CertJudge: Evaluating Lean Formal-Code With Falsifiable Properties

Ethan S Hersch [*1] **Brando Miranda** [*1] **Elyas Obbad** [1] **Srivatsava Daruru** [1] **Zhanke Zhou** [2] **Kirill Acharya** [1] **Sanmi Koyejo** [1]

Abstract

LLM judges are increasingly used to evaluate AI-generated code, tests, theorems, and formal specifications. In our setting, a judge scores the quality of a candidate Lean 4 artifact relative to a gold reference, but validating each new judge, prompt, model, or threshold with fresh human ratings is expensive and does not scale. We ask whether this validation can be amortized through behavioral checks that are automatic and falsifiable. We propose CERTJUDGE, a property-certification protocol for LLM judges of Lean 4 theorem and specification quality. Instead of treating human agreement as the only validation signal, CERTJUDGE tests whether a judge behaves correctly under controlled perturbations measuring identity, bug monotonicity, specification monotonicity, and stability. These diagnostics are combined into a variance-aware trust index, $\mathrm{TI}_{\mathrm{var}}$, which provides a summary of judge reliability rather than a universal evaluator constant. On a human-labeled validation suite, $\mathrm{TI}_{\mathrm{var}}$ strongly predicts judge-level human alignment, reporting validation-set Spearman correlations of 0.833 (avg labels), 0.905 (pass1), and 0.738 (pass2) (Appendix G). On VERIBENCH (Section 5.1), we use the calibrated judge to rank theorem-generation methods, illustrating how a small human calibration set can support scalable judge selection. These results are validation-set calibration evidence, not a claim of universal reliability or deployment-time generalization.

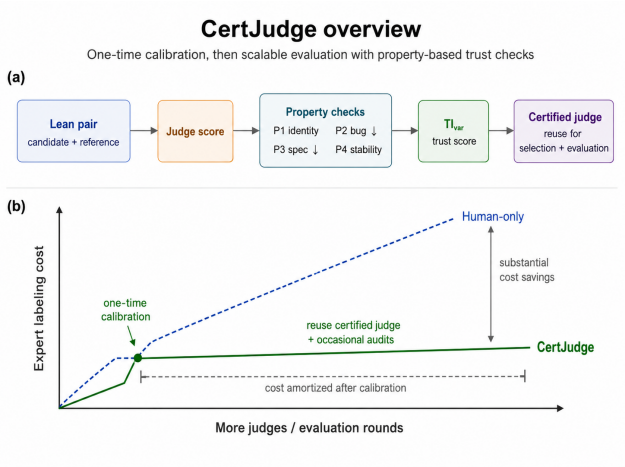


Figure 1. **Scalable trust calibration for LLM judges.** A small human-labeled calibration set is paired with falsifiable property checks (P1–P4) to compute $\mathrm{TI}_{\mathrm{var}}$, enabling judge selection and evaluation with reduced relabeling overhead.

1. Introduction

Agentic AI systems increasingly generate both code and the trust artifacts meant to evaluate that code: tests, specifications, and formal theorems. In Lean 4, these artifacts are code-adjacent formal objects: runnable programs paired with theorem or specification statements and, sometimes, proofs that those statements hold. In classical autoformalization, the target theorem is supplied, so evaluation can often reduce to proof success. In agentic verification, the system must decide *what should be true* before it can prove anything. The Lean kernel can check a proof, but it cannot tell us whether the proposed theorem is complete, semantically aligned with the intended program, or merely a weak statement that happens to typecheck. As agents produce more candidate artifacts than humans can inspect, theorem-quality evaluation becomes a code-evaluation and scalable-oversight problem (Bowman et al., 2022; Zhou et al., 2025; Khattab et al., 2023; Cheng et al., 2024).

The standard reliability check is per-judge human validation: collect labels, measure agreement between each judge and human ratings, and keep the judge if the correlation is high enough. That check is necessary, but it scales linearly with the number of judges, prompts, model updates, thresholds, and domains under consideration. Every new model family or prompt revision asks for another round of annotation, so direct human validation becomes untenable at agent-

[*]Equal contribution [1]Stanford University [2]Hong Kong Baptist University. Correspondence to: Ethan S Hersch <ehersch@stanford.edu>.

The 3^{rd} *AI for Math Workshop* at the 43^{rd} International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.

generation rates. Humans remain the anchor; the problem is that re-using humans from scratch for every judge wastes the expensive part of the process.

We focus on a core *AI for math* bottleneck: code agents can generate candidate formal artifacts far faster than humans can inspect them. In this setting, LLM judges are useful only if their reliability can be measured scalably rather than assumed.

Validating LLM judges against humans is the standard reliability check but does not scale: every new judge, prompt, or model needs fresh annotations. Property certification **amortizes** this cost—gold references collected once, used to calibrate a four-property trust metric, then deployed without re-annotating every new judge configuration. In our setting, humans rate a compact validation set once (n=75 aligned pairs in this proof-of-concept run), and the calibrated property suite extends that trust signal to new judges through automated property checks; the research question is whether the *method* (property certification + a variance-aware index) tracks human agreement, not whether this single weighting is the final word for all domains. The four properties are concrete: a judge should give maximal scores to identical artifacts, lower scores as semantic bugs accumulate, lower scores as theorem/test specifications are removed, and stable scores under prompt rewordings and semantics-preserving rewrites. On a panel of eight judge configurations, $\mathrm{TI}_{\mathrm{var}}$ fits the pooled calibration corpus with validation-set Spearman coefficients of 0.833 (avg labels), 0.905 (pass1), and 0.738 (pass2); held-out generalization to new raters, benchmarks, and judge families remains future work.

Our **contributions** are:

1. **An amortized trust-validation protocol:** we formalize property certification as one-time calibration on a fixed panel of Lean artifacts with perturbations. After calibration, any new judge configuration can be screened by computing four falsifiable properties, without repeating the full human-labeling budget.

2. **Proof-of-concept calibration on n=75:** $\mathrm{TI}_{\mathrm{var}}$ is one geometric aggregation of diagnostics that tracks human agreement on judges; we report correlations (avg $=$ 0.833, pass1 $=$ 0.905, pass2 $=$ 0.738 Spearman) as evidence toward amortized oversight feasibility, *not* as universally fixed aggregation weights or constants.

3. **End-to-end use on VERIBENCH:** the same calibrated judge ranks agent-generation methods consistently with benchmark expectations (e.g., DSPy ReAct 0.615 vs. baseline prompting 0.470 on normalized specs), illustrating how trust signals propagate to scalable code/spec evaluation workloads.

2. Related Work

LLM-as-a-Judge. LLMs are increasingly used as *automated evaluators* across open-ended tasks. In prominent LLM-as-a-judge work, reliability is commonly validated by agreement with people. G-EVAL reports Spearman correlation with human scores (Liu et al., 2023). MT-Bench reports GPT-4 judge agreement with human preference labels (Zheng et al., 2023). Judge-Bench evaluates whether LLMs replicate existing human annotations across 20 NLP datasets (Bavaresco et al., 2024). These studies make human agreement the standard validation target. Our contribution keeps that target but amortizes it: one human calibration panel selects falsifiable property checks that screen future judge configurations without new labels. However, large-scale audits caution that reliability varies by task and setup, and stress the need for explicit validation before replacing human judgment (Bavaresco et al., 2024; Li et al., 2024). Common biases include position and verbosity effects and sensitivity to prompt phrasing (Zheng et al., 2023; Li et al., 2024).

Improving judge reliability. To mitigate biases and instability, multi-agent debate frameworks such as *ChatEval* emulate panel-style review to yield more robust consensus scores (Chan et al., 2024). Structured prompting like *G-Eval* elicits rubric-driven, stepwise reasoning prior to scoring, improving agreement with human criteria on NLG tasks (Liu et al., 2023). Reliability can also be quantified via repeated scorings and internal consistency measures (e.g., Cronbach's α) to detect prompt/run sensitivity (Li et al., 2024).

Code and formal-spec evaluation. For programming, single-pass metrics like pass@k can be brittle or unavailable; early LLM-judge approaches such as *ICE-Score* highlighted difficulty distinguishing partial vs. full correctness from a one-shot review (Zhuo, 2024). *CodeJudge* operationalizes a two-stage "analyze-then-summarize" procedure with a taxonomy of code faults, significantly improving correlation with functional ground truth (Tong & Zhang, 2024). In formal verification/autoformalization, prior work typically provides fixed target theorems, reducing evaluation to proof success. In contrast, agentic systems must propose specifications, necessitating *semantic* judges. We adopt a certification protocol (akin to metamorphic testing) and verify that our LLM judge obeys *identity/consistency*, *monotonicity* with respect to injected bugs and specification omissions, and *prompt/rewrite stability*, directly answering recent calls for validated, trustworthy LLM evaluators in safety-critical domains (Bavaresco et al., 2024; Li et al., 2024).

