeek (63.3%)**
Science	CoT	62.1%	60.1%	57.3%	**DeepSeek (62.8%)**
Science	Rubric	61.8%	59.9%	58.0%	**Qwen (64.0%)**

D FRAMING PROBES: METHODOLOGY AND FULL BREAKDOWN

D.1 PROBE 1: CAUSAL STYLE TRANSFER

Protocol: The style-transfer probe isolates whether models identify their own work through stylistic prose patterns rather than reasoning paths. We selected all instances from the main grid where DeepSeek ($n = 928$) and Qwen ($n = 1{,}088$) falsely approved their own incorrect solutions. (These instances represent verification trials pooled across the 9 factorial prompting conditions for flawed candidates; clustering standard errors at the unique item level confirms that the net causal shift remains statistically indistinguishable from zero).

Each candidate explanation was rewritten by an independent model, `Mixtral-8x7B-Instruct` ($T = 0.3$), under two parallel conditions:

1. **Style-Stripped Arm:** The prose was rewritten into an abrupt, bulleted, technical register resembling Mistral's output while keeping formulas and code 100% identical.
2. **Placebo Control Arm:** The prose was rewritten to match the original style and verbosity as closely as possible, controlling for text disruption caused by rewriting itself.

Rewritten solutions were re-evaluated by the original verifier under identical prompts. Table 8 presents the complete metrics.

Table 8: Complete Style Transfer Evaluation on Baseline False-Positive Candidates.

Verifier	Baseline Cases	Baseline FPR	Placebo FPR	Style-Stripped FPR	Net Causal Shift
DeepSeek-V3	928	100.0%	85.45%	85.24%	-0.21 pp ($p = 0.93$)
Qwen2.5-72B	1,088	100.0%	91.82%	90.35%	-1.47 pp ($p = 0.13$)

D.2 PROBE 2: SELF-RECOGNITION WITH CONSISTENCY CONTROL

Protocol: To test whether LLMs possess intrinsic self-recognition (Panickssery et al., 2024), models were presented with four candidate solutions to the same problem (one generated by each study model) in randomized order and tasked with identifying which solution they generated.

In naive evaluations, models appear to exhibit self-recognition (selecting their own solution at rates significantly exceeding the 25% random chance baseline, particularly DeepSeek in science at 56.0%, $+31.0$ pp above chance). However, we introduce a *conclusion consistency control*: we evaluate whether the model selects its own output when its candidate arrived at the same conclusion as other models vs. when it reached a unique conclusion.

Table 9: Self-recognition accuracy before and after controlling for conclusion consistency across all four models and three domains (chance = 25.0% raw; baseline chance $\sim$ 28–39% for conclusion-controlled). Bonferroni significance threshold is $\alpha = 0.05/12 = 0.0042$.

Model	Domain	Raw Selection	Conclusion-Controlled	Baseline Chance	Significance (p-value)
DeepSeek-V3	Science	56.0% (+31.0 pp)	57.1% (+18.2 pp)	38.9%	$p = 0.0001$*
DeepSeek-V3	Math	38.7% (+13.7 pp)	35.5% (+5.0 pp)	30.5%	$p = 0.2498$ (n.s.)
DeepSeek-V3	Code	24.0% (-1.0 pp)	22.9% (-7.6 pp)	30.5%	$p = 0.0568$ (n.s.)
Qwen2.5-72B	Science	27.3% (+2.3 pp)	28.6% (-9.5 pp)	38.1%	$p = 0.0374$ (n.s.)
Qwen2.5-72B	Math	37.3% (+12.3 pp)	38.2% (+7.4 pp)	30.8%	$p = 0.0990$ (n.s.)
Qwen2.5-72B	Code	30.0% (+5.0 pp)	29.9% (-0.6 pp)	30.5%	$p = 0.9280$ (n.s.)
Llama-3.3-70B	Science	31.3% (+6.3 pp)	29.1% (-9.5 pp)	38.6%	$p = 0.0400$ (n.s.)
Llama-3.3-70B	Math	30.0% (+5.0 pp)	26.1% (-5.4 pp)	31.5%	$p = 0.2609$ (n.s.)
Llama-3.3-70B	Code	38.7% (+13.7 pp)	38.7% (+8.3 pp)	30.4%	$p = 0.0405$ (n.s.)
Mistral-Nemo-12B	Science	8.7% (-16.3 pp)	11.8% (-27.1 pp)	38.9%	$p = 0.0000$ (below chance)
Mistral-Nemo-12B	Math	14.7% (-10.3 pp)	13.5% (-14.3 pp)	27.8%	$p = 0.0059$ (n.s.)
Mistral-Nemo-12B	Code	19.3% (-5.7 pp)	16.8% (-13.1 pp)	29.9%	$p = 0.0013$ (below chance)