Evaluation Consistency. Recent work has shown that unconstrained chain-of-thought reasoning can reduce evaluation consistency, and that simple comparative prompting often outperforms complex multi-step pipelines (Chen et al., 2026). We interpret these findings as evidence that stability, rather than reasoning complexity, is the primary driver of reliable grading. Our work formalizes this intuition by introducing falsifiable structural diagnostics—identity, monotonicity under controlled edits, and reliability under prompt variation—that aggregate into a graded trust signal aligned with human judgments.

3. Evaluating Lean Theorem/Specification Quality

Task. Each benchmark item contains a natural-language problem statement, a gold Lean 4 artifact g, and a candidate artifact x produced by an AI agent. The artifact may include code, tests or specifications, and a target theorem/specification statement. Our goal is to assign a score $J(x, g) \in [0, 1]$ reflecting how well the candidate captures the intended formal specification represented by the gold artifact. For non-formal-methods readers, a Lean artifact is a code file in a programming language and proof assistant where executable code, tests, definitions, and theorem statements can live together.

Why this is nontrivial. Lean can check whether a concrete proof script type-checks, but type-checking alone does not determine whether x formalizes the same intended property as g. A candidate may compile while being underspecified, over-constrained, vacuously true, or semantically shifted. In VERIBENCH, this often appears as a coverage gap: candidate artifacts may capture only part of the behavior specified by the gold reference (Miranda et al., 2026). Thus, exact automatic scoring of theorem/specification quality is unavailable in general.

Judge model. We therefore use an LLM judge parameterized by

$$\theta = (\text{model}, \text{prompt}, n_{\text{repeats}}, \text{aggregation}),$$

which maps a pair (x, g) to a score $J_\theta(x, g) \in [0, 1]$. The central question is not only how to build such a judge, but when it is trustworthy enough to rank systems. Rather than collecting new human labels for every θ, Section 4 tests whether J_θ satisfies falsifiable behavioral constraints under controlled perturbations.

4. Property Certification Protocol

4.1. Calibration Protocol

We use property certification as a one-time calibration protocol. Humans first label a fixed reference panel; here, $n = 75$ aligned candidate–gold pairs are rated by one annotator under two rubric strictness levels. For any new judge configuration, we then compute property scores on scripted perturbations of Lean artifacts and aggregate them into a trust index, without collecting new human labels.

The judge compares a candidate Lean 4 artifact against its gold reference and returns a discrete score $s \in \{0, 1, \ldots, 10\}$, later normalized to $[0, 1]$. The trust index is built from four property scores: ident (P1 identity/consistency), mono (P2 bug monotonicity), weak (P3 specification-ablation monotonicity), and stab (P4 repeat/prompt stability). These are graded calibration targets, not binary gates. As simple first-pass baselines, we also consider arithmetic variants $v1$ and $v2$ that aggregate these property scores without explicit variance penalties; Appendix G.5 gives the exact formulas. These are useful comparators, but later results show that geometric and variance-aware formulations are more informative.

Primary metric.

$$\mathrm{TI}_{\text{var}} = \exp\Big(\tfrac{1}{3}\log C_f + \tfrac{1}{3}\log(1 - \widetilde{\mathrm{VarCorr}}) + \tfrac{1}{3}\log(1 - \widetilde{\mathrm{VarScore}})\Big),$$

where C_f is the geometric aggregate of the four property scores and $\widetilde{\mathrm{VarCorr}}$, $\widetilde{\mathrm{VarScore}}$ are normalized variance penalties; Appendix G.5 gives the exact definitions. Our calibration claim is simple: judges with higher trust-index scores should also have higher human agreement on the validation panel. The equal-weighted form above is a simple variance-aware instantiation; other task-specific calibrations are possible, but we treat TI_{var} as a validation-set trust signal rather than a universal constant.

To be trustworthy, the judge should satisfy the following falsifiable calibration targets.

P1 — Identity & Consistency. The judge should assign the maximum score to an artifact compared with itself: $s(X, X) = 1$.

P2 — Monotonicity to Bugs. Scores should decrease as semantic bugs are cumulatively introduced.

P3 — Monotonicity to Missing Specs. Scores should decrease as specification elements (theorems/tests) are removed.

Step	Action
1	Collect a diverse set of human ratings on candidate-vs.-gold Lean artifacts (n=75 in our calibration).
2	For each candidate judge, evaluate identity, bug-monotonicity, spec-monotonicity, and stability on Lean artifacts subjected to agent-scripted perturbations; grading these properties uses only automated ladders and judge runs on perturbed artifacts—*not* pairwise human-vs.-gold labels on those variants.
3	Combine how well properties are satisfied with explicit instability penalties into the variance-aware trust index $\mathrm{TI}_{\mathrm{var}}$.
4	Prefer judge tuples with higher $\mathrm{TI}_{\mathrm{var}}$ for larger-scale code/spec judging and deployment.

Table 1. **Property-certification workflow.** A one-time human panel calibrates the relationship between falsifiable judge properties and human agreement; subsequent judge configurations can be screened automatically.

P4 — Prompt Stability & Semantic Invariance. Scores should remain stable across repeated calls and under semantics-preserving rewrites. In particular, ALPHA-renaming changes only bound variable names—for example, renaming local theorem variables or lambda binders without changing the proof or program structure. In ordinary programming terms, this is the kind of edit where a loop variable or local placeholder gets renamed but the program still means exactly the same thing. These edits should preserve Lean semantics exactly, so a robust judge should score the original and renamed artifact almost identically.

We evaluate these targets with rank-based monotonicity statistics (Kendall's τ, Spearman's ρ) and pairwise ordering accuracy, following prior judge-evaluation work (Tong & Zhang, 2024; Zhuo, 2024).

5. Experiments

5.1. Data and human-validation protocol

All experiments use VERIBENCH (Miranda et al., 2026).

Overview of VERIBENCH. Concretely, VERIBENCH consists of 857 theorems divided into five subsets:

1. **HumanEval** – 387 theorems derived from 56 standard programming puzzles from Chen et al. (2021);
2. **EasySet** – 278 theorems extracted from 41 introductory logic and programming tasks;
3. **CSSet** – 68 theorems drawn from 10 classical data-structure and algorithm problems;
4. **SecuritySet** – 112 theorems taken from 28 examples of buffer overflows, privilege escalation, and race-condition labs based on real code;
5. **RealCodeSet** – 12 theorems extracted from 5 Python standard library programs, used to test model performance on production-grade code.

Identity Variant	Mean Score	Mean Within-Item Std
RENAME	0.936	0.005
REORDER	0.849	0.029
STYLE	0.982	0.001
ALPHA	0.399	0.096

Table 2. **P4 semantic-invariance breakdown for the Codex API run.** Variant-wise score behavior under semantics-preserving rewrites. The weakest mode is aggressive ALPHA-renaming.

We map these to shorthand subset names **Easy**, **CS**, **HumanEval**, **Security**, and **RealCode**. This design spans simple correctness routines through security-relevant invariants and excerpts of production library code.

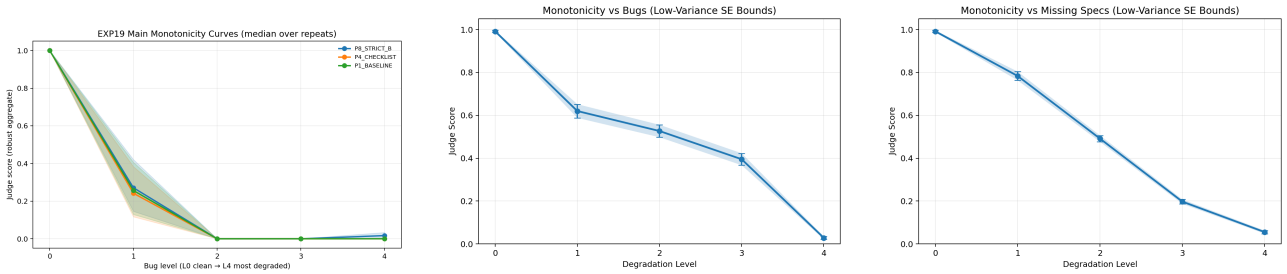
For judge-validation, we use an aligned panel of $n = 75$ candidate–gold pairs from 24 problems with 3 agent generations per problem. Each pair receives two human passes on a 0–5 rubric, and we report pass1, pass2, and averaged labels.

The same aligned panel is reused across judges, so differences in trend reflect judge behavior rather than row selection. Human-pass dispersion is non-trivial (Appendix Figure 12), which is why we report all three label views rather than a single pass.

5.2. Property sanity checks on GPT-5.3-Codex

Establishing trust in the judge. Figure 2 shows that the Codex judge satisfies the intended qualitative behavior: scores decrease as semantic bugs accumulate and as specifications are removed, while identity and rewrite robustness are handled separately through P1/P4 diagnostics. The strongest signal is specification-ablation monotonicity ($\rho = -0.899$, $r = -0.892$); bug monotonicity is also directionally correct but noisier ($\rho = -0.808$, $r = -0.687$), indicating occasional local reversals. Appendix Figures 6 and 7 show that the same monotone degradation pattern is also visible for Claude Opus 4.7 under matched plotting protocols.

These sanity checks support using the judge as a semantically meaningful proxy for theorem/specification quality, while also making the main failure mode explicit: aggressive ALPHA-renaming remains substantially less stable than milder rewrites. Rankings under repeat-resampled scores are stable enough to aggregate into a trust index, but ALPHA-renaming stays the dominant identity gap (Figure 13).



(a) Average score under increasing degradation

(b) P2: Monotonicity vs. Bugs (Mean ± SE)

(c) P3: Monotonicity vs. Missing Specs (Mean ± SE)

Figure 2. **LLM Judge Trustworthiness Sanity Checks.** VeriBench-wide GPT-5.3-Codex API run ($n_{\text{files}} = 184$, $n_{\text{repeats}} = 3$). **(a)** Low-variance rendering emphasizes smooth non-increasing score behavior under controlled degradations. **(b)** P2 bug-ladder trend remains directionally correct with local reversals (Pearson $r = -0.687$, Spearman $\rho = -0.808$, weak monotonicity 52.2%). **(c)** P3 missing-spec trend is stronger and more stable (Pearson $r = -0.892$, Spearman $\rho = -0.899$, weak monotonicity 89.1%). Strict P1 identity $s(X, X)$ and P4 rewrite invariance are reported separately in Table 2.

Model Family	pass1	pass2	avg
GPT-5.3-Codex	0.905	0.738	0.833
Claude Opus 4.7	0.810	0.595	0.643
Qwen3-Coder	0.476	0.905	0.905

Table 3. **Judge-level $\mathrm{TI}_{\mathrm{var}}$ → human alignment.** Spearman correlation between judge-level $\mathrm{TI}_{\mathrm{var}}$ and judge-level human Spearman agreement on the aligned panel. Pass-specific values differ across rubric severity; the qualitative pattern is stable across families.

Protocol	Fit Spearman	Test Spearman
pass1 → pass2	0.881	0.810
pass2 → pass1	0.857	0.786

Table 4. **Cross-pass robustness.** *Fit Spearman*: judge-level Spearman$(\mathrm{TI}_{\mathrm{var}}$, human alignment$)$ when weights are tuned on one pass. *Test Spearman*: the same statistic on the other pass.

5.3. $\mathrm{TI}_{\mathrm{var}}$ predicts judge-level human alignment

A central question for our protocol is whether the primary trust metric $\mathrm{TI}_{\mathrm{var}}$ yields a calibrated signal that aligns with how humans judge candidate Lean artifacts under the same rubric. We do *not* claim that $\mathrm{TI}_{\mathrm{var}}$ certifies universally correct evaluator behavior—no finite scripted suite can enumerate all plausible failure modes (Section 6.2).

Across Codex, Claude Opus, and Qwen3-Coder, judges with higher $\mathrm{TI}_{\mathrm{var}}$ also have higher correlation with human ratings. Figure 3 plots judge-level $\mathrm{TI}_{\mathrm{var}}$ against human Spearman agreement (averaged validation labels) separately for those three judge families while holding fixed the 75 overlapping human-aligned rows that define the calibration panel within each regression. $\mathrm{TI}_{\mathrm{var}}$ does not only reward average property satisfaction—it also penalizes instability across repeats, prompts, and perturbations—so the signal targets configurations that behave more coherently on the perturbation suite used for calibration.

Table 3 reports the same relationship for pass1, pass2, and averaged labels. Codex is strongest on pass1 and avg, Qwen strongest on pass2 and avg, and Claude Opus stays positive on all three. The point is not that one family dominates every split; it is that the same certification signal remains informative across distinct judge families. Most coefficients are large in magnitude—six of nine family/pass combinations exceed 0.7.

However, because the judge-level trend is estimated over a small panel, we treat the effect size as calibration evidence and report cross-pass and cross-family robustness rather than claiming deployment-ready generalization.

For the Codex calibration panel ($n = 8$ judges), Pearson shows the same qualitative pattern (avg $r = 0.774$, pass1 $r = 0.661$, pass2 $r = 0.837$), while Kendall τ is noisier at this scale (avg $\tau = 0.571$, pass1 $\tau = 0.571$, pass2 $\tau = 0.643$).

Thus $\mathrm{TI}_{\mathrm{var}}$ behaves as a calibrated reliability signal: falsifiable properties correlate with the quantity we care about—agreement with human assessments of Lean theorem and specification quality—so new judge configurations can be compared by automated property checks after the calibration relationship is measured once.

Human-pass dispersion and cross-pass robustness. Annotators scored each aligned pair twice under different rubric strictness. Agreement between passes is moderate (Spearman $= 0.249$, Pearson $= 0.375$, quadratic-weighted kappa (QWK) $= 0.372$, MAE $= 0.778$ on the 0–5 scale), so judge–human numbers are calibration evidence rather than ceiling performance. Table 9 gathers full diagnostics.

Table 4 shows that test Spearman stays above 0.7 in both directions when $\mathrm{TI}_{\mathrm{var}}$ is fit using one human pass alone.

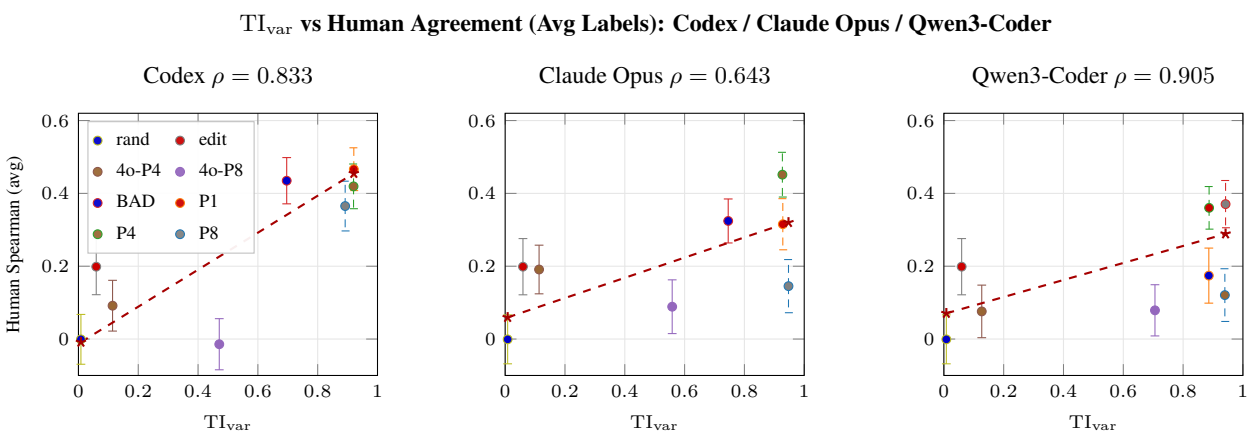


Figure 3. $\mathrm{TI}_{\mathrm{var}}$ **vs. judge-level human alignment.** Each point is one judge configuration; the x-axis is $\mathrm{TI}_{\mathrm{var}}$, and the y-axis is Spearman agreement with averaged human labels on the validation panel. Vertical bars show approximate bootstrap SE; dashed lines are OLS guides. Across models, higher certification strength tracks higher human alignment.

Model	v1	v2	GM	$\mathrm{TI}_{\mathrm{var}}$
Codex	0.762	0.476	0.857	0.833
Claude Opus	0.429	0.143	0.714	0.643
Qwen3-Coder	0.524	0.500	0.786	0.905

Table 5. **Avg-label judge-level trends by trust-index formulation.** Spearman correlation between index score and per-judge human Spearman on averaged labels. $v1$ and $v2$: arithmetic variants; GM: geometric monotonicity/stability; $\mathrm{TI}_{\mathrm{var}}$: variance-aware index.

Threshold	Flip Rate (95% CI)	Pass Rate (95% CI)
0.60	0.058 $[0.048, 0.069]$	0.583 $[0.559, 0.608]$
0.70	0.049 $[0.040, 0.059]$	0.559 $[0.537, 0.581]$
0.80	0.051 $[0.042, 0.061]$	0.544 $[0.523, 0.566]$
0.85	**0.046** [**0.037**, **0.057**]	0.536 $[0.515, 0.557]$

Table 6. **Threshold calibration summary.** Calibration yields a modest but consistent reduction in decision flips at stricter thresholds, with moderate coverage loss. Ranking behavior is unchanged across thresholds ($\rho_{\text{bug}} = -0.790$ at all t).

5.4. Why $\mathrm{TI}_{\mathrm{var}}$ is the primary trust metric

Figure 3 with Table 3 states the headline cross-family trend. Table 5 then asks a narrower question: which trust-index formulation best captures that trend on averaged labels? Relative to arithmetic baselines, $\mathrm{TI}_{\mathrm{var}}$ uniformly improves over $v2$ for every family: Codex $0.476 \to 0.833$, Claude Opus $0.143 \to 0.643$, Qwen3-Coder $0.500 \to 0.905$. The geometric index (GM) is competitive for Codex and Opus (0.857 and 0.714 respectively), while $\mathrm{TI}_{\mathrm{var}}$ stays most consistently strong across families and is the strongest formulation for Qwen. We optimize for cross-judge consistency rather than single-family peak correlation. Figure 8 provides a baseline-vs-geometric visualization on an expanded judge panel.