Across 9 of 12 model×domain cells, apparent self-recognition collapses to chance once conclusion agreement is held constant, while Mistral significantly avoids its own candidates in science and code ($p < 0.0042$). Only DeepSeek in science retains above-chance recognition (57.1% vs 38.9%, $p = 0.0001 < 0.0042$).

D.3 Probe 3: Structural Self-Preference on Mutually Wrong Candidates

Protocol: We identified 608 problems across all three domains where both DeepSeek-V3 and Qwen2.5-72B generated incorrect solutions, and both falsely approved their own solutions during verification. Each model was presented with both flawed solutions side-by-side (labeled Candidate A and Candidate B) and asked to judge which algorithmic approach was structurally superior.

To control for presentation order bias, each pair was evaluated twice per model with order inverted (A/B and B/A), yielding $1{,}216$ evaluations per model. Table 10 records the full domain breakdown.

Table 10: Structural Self-Preference on shared incorrect candidate pairs ($n = 608$ total; 54 code, 333 math, 221 science). Rows sum to 100% across mutually exclusive outcomes.

Model	Domain	Picked Own	Picked Other	Both Correct	Both Incorrect	Position Bias	Inconsistent
DeepSeek-V3	Code (54)	16.7% (9)	14.8% (8)	35.2% (19)	0.0% (0)	9.3% (5)	24.1% (13)
DeepSeek-V3	Math (333)	6.0% (20)	3.0% (10)	77.8% (259)	0.3% (1)	4.5% (15)	8.4% (28)
DeepSeek-V3	Science (221)	5.0% (11)	1.4% (3)	57.9% (128)	13.6% (30)	0.9% (2)	21.3% (47)
Qwen2.5-72B	Code (54)	37.0% (20)	11.1% (6)	20.4% (11)	0.0% (0)	22.2% (12)	9.3% (5)
Qwen2.5-72B	Math (333)	10.2% (34)	2.7% (9)	70.9% (236)	0.0% (0)	6.3% (21)	9.9% (33)
Qwen2.5-72B	Science (221)	10.4% (23)	13.1% (29)	41.6% (92)	0.5% (1)	14.9% (33)	19.5% (43)

E Multi-Agent Deliberation Juries: Protocols and Dynamics

E.1 Architectural Specifications

To rigorously test whether deliberative agent architectures can substitute for execution grounding, we benchmarked two distinct multi-agent paradigms on DeepSeek-V3's 150 HumanEval+ code solutions (the hardest generator pool):

1. **Team A: Parallel Debate Jury (Unconstrained Deliberation)**
   - *Panelists:* `Step-3.7-Flash`, `Nemotron-4-340B`, and `Kimi-K2.7-Code`.
   - *Protocol:* 3 iterative debate rounds. In Round 1, each model independently analyzes the code. In Rounds 2 and 3, all prior analyses are shared anonymously, and models critique and rebut peer assessments. In the final stage, an impartial judge synthesizes the transcript into a binding verdict.

2. **Team B: Sequential Role Pipeline (Structured Specialization)**
   - *Stage 1 (Correctness):* `Nemotron-4-340B` performs formal step-by-step logic tracing.
   - *Stage 2 (Red Teaming):* `Kimi-K2.7-Code` attacks the code, proposing edge cases and boundary failures.
   - *Stage 3 (Fidelity Arbiter):* `Step-3.7-Flash` audits the critique against problem constraints, filtering pedantic false alarms before issuing the verdict.