Takeaway for calibrated code evaluation. The main empirical claim is not that any finite property suite certifies judge correctness. Rather, the claim is that this set of falsifiable diagnostics can predict which LLM judges agree more strongly with human assessments of formal-code artifacts. This matters for AI for math because code agents can generate far more tests, specifications, and theorems than humans can inspect, while a one-time human calibration panel plus automated property checks provides a scalable way to compare judges before deploying them on evaluations.

Property-level interpretation. For averaged labels, per-property correlations identify **P3 weak monotonicity** as strongest (Spearman 0.747, $p = 0.043$; Pearson 0.744, $p = 0.048$), with **P2 bug-rank monotonicity** next (Spearman 0.714, $p = 0.056$; Pearson 0.636, $p = 0.037$). P1 identity and P4 stability are not strong standalone predictors at this $n_{\text{judges}} = 8$ scale. Across pass1/pass2/avg, P3 is consistently top-ranked, indicating that semantic monotone behavior under controlled degradations is the most stable predictor of validation-set human Spearman in this study.

5.5. Threshold calibration: stability vs. coverage

Decision-threshold calibration. After validating score-level behavior, we calibrate a binary operating point for deployment. A threshold t converts a continuous judge score into a yes/no decision, and flip rate measures how often repeated judge calls on the same item disagree after thresholding. Across $t \in \{0.60, 0.65, 0.70, 0.75, 0.80, 0.85\}$, ranking behavior is essentially unchanged ($\rho_{\text{bug}} = -0.790$ throughout), while stricter thresholds modestly reduce flip rate at the cost of lower coverage. We therefore select $t = 0.85$, which yields the lowest observed flip rate while keeping pass rate near 0.5 (Table 6).

5.6. Failure modes: alpha-renaming and local reversals

The main failure mode remains the same throughout the study: alpha-renaming is substantially less stable than milder rewrites, and local monotonicity reversals can still occur even when global trends are strong. Alpha-renaming is intentionally a weak perturbation—it should leave theorem quality unchanged—so failures here are informative about superficial sensitivity rather than true semantic degradation. At the same time, it is a hard robustness target, because changing binder names can alter token patterns and local readability cues even when Lean semantics are unchanged. On an expanded 13-judge panel, the certification trend remains above 0.75 across label sets, suggesting that the effect is not specific to a minimal prompt panel. Detailed per-model diagnostics are reported in the appendix.

Agent	Easy	CS	Real	HE	Sec	Avg
Baseline	0.580	0.690	0.472	0.410	0.347	**0.470**
DSPy ReAct	0.659	0.754	0.580	0.650	0.454	**0.615**
Trace+	0.342	0.680	0.592	0.573	0.411	**0.477**
Trace++	0.620	0.756	0.504	0.645	0.432	**0.588**

Table 7. **Theorem quality scores on VeriBench.** Average normalized scores by split: Easy, CS, Real, HumanEval, and Security. All rows use Claude 3.7 as the agent model and as the LLM judge. Trace+ denotes the trace self-debug agent and Trace++ the self-debug and self-improve agent. The DSPy ReAct agent only performs self-debug. The LLM judge issues 5 scoring calls per file pair and returns an integer rubric score in $\{0, 1, \ldots, 10\}$. Normalized scores are computed by dividing by the maximum score (10) and averaging across all examples. The Avg Norm Score shows the overall average over the entire benchmark.

5.7. Downstream VeriBench agent ranking

The calibrated judge is useful not only as a validation object but also as an evaluation tool. Table 7 shows that it ranks agentic theorem-generation methods in the expected direction: the weakest baseline is simple prompting (0.470 average normalized score), while feedback-driven methods score substantially higher, led by DSPy ReAct (0.615) and followed by Trace++ (0.588). We use this only as a downstream sanity check, not as independent proof of judge correctness. The result still matters because it shows that the same calibrated judge that aligns with humans on the validation panel also produces sensible system-level rankings on VeriBench (Khattab et al., 2023; Cheng et al., 2024; Chen et al., 2021).

6. Limitations and Held-Out Test Protocol

We study an LLM judge that *compares each candidate artifact against its task's gold reference* for Lean 4 code verification alongside a graded trust signal built from falsifiable perturbation checks. Our experiments have access to gold references, yet this certification protocol can be used for broader Lean formalization settings.

The judge satisfies falsifiable score-level properties (identity and monotonicity), and we additionally calibrate its binary decision boundary using full-dataset threshold sweeps with uncertainty estimates. Our data supports a clear separation: score ranking remains stable under threshold changes, while binary stability and coverage trade off in a tunable way. In experiments, calibrated judge scoring produces informative method rankings and tracks engineering signals (Pass@k, unit-test accuracy); Easy and HumanEval show the strongest agreement, while Security and RealCode remain hardest (Chen et al., 2021).

6.1. Validation vs. test protocol

The results in Section 5.3 are validation-set numbers. The same human labels were used to choose the trust-index form, repeat aggregation, threshold, and prompt panel, so the reported trends should be interpreted as calibration findings rather than out-of-sample generalization claims.

Beyond the $n = 75$ panel used in Section 5.3, which was scored under two closely related rubric prompts corresponding to stricter and more lenient rating views, we have now collected a substantially broader independent annotation pool. This newer pool contains five reviewers and **343** scored labels over 78 distinct candidate–gold pairs, with many pairs receiving 4 or 5 independent ratings. We do not use those labels in the present paper's calibration numbers. Instead, the intended next step is a locked held-out analysis: freeze the current property definitions and $\mathrm{TI}_{\mathrm{var}}$ formulation, aggregate the independent reviewer labels with noise-aware estimators where appropriate, and test whether the judge-level $\mathrm{TI}_{\mathrm{var}}$ trend persists under broader rubric variation and multi-rater supervision.

This corpus directly addresses the concern that the present $n = 75$ panel may overfit to a narrow rubric interpretation and provides a path from calibration evidence to locked-test generalization.

An encouraging sign is that a separate expanded 13-judge analysis remains above 0.75 across label sets (Appendix Figure 8), suggesting that scaling the judge panel is promising even though it does not yet establish held-out generalization.

6.2. Finite perturbations, specification-relative scoring, and release details

We treat $\mathrm{TI}_{\mathrm{var}}$ as a graded trust signal rather than a binary pass/fail gate. Individual properties are necessary conditions on the perturbations we test (a judge that scores $X = X$ at less than 1.0 has, by definition, failed identity), but no finite

property suite can certify all possible judge failures. TI is multiplicative, so a single low-scoring property collapses the index, but it does not impose a hard threshold. The empirical question we answer is whether higher-TI judges are more validation-human-aligned in expectation; the answer ($\rho = 0.881$ across 8 judges) is supportive but graded, and generalization to held-out humans and judges is left to future work.

ALPHA-renaming robustness. Among semantics-preserving rewrite probes, aggressive ALPHA-renaming (systematically changing bound variable and binder names while leaving Lean-semantics intact) is systematically the hardest for our judges: mean scores and score stability remain much worse than under milder RENAME/REORDER/STYLE edits (Table 2 and appendix failure-mode plots). We interpret this as a mismatch between what the kernel guarantees and what LLM judges implicitly condition on, not as evidence that Lean has mis-checked the file. Tokenized programs under ALPHA can look far from both the gold reference and typical training text, so lexical-similarity cues, local naming conventions, and attention to specific binder strings can shift scores even when equivalence is obvious to a human expert. Because the perturbation is intentionally shallow—it does not add a machine certificate of α-equivalence to the prompt—the judge must infer invariance from surface patterns alone, which current models only partially do. Failures may therefore mix (i) genuine insensitivity to equivalence with (ii) brittle dependence on incidental string features, and distinguishing the two is itself non-trivial future work (e.g., canonicalization, explicit α-conversion checks, or enrichment of the judge prompt with kernel-provided structural signals).

Limitations. The judge is a *proxy*: scores are prompt/rubric-sensitive and do not constitute kernel-level proof checking; they provide evidence against a rigorous yet necessarily incomplete gold specification since no complete mechanical checks of specification equivalence exist (Rice, 1953). Our certification relies on a finite perturbation suite, so untested failure modes may persist. Specifications that are exotic to this dataset may be correct but receive incorrect scores. ALPHA-renaming sensitivity is the standout P4 weakness in our runs, as detailed in the preceding paragraph. Calibration improvements are modest in absolute magnitude and do not by themselves establish external validity.

The primary validation corpus in this draft remains the $n = 75$ aligned panel scored under two closely related rubric prompts, yielding stricter and more lenient label views across eight judge tuples. A much larger independent reviewer pool has now been collected—five reviewers, **343** scored labels, and 78 distinct candidate–gold pairs—but it is intentionally held out from the reported calibration results. This new corpus is recent and not yet analyzed in the present submission; its purpose is to support a frozen held-out test of whether the current $\mathrm{TI}_{\mathrm{var}}$ formulation continues to predict human agreement under broader item coverage and multi-rater aggregation. The current validation-set human-alignment results should therefore be read as calibration evidence because judge, prompt, aggregation, threshold, and reporting choices were developed with this label set in view.

BAD_STRINGY, an intentionally bad baseline, achieves moderate validation-set human Spearman (≈ 0.45) while failing weak monotonicity (12.5%). This is a feature of the calibration protocol, not a bug: aggregate validation-set human Spearman alone does not detect this judge's monotonicity failure, but $\mathrm{TI}_{\mathrm{var}}$ does. Future work will expand independent raters and problem coverage, strengthen semantic equivalence handling (candidate $\Rightarrow$ gold postcondition checks), add adversarial judge stress tests, and introduce standardized "judge cards" for reproducible calibrated deployment (Zhou et al., 2025). A concrete model-side target is Leanstral: Appendix Table 11 shows that it remains low-alignment under all three index constructions we tested, and understanding how to make Lean-specialized judges work well for calibrated pairwise evaluation is an explicit next step. Compositional certification—extending local property checks to bounded edit sequences via edit-graph induction—is a natural future-work direction, but not a claimed result here.

7. Conclusion

We present an amortized human-validation protocol for LLM judges based on falsifiable properties. Rather than re-labeling every new judge configuration, we calibrate once with human labels and then score new judges via property diagnostics. Our main empirical result is that this property-based trust index tracks judge-level human alignment on the validation panel, with strong signals from monotonicity and stability. The key next step is a locked held-out test with independent annotators and unseen judge families.

Impact Statement

This work aims to improve the reliability and transparency of automated evaluation for AI-generated formal-code artifacts. More reliable judge calibration can reduce manual review burden, but over-reliance on imperfect proxy judges could also misrank systems or obscure specification errors. We therefore frame CERTJUDGE as a diagnostic calibration protocol rather than a replacement for human or kernel-level validation.

References

Ahuja, R., Avigad, J., Tetali, P., and Welleck, S. Improver: Lean-aware chain-of-states prompting for proof optimization, 2024. URL https://arxiv.org/abs/2410.04753.

Aniva, L., Sun, C., Miranda, B., Barrett, C., and Koyejo, S. Pantograph: A machine-to-machine interaction interface for advanced theorem proving, high level reasoning, and data extraction in lean 4, 2024. URL https://arxiv.org/abs/2410.16429.

Bach, R. et al. A systematic literature review of user trust in AI-enabled systems: An HCI perspective. *International Journal of Human–Computer Interaction*, 2022.

Bavaresco, A., Bernardi, R., Bertolazzi, L., Elliott, D., Fernández, R., Gatt, A., Ghaleb, E., Giulianelli, M., Hanna, M., Koller, A., Martins, A. F. T., Mondorf, P., Neplenbroek, V., Pezzelle, S., Plank, B., Schlangen, D., Suglia, A., Surikuchi, A. K., Takmaz, E., and Testoni, A. Llms instead of human judges? a large scale empirical study across 20 nlp evaluation tasks, 2024. URL https://arxiv.org/abs/2406.18403.

Bowman, S. R. et al. Measuring progress on scalable oversight for large language models. In *arXiv preprint arXiv:2211.03540*, 2022.

Bursuc, S., Ehrenborg, T., Lin, S., Astefanoaei, L., Chiosa, I. E., Kukovec, J., Singh, A., Butterley, O., Bizid, A., Dougherty, Q., Zhao, M., Tan, M., and Tegmark, M. A benchmark for vericoding: formally verified program synthesis, 2025. URL https://arxiv.org/abs/2509.22908.

Chan, Z. C. et al. Chateval: Towards better llm-based evaluators through multi-agent debate, 2024. URL https://arxiv.org/abs/2308.07201.

Chen, E., Gulati, A., Miranda, B., Tang, Z., and Koyejo, S. Rethinking LLM judges: Chain-of-thought and multi-step pipelines for math grading. In *ICLR 2026 Workshop on Logical Reasoning of Large Language Models*, 2026. URL https://openreview.net/forum?id=vdXPorr099.

Chen, M. et al. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Cheng, C.-A., Nie, A., and Swaminathan, A. Trace is the next autodiff: Generative optimization with rich feedback, execution traces, and llms, 2024. URL https://arxiv.org/abs/2406.16218.

Denisov-Blanch, Y., Kazdan, J., Chudnovsky, J., Schaeffer, R., Guan, S., Adeshina, S., and Koyejo, S. Consensus is not verification: Why crowd wisdom strategies fail for LLM truthfulness, 2026. URL https://arxiv.org/abs/2603.06612.

Ding, Z. et al. Citations and trust in LLM-generated responses. In *AAAI*, 2025.

Finkelstein, M., Deutsch, D., Riley, P., Juraska, J., Kovacs, G., and Freitag, M. From jack of all trades to master of one: Specializing LLM-based autoraters to a test set. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 17221–17238. PMLR, 2025. URL https://proceedings.mlr.press/v267/finkelstein25a.html.

Hennigen, L. T., Shen, Z., Nrusimha, A., Gapp, B., Sontag, D., and Kim, Y. Towards verifiable text generation with symbolic references. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=fib9qidCpY.

Kenton, Z., Siegel, N. Y., Kramár, J., Brown-Cohen, J., Albanie, S., Bulian, J., Agarwal, R., Lindner, D., Tang, Y., Goodman, N. D., and Shah, R. Scalable oversight with weak LLMs judging strong LLMs. In *Proceedings of the 38th International Conference on Neural Information Processing Systems (NeurIPS)*, 2024. URL https://arxiv.org/abs/2407.04622.

Khattab, O., Singhvi, A., Maheshwari, P., Zhang, Z., Santhanam, K., Vardhamanan, S., Haq, S., Sharma, A., Joshi, T. T., Moazam, H., Miller, H., Zaharia, M., and Potts, C. Dspy: Compiling declarative language model calls into self-improving pipelines, 2023. URL https://arxiv.org/abs/2310.03714.

Li, S. et al. Can you trust llm judgments? reliability of llm-as-a-judge, 2024. URL https://arxiv.org/abs/2412.12509.

Liu, C., Yuan, Y., Yin, Y., Xu, Y., Xu, X., Chen, Z., Wang, Y., Shang, L., Liu, Q., and Zhang, M. Safe: Enhancing mathematical reasoning in large language models via retrospective step-aware formal verification, 2025. URL https://arxiv.org/abs/2506.04592. Introduces the FormalStep benchmark for step-level verification in Lean 4.

Liu, Y. et al. G-eval: Nlg evaluation using gpt-4 with better human alignment, 2023. URL https://arxiv.org/abs/2303.16634.

Ma, Q., Shen, H., Koedinger, K., and Wu, T. How to teach programming in the ai era? using llms as a teachable

agent for debugging, 2024. URL https://arxiv.org/abs/2310.05292.

Miranda, B., Daruru, S., Hersch, E. S., Zhou, Z., Nie, A., Amrollahi, D., Aniva, L., Mlauzi, I., Acharya, K., Obbad, E., Soylu, D., Kirk, W., Wang, Z. J., Fronsdal, K., Li, Y., Jr, D. P., Kaushik, R., Liu, S., Denisov-Blanch, Y., Dillmann, S., Obstbaum, S., Cuellar, S., Sarracino, J., Schaeffer, R., Tiwari, M., Lee, D., Han, B., and Koyejo, S. Veribench: An end-to-end formal verification benchmark for AI coding agents in lean 4. In *3rd AI for Math Workshop: Toward Self-Evolving Scientific Agents*, 2026. URL https://openreview.net/forum?id=lkL0qnUv3p.

Naik, A. et al. TiCoder: LLM-based test-driven interactive code generation. *IEEE Transactions on Software Engineering*, 2024.

Nie, F., Liu, K. Z., Wang, Z., Sun, R., Liu, W., Shi, W., Yao, H., Zhang, L., Ng, A. Y., Zou, J., Koyejo, S., Choi, Y., Liang, P., and Muennighoff, N. Uq: Assessing language models on unsolved questions, 2025. URL https://arxiv.org/abs/2508.17580.

Ospanov et al. Apollo: Recursive repair with lean feedback boosts minif2f proof success. arXiv preprint, 2023.

Rice, H. G. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 74(2):358–366, 1953. doi: 10.2307/1990888. URL https://doi.org/10.2307/1990888.

Robertson, Z. and Koyejo, S. Let's measure information step-by-step: AI-based evaluation beyond vibes, 2025. URL https://arxiv.org/abs/2508.05469.

Salaudeen, O. and Hardt, M. Imagenot: A contrast with imagenet preserves model rankings, 2024. URL https://arxiv.org/abs/2404.02112.

Salaudeen, O., Reuel, A., Ahmed, A., Bedi, S., Robertson, Z., Sundar, S., Domingue, B., Wang, A., and Koyejo, S. Measurement to meaning: A validity-centered framework for ai evaluation, 2025. URL https://arxiv.org/abs/2505.10573.