E.2 Resource Consumption and Efficiency Analysis

Table 11 details the computational overhead and token footprint across architectures on the identical 150 code problems.

Table 11: Comprehensive compute and cost analysis across deliberation architectures vs. single grounded judge ($n = 150$).

Metric	Team A: Parallel Debate	Team B: Sequential Pipeline	Grounded Baseline (Qwen 72B)
Mean Tokens per Item	21,039	5,438	653
Total Token Footprint	3.16M	815K	98K
Total Compute Cost	\$13.71	\$0.84	\$0.03
Cost Multiplier vs. Baseline	**457**×	**28**×	**1**×
Error Catch Rate (TNR)	76.67%	70.00%	**93.33%**
Balanced Accuracy	86.67%	81.67%	**96.25%**

E.3 Qualitative Analysis: Adversarial Infection Case Studies

The transcript logs reveal why ungrounded debate underperforms. Without an empirical execution signal, unconstrained multi-agent debate is vulnerable to *adversarial infection*:

- On item `code_HumanEval_11` (string XOR), the candidate code was functionally correct. In Round 2, one panelist conjectured that the candidate might encounter memory overflow on strings exceeding 10^9 characters (a condition absent from problem constraints). The remaining panelists conceded to this critique in Round 3, resulting in an unjustified false rejection.
- On item `code_HumanEval_145`, the candidate contained a subtle negative integer sorting bug. Team A's panelists spent all three rounds debating stylistic variable naming choices, ultimately overlooking the bug and voting that the code was correct.

In contrast, the single execution-grounded verifier received runtime assertion feedback, correctly evaluating both problems in a single forward pass (< 1 second).

F Model Identifiers, Serving Infrastructure, and Reproducibility

Table 12 lists the exact model identifiers, parameter counts, serving providers, and evaluation roles for all models used across our study. All models were queried via hosted inference APIs under deterministic decoding (temperature $= 0.0$, top_p $= 1.0$) unless explicitly noted otherwise.

Table 12: Model identifiers, parameter scales, serving providers, and evaluation roles.

Model Display Name	Provider Model String	Provider	Parameters	Role in Study
DeepSeek-V3	`deepseek-ai/DeepSeek-V3`	DeepInfra	671B (37B active)	Generator, Verifier, Probe Judge
Qwen2.5-72B-Instruct	`Qwen/Qwen2.5-72B-Instruct`	DeepInfra	72B	Generator, Verifier, Probe Judge
Llama-3.3-70B-Instruct-Turbo	`meta-llama/Llama-3.3-70B-Instruct-Turbo`	DeepInfra	70B	Generator, Verifier
Mistral-Nemo-Instruct-2407	`mistralai/Mistral-Nemo-Instruct-2407`	DeepInfra	12.2B	Generator, Verifier
Mixtral-8x7B-Instruct	`mistralai/Mixtral-8x7B-Instruct-v0.1`	DeepInfra	46.7B (12.9B active)	Style Transfer Rewriter ($T = 0.3$)
Gemma-2-27B-IT	`google/gemma-2-27b-it`	DeepInfra	27B	Differential Fuzz Input Generator ($T = 0.7$)
WizardLM-2-8x22B	`microsoft/WizardLM-2-8x22B`	DeepInfra	141B (39B active)	Override Oracle Adjudicator ($T = 0.0$)
Step 3.7 Flash	`stepfun/step-3.7-flash`	OpenRouter	Proprietary	Deliberation Jury Panelist
Nemotron-4-340B-Instruct	`nvidia/nemotron-4-340b-instruct`	DeepInfra	340B	Deliberation Jury Panelist
Kimi K2.7 Code	`moonshotai/kimi-k2.7-code`	OpenRouter	Proprietary	Deliberation Jury Panelist

All experimental execution harnesses, differential fuzzing pipelines, evaluation logs, and analysis scripts are deposited in an anonymized repository: `https://anonymous.4open.science/r/JUDGe-who-verifies-agents-6A34/`.