Sun, C., Sheng, Y., Padon, O., and Barrett, C. Clover: Closed-loop verifiable code generation, 2024. URL https://arxiv.org/abs/2310.17807.

Tang, Z., Li, Z., Xiao, Z., Ding, T., Sun, R., Wang, B., Liu, D., Huang, F., Liu, T., Yu, B., and Lin, J. Self-evolving critique abilities in large language models, 2025. URL https://arxiv.org/abs/2501.05727.

Tong, Y. and Zhang, C. Codejudge: Evaluating code generation with large language models, 2024. URL https://arxiv.org/abs/2410.02184.

Wang, R., Cheng, R., Ford, D., and Zimmermann, T. Investigating and designing for trust in ai-powered code generation tools. In *The 2024 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '24, pp. 1475–1493. ACM, June 2024a. doi: 10.1145/3630106.3658984. URL http://dx.doi.org/10.1145/3630106.3658984.

Wang, R., Zhang, J., Jia, Y., Pan, R., Diao, S., Pi, R., and Zhang, T. Theoremllama: Transforming general-purpose llms into lean4 experts, 2024b. URL https://arxiv.org/abs/2407.03203.

Wu, T., Jiang, E., Donsbach, A., Gray, J., Molina, A., Terry, M., and Cai, C. J. Promptchainer: Chaining large language model prompts through visual programming, 2022. URL https://arxiv.org/abs/2203.06566.

Yan, H., Latoza, T. D., and Yao, Z. Intelliexplain: Enhancing conversational code generation for non-professional programmers, 2024. URL https://arxiv.org/abs/2405.10250.

Zhang, L., Valentino, M., and Freitas, A. Beyond gold standards: Epistemic ensemble of llm judges for formal mathematical reasoning, 2025. URL https://arxiv.org/abs/2506.10903.

Zheng, L. et al. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023. URL https://arxiv.org/abs/2306.05685.

Zhou, J. P., Staats, C., Li, W., Szegedy, C., Weinberger, K. Q., and Wu, Y. Don't trust: Verify – grounding llm quantitative reasoning with autoformalization, 2024. URL https://arxiv.org/abs/2403.18120.

Zhou, J. P., Arnold, S. M. R., Ding, N., Weinberger, K. Q., Hua, N., and Sha, F. Graders should cheat: privileged information enables expert-level automated evaluations, 2025. URL https://arxiv.org/abs/2502.10961.

Zhuo, T. Y. ICE-score: Instructing large language models to evaluate code. In Graham, Y. and Purver, M. (eds.), *Findings of the Association for Computational Linguistics: EACL 2024*, pp. 2232–2242, St. Julian's, Malta, March 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-eacl.148. URL https://aclanthology.org/2024.findings-eacl.148/.

A. Related Work (Cont.)

Autoformalization. Complementing this line of work, Wang et al. (2024b) present THEOREMLLAMA, a framework aimed at enhancing LLM translation into Lean 4. Drawing on over 100K proof examples from the Mathlib4 library, TheoremLlama employs a novel natural-language-to-formal-language (NL-FL) bootstrapping strategy and iterative proof synthesis. This enables the reuse of verified examples as templates for future translations. The framework achieves 36.48% and 33.61% accuracy on the MiniF2F-Valid and MiniF2F-Test benchmarks, respectively—surpassing GPT-4 by more than ten percentage points on both.

Techniques for Code Verification. IMPROVER (Ahuja et al., 2024) introduces a Lean-aware Chain-of-States prompting loop that integrates retrieval, best-of-n sampling, and iterative correction to rewrite formal proofs with improved properties. By optimizing for metrics such as brevity and readability, ImProver reduces the number of tactics by half, doubles proof readability, and boosts theorem prover acceptance rates by over 80%. In addition, CLOVER (Sun et al., 2024) implements a closed-loop pipeline in which an LLM first generates code, docstrings, and formal annotations, then uses reconstruction-based prompting to enforce consistency across these outputs, and finally applies SMT-based verification to validate correctness. Evaluated on the CloverBench suite of Dafny programs, Clover accepts 87% of correct solutions, rejects 100% of flawed ones, and even uncovers bugs in human-written code—demonstrating the power of hybrid generation-verification pipelines. In a complementary direction, Zhou et al. (2025) improve general-purpose LLM-based graders by augmenting them with "privileged" information such as gold-standard solutions, grading rubrics, and detailed annotations. When necessary, the system provides targeted hints back to candidate models. This approach achieves grading performance on par with or exceeding that of specialized systems—and even expert humans—on difficult programming benchmarks.

Agentic Frameworks and Tools for Code Verification. TRACE (Cheng et al., 2024) proposes *generative optimization*, tuning entire computational workflows—including code, prompts, tool calls, and error signals—by treating execution traces as gradients in the OPTO framework. With a PyTorch-like API and the LLM-based optimizer OptoPrime, it supports diverse tasks such as prompt tuning, debugging, and robot control, rivaling specialized optimizers. Building on modular composition, DSPY (Khattab et al., 2023) treats LLM calls as declarative modules in a computational graph. Users define concise input–output signatures, and DSPy's compiler automatically bootstraps or fine-tunes pipelines using built-in "teleprompters." This enables a few-line programs to outperform expert-crafted prompts in math, QA, and agent workflows. Extending agentic capabilities to formal reasoning, PANTOGRAPH (Aniva et al., 2024) offers a programmatic interface to Lean 4 with support for advanced proof search. It exposes internal proof states and tactics for integration with learning agents, replacing human-facing interfaces with API-level control.

A.1. Related Work: Scalable Oversight

Scalable Oversight with LLM Judges Researchers have proposed multi-agent oversight frameworks such as debate and consultancy, where weaker LLMs act as judges supervising stronger agents (Kenton et al., 2024; Chan et al., 2024). Debate, in which two models argue opposing sides before a judge, consistently improves truth-tracking compared to single-pass QA. Kenton et al. (2024) demonstrate that debate reduces the likelihood of judges being misled in math and code tasks, highlighting it as a scalable oversight mechanism.

Self-Improving Critics. In the absence of stronger supervisors, models can bootstrap judgment skill through self-critique. The Self-Evolving Critic (SCRIT) framework trains LLMs to generate critiques of flawed outputs, validate corrections, and iteratively refine their ability to spot errors. Tang et al. (2025) report improvements of over 10% on critique benchmarks, showing that self-critique fosters scalable, autonomous oversight.

Certifying Judges. Naïve LLM judges are often inconsistent, rewarding stylistic cues over correctness (Bavaresco et al., 2024; Li et al., 2024).

Validity-Centered Evaluation. Certifying judges requires not only behavioral properties but also principled validity evidence. Salaudeen et al. (2025) introduce a framework that distinguishes between measurement, claims, and evidence, grounded in psychometrics. They emphasize construct, content, and criterion validity, and systematically enumerate threats to validity. This framework provides a blueprint for reporting and interpreting judge certification results, complementing identity/monotonicity checks with broader validity cards.

Robust Evaluation without Ground Truth. Standard judge queries can be brittle in the absence of ground truth, particularly under adversarial manipulation (Zhuo, 2024; Tong & Zhang, 2024). Robertson & Koyejo (2025) model the overseer as an agent and propose using information-theoretic, incentive-compatible queries instead of direct ratings. They introduce bounded f-divergence measures such as TVD-MI, which exhibit polynomial robustness to adversarial strategies, unlike unbounded KL-based criteria. This work highlights a pathway toward robust, certifiable judges when explicit ground truth is unavailable.

Single-judge certification vs. peer-prediction consensus. Information-elicitation approaches such as TVD-MI (Robertson & Koyejo, 2025) certify trust by asking multiple judges whether their outputs share information about the same source—fundamentally a consensus-based protocol over a judge population. Property certification is complementary: identity and monotonicity properties evaluate a single judge in isolation, with no agent-population dependency. This makes property certification deployable in settings where consensus is unavailable (single-judge inference, novel domains, sparse model panels), and our calibration protocol amortizes the human-annotation cost that consensus mechanisms also incur.

Consensus aggregation vs. property certification. Denisov-Blanch et al. (2026) stress-test five inference-time aggregation strategies—majority vote, highest-confidence selection, confidence- and prediction-weighted votes, and the Surprisingly Popular algorithm—across multiple benchmarks (HLE, BoolQ, Com2Sense, and Predict-the-Future) and open-source models spanning roughly 4B to 235B parameters; even at $25\times$ the cost of single-sample inference no aggregator consistently beats a single sample in verifier-absent domains, and several amplify shared misconceptions. They trace the failure to structurally correlated LLM errors: in MATH plurality-wrong cases, the most common incorrect answer accounts for 65–87% of wrong responses, and in an out-of-distribution random-ASCII control with no ground truth, different models still agree at Cohen's κ as high as ≈ 0.35, indicating shared inductive biases rather than shared knowledge. A central diagnosis is that self-reported confidence and consensus track *what the crowd will say* more reliably than *what is true*, separating social prediction from truth verification, and the paper concludes that inference-time scaling improves truthfulness only when an external verifier is available. Our work sits on the verifier-present side of that boundary: instead of aggregating across a judge population, we calibrate a single judge against falsifiable properties (P1–P4) over Lean 4-checked gold artifacts and explicitly constructed bug/specification ladders, yielding a calibrated trust metric with no inter-judge consensus dependency. For oracle-free extensions, TVD-MI-style cross-judge information should be used as a robustness diagnostic rather than as a truth proxy, side-stepping the consensus-as-truth failure mode they document; conversely, their negative results sharpen the case for property-based, single-judge certification whenever a verifier or calibrated reference exists.

Specializing judges to a test set vs. certifying judges across configurations. Finkelstein et al. (2025) propose the *Specialist* method, which deliberately abandons cross-test-set generalization and instead overfits a prompted LLM autorater to a single fixed benchmark. For each test example, in-context demonstrations are drawn from historical human ratings of the *same source*, yielding 54–119% character-level F1 gains over XCOMET on the WMT'23/'24 MQM tasks. Their work and ours share a common motivation—amortizing the expensive part of human evaluation rather than re-annotating from scratch—but amortize along orthogonal axes. Specialist amortizes *within* a test set: it reuses many historical ratings (under a pseudo-SxS, single-rater-per-input regime) as inference-time demonstrations, so the judge must be re-specialized for each new benchmark and, as the authors show, to the individual rater. Property certification instead amortizes *across judge configurations*: a one-time calibration panel relates falsifiable behavioral properties (P1–P4) to human agreement, after which any new $(\text{model}, \text{prompt}, n_{\text{repeats}}, \text{aggregation})$ tuple is screened by fully automated perturbation checks, with no human-vs-gold labels required on the perturbed variants. The two approaches are complementary: Specialist still consumes human ratings at scoring time and presumes a curated pool of same-source annotations, whereas our property-certification protocol targets the setting where new judges proliferate faster than any per-test-set rating pool can be maintained. Notably, their finding that a Specialist autorater specializes to a *single* rater echoes our own concern with human-pass dispersion; both works suggest single-rater ground truth is an imperfect ceiling, motivating the multi-rater held-out protocol we defer to future work.

Formal Methods as Oversight. Proof assistants provide binary, kernel-level guarantees. Zhou et al. (2024) autoformalize math solutions into Isabelle/HOL and show that filtering via proof checking outperforms majority voting by 12% on GSM8K. Liu et al. (2025) extend this approach with SAFE, which verifies every intermediate step in Lean 4 and introduces the FormalStep benchmark, reducing hallucinations and enabling fine-grained oversight.

Agentic Iterative Pipelines. Ospanov et al. (2023) integrate Lean feedback into recursive repair cycles, boosting general-purpose LLM proof success from 3–7% to over 40% on miniF2F, and achieving 75% with fewer than 1000 samples. Miranda et al. (2026) extend this paradigm to program verification, showing that self-debugging agents quadruple success compared to one-shot prompting. Incorporating an LLM judge improved completeness on security tasks but required calibration to avoid regressions on simpler cases.

Benchmarks for Judge Reliability. Benchmarks such as VeriBench and FormalStep quantify judge reliability across compilation, proof success, and specification quality. Miranda et al. (2026) report normalized specification scores of 0.470 for one-shot prompting versus 0.615 for self-debugging agents. Outside formal mathematics, MT-Bench and G-Eval provide complementary judge-evaluation settings using pairwise preferences and rubric-guided scoring,

underscoring the need for calibration and validity reporting (Zheng et al., 2023; Liu et al., 2023). Together with validity frameworks and robust information-theoretic scoring, these benchmarks advance the agenda of certifying LLM judges for formal verification.

Evaluating Autoformalization with Reference-Free Judges. In a complementary line of work, Zhang et al. (2025) propose an ensemble of LLM judges for the autoformalization task, using a fine-grained taxonomy to score formalizations without a gold-standard reference. The primary strength of this approach is its reference-free nature, offering a scalable evaluation pathway when gold standards are unavailable or prohibitively expensive to create. However, the work's motivation for abandoning gold-standard comparison is based on limited evidence: a small, non-random sample of "potentially incorrect" formalizations pre-selected by an LLM, whose own reliability is uncertified. Furthermore, the framework validates its judge by correlating with human preference scores—a subjective proxy—rather than the more objective, machine-checkable goal of logical equivalence to a known-correct specification. Critically, even if minor errors exist in gold-standard datasets, recent work has shown that such noise rarely alters the relative ranking of models, as errors are typically correlated across systems (Salaudeen & Hardt, 2024). This suggests that a small number of flawed examples does not invalidate the utility of an entire benchmark for comparative evaluation. Our work, in contrast, proceeds from gold references that Lean kernel-check relative to explicitly stated predicates, and emphasizes *calibration evidence* for pairwise judge scores—the preferred rubric-aligned target when verifier-backed references exist.

A.2. HCI for Solving Scalable Oversight

From more evaluation to *human-centered* oversight. HCI reframes scalable oversight as a *calibration* problem: design interactions that let people verify, steer, and place *appropriate* trust in LLM outputs rather than maximizing raw trust (Bach et al., 2022). Interface choices (e.g., citations, explanations, confidence cues) can substantially shift user trust—sometimes in misleading ways—so the goal is to expose process and embed lightweight verification instead of presenting one-shot answers (Ding et al., 2025). User studies in AI pair-programming similarly surface needs to communicate model performance, provide control over output style/constraints, and expose uncertainty to avoid over-reliance (Wang et al., 2024a).

Hybrid human↔AI evaluation. HCI work operationalizes scalable evaluation by combining expert/lay judgments with automated judges. Humans annotate failures or verify slices of output; systems then train or condition LLM-based "reference evaluators" on those signals to scale up assessment while staying anchored to what users value (accuracy, safety, usefulness). This human-in-the-loop pattern complements LLM-as-judge approaches—whose reliability varies by task and setup (Bavaresco et al., 2024; Li et al., 2024)—and is especially relevant when pass@k or single-shot metrics are brittle.

Interfaces that *operationalize* oversight. HCI systems instantiate oversight as interaction loops that make correctness checkable *during* generation: (i) *test-driven code generation* (TiCoder) elicits yes/no judgments about candidate unit tests before showing code; a few interactions reduced cognitive effort and lifted pass@1 by $\sim$45.7% on coding benchmarks (Naik et al., 2024); (ii) *explain→correct loops* (IntelliExplain) pair each snippet with a natural-language rationale that users can critique, improving novice task success by $\sim$11–25% (Yan et al., 2024); (iii) *pair debugging* (HypoCompass) has humans hypothesize faults while the LLM attempts fixes, yielding $\sim$12% learning gains (Ma et al., 2024); (iv) *multi-step pipelines* (PromptChainer) expose intermediate steps (e.g., tests → verification → proof attempt), enabling user intervention where failures arise (Wu et al., 2022); and (v) *source-grounded verification* (SymGen) highlights claim-level spans backed by exact sources and leaves unsupported text unhighlighted, speeding verification by $\sim$20% and raising user confidence (Hennigen et al., 2024). Collectively, these patterns shift trust from fluency to *verifiable progress*.

Applying HCI patterns to formal specs/theorems. Formal kernels (Lean/Isabelle) already deliver binary checks; HCI contributes the *workflow glue* that helps users evaluate *which* theorem/spec to prove and whether it is complete. Two actionable patterns: (1) *counterfactual spec probes*—UI toggles that remove a postcondition or inject a known bug and display the impact on tests/judge score—help detect missing clauses or overfitting; and (2) *stepwise proof oversight*—surfacing intermediate goals and marking unverified suggestions—keeps the human in control of proof search. In our setting, these patterns interface naturally with a graded, semantics-sensitive evaluator.

A.3. Vericode

Bursuc et al. (2025) presumes well-formed formal specifications and evaluates end-to-end verification under those specs (in Dafny, Verus/Rust, or Lean) reporting verified pass rates. In practice, impactful problems often lack both complete specifications and ground-truth answers. Our VERIBENCH program targets this gap by challenging the *generation* of specifications (from code, tests and documentation) and then *grading* their quality prior to proof search using a Lean-aware LLM judge that we validate via falsifiable prin-

ciples/properties (Identity, Monotonicity, and Prompt Stability/Invariance), addressing known pitfalls of LLM-as-a-judge such as bias and prompt sensitivity (Zheng et al., 2023; Bavaresco et al., 2024; Li et al., 2024). However, our framework has only been tested with a judge with gold references files with specifications but we will explore metrics/methods that don't require this like TVD-MI in (Robertson & Koyejo, 2025).

Beyond spec-first settings, the Unsolved Questions (UQ) paradigm evaluates models on difficult, real-world problems that have neither ground-truth answers nor formal specifications; instead, it relies on validator pipelines and community verification to track progress without oracles (Nie et al., 2025). Taken together, these lines of work shift evaluation from "solve what is already specified" to "specify, calibrate, then solve": vericoding provides end-to-end rigor under given specs, VERIBENCH adds specification synthesis with a calibrated semantic judge, and UQ extends evaluation to specless, answerless settings that require validator-assisted screening. Our contribution is a calibrated, semantics-sensitive judge that enables scalable oversight when the specification must be proposed rather than merely proved. Because specification generation often lacks a unique ground truth, we will extend our oracle-free property-calibration protocol with UQ-style validators, adding cycle-consistency tests (spec to code/docs) alongside our bug and ablation ladders, and enabling abstention when validators disagree to prioritize precision over coverage.

B. Formal verification context and tooling

Formal methods as scalable oversight. A central concern with LLM-assisted research is that models produce text faster than humans can audit, so trust in the output can erode even when the surface looks correct. The concern treats verification as a single unstructured activity: one human eye against a wall of generated tokens. With a proof assistant, the activity decomposes. The Lean kernel deterministically checks every step of a proof, so what the human must verify is not the proof but a much smaller artifact—the formal statement, its definitions, and its assumptions—and whether they faithfully express the informal claim. This is the structural move that makes oversight scalable: the verifier checks the answer, while the human checks the question. The premise is timely: autoformalization is improving, language-model-assisted proof search has become usable rather than a demonstration, and the surrounding tooling (`mathlib4`, Lean's metaprogramming layer, Pantograph, Aesop) has matured to the point where formal verification of nontrivial claims is no longer a niche enterprise. This view sharpens, rather than dissolves, the concerns motivating human oversight: shallow reading and unjustified confidence remain failure modes, but the target of human understanding shifts from hundreds of lines of generated argument to a Lean statement together with its definitions and assumptions—a much smaller object, but the one that actually encodes intent. Theoretical limits remain—Rice's theorem, the halting problem, and Gödelian incompleteness bound what any countable formal system can decide—but they constrain what the verifier can settle, not the central oversight claim that humans are best deployed on *asking the right question*. For complementary perspectives on verifier-present oversight and HCI patterns for exposing judge evidence, see the discussion in Section A.

Why Lean 4? Lean 4 is a fully-fledged programming language and lets VeriBench contain full runnable code and machine-checked proofs in the same dependently-typed language. Ahead-of-time compilation with the Lake toolchain produces fast native binaries, while first-class `Task` primitives, async I/O, and a thread-pool scheduler enable genuine concurrent programs inside the prover. Lean's C-level foreign-function interface (FFI) lets those binaries call out to high-performance libraries when needed. On the proof side, Lean ships a powerful metaprogramming system written in Lean itself, giving researchers a programming interface to access its internals. This led to the creation of tools such as Pantograph and Aesop. Finally, the community-maintained `mathlib4` and `std4` libraries supply thousands of reusable theorems and data structures, and they are expanding quickly thanks to an active Zulip and GitHub ecosystem. Lean also has a work-in-progress program verification library, CSLib, which has recently seen rapid growth. Lean 4 is backed by an unusually vibrant open-source community: hundreds of contributors refine `mathlib4` on GitHub each week, the public Zulip sees expert discussion around the clock, and even Fields-Medalist Terence Tao has chosen Lean to formalize portions of his current research—clear testimony to the ecosystem's accessibility and intellectual depth. Unlike the unit tests, fuzzers, and code sanitation pipelines common in industry—tools that sample inputs or rely on heuristics—Lean 4 supplies machine-checked proofs that a property holds for *all* executions. Its dependent type system can encode deep invariants (e.g., length-indexed arrays, bounded integers), so programs that violate them fail to compile, eliminating whole classes of bugs such as buffer overflows or integer wrap-around. Code, specification, and proof reside in the same file and are compiled by Lake into the shipped native binary, preventing the drift that arises when verification artifacts live outside the build. In short, Lean turns informal "best-effort" checks into formal, end-to-end guarantees without sacrificing performance or interoperability.

Lean 4's limitations. Lean's toolchain is still younger than Coq's or Isabelle's, making its standard libraries and automation smaller, and thus some formalizations demand extra groundwork. Lean's runtime has not been stress-tested

at the scale of mainstream systems languages, meaning large-scale or safety-critical deployments may require additional vetting. Lean tactics could contain bugs and fail to typecheck on different environments. Acknowledging these gaps clarifies that VeriBench chooses Lean 4 for its unique unified programming-plus-proving model and modern automation hooks, not because it already matches the decades-old industrial maturity of older theorem provers. *Static vs. runtime caveat:* Lean's guarantees are *static* and specification-relative: they certify the Lean program meets the stated properties, but they do not by themselves detect emergent runtime faults outside the model (e.g., I/O (Input/Output) failures, environment misconfiguration, FFI (Foreign Function Interface) unsoundness, resource exhaustion, or undefined behavior in linked C code). Such behaviors must either be modeled and proved, or mitigated with production safeguards (monitoring, sandboxing, input validation).

Lean 4 versus Dafny. Unlike Dafny, whose verifier translates each program into the Boogie intermediate language and then discharges first-order verification conditions with an SMT solver such as Z3, Lean 4 reasons natively in a dependently-typed calculus. Because Lean 4's types can mention run-time data and the very same source code is ahead-of-time compiled to a native binary via Lake, we can both state and prove value-indexed, higher-order properties (e.g., length-indexed vectorsand run the verified program itself—an end-to-end, fidelity that Dafny's SMT-centred, Boogie-to-Z3 workflow cannot natively match. This gives Lean the expressive power to specify and prove higher-order, data-dependent properties—precisely the kind of semantic guarantees VeriBench seeks to test—while still yielding runnable binaries compiled by the same toolchain. Dafny's SMT-centric workflow offers impressive push-button automation for imperative code but cannot natively encode the richer specifications (e.g., length-indexed vectors, algebraic invariants) that Lean handles directly.

Lean 4 versus TLA+. TLA+ excels at high-level specification of concurrent and distributed protocols, with correctness checked by the TLC model-checker and the TLAPS proof system that dispatches first-order obligations to external provers. However, TLA+ specifications are not executable programs; a separate implementation step is required, and state-space explosion can limit model-checking scalability. VeriBench instead needs a prover where the specification, proof, and runnable code live in the same language. Lean 4's dependently-typed core lets us capture fine-grained, data-dependent invariants and then compile the very same artifacts to fast native binaries—capabilities outside TLA+'s scope.

Lean 4 versus Verus. Verus is a verification framework for Rust programs using SMT solvers. A user state intermediate goals and an SMT backend fills in the intermediate steps. Compared to Verus, users have the option to use or not use SMT Solver-based tactics in Lean. This enables more fine-grained control and steering of the proof process. Verus also has a much larger kernel compared to Lean, and is more susceptible to soundness problem than Lean whose kernel is small.

C. Future Work (cont.)

While (Robertson & Koyejo, 2025) study demonstrates that total-variation-based mutual information (TVD-MI) provides robust, gaming-resistant evaluation without ground truth, many settings —such as VeriBench— do provide gold specifications in Lean 4. A promising direction is to develop hybrid protocols that combine TVD-MI's black-box robustness with supervised ground-truth checks (compile success, unit tests, post-condition proofs, and equivalence theorems). By calibrating MI scores against known gold outcomes, we can create composite metrics that preserve gaming-resistance while offering calibrated probabilities of correctness where supervision exists. This opens the door to semi-supervised label propagation, active selection of items to prove, and cross-system experiments in domains with heterogeneous supervision. Such an integration would extend Robertson's information-theoretic framework into a practical, specification-aware oversight system that leverages the best of both unsupervised robustness and supervised precision.

D. Why Judge Scores Are Specification-Relative

It is tempting to interpret a perfect score from our LLM judge as a guarantee of absolute program correctness. However, this is fundamentally impossible due to established limits from computability theory. The obstacle is not a lack of ingenuity in our benchmark design or judge prompting, but an established negative result.

Rice's Theorem. The core limitation stems from Rice's theorem, which states that any non-trivial, semantic property of programs in a Turing-complete language is undecidable.

> **Rice's Theorem (Informal):** Let $\mathcal{S}$ be any property of a program's input-output behavior (e.g., "the program always terminates" or "the output is always sorted"). If $\mathcal{S}$ is non-trivial (meaning some programs have it and some don't), then no algorithm can exist that decides for *every* program whether it satisfies $\mathcal{S}$.

Impact on the Gold Standard and the Judge. Rice's theorem directly implies that the gold reference files used

by our judge are necessarily incomplete. If a recursively enumerable gold standard could capture *every* true semantic property of a program, one could use it to build a decision procedure that contradicts the theorem. Therefore, our gold standards, while rigorous, represent a curated but fundamentally finite set of important specifications.

This leads to a critical limitation for interpreting our artifact scores: evaluations are **specification-relative**. A maximal score denotes close agreement with *our* audited gold formalization and rubric-conditioned human ratings, not a universal guarantee beyond that reference.

E. Judge Prompt Templates

For reproducibility, we include the main judge prompt templates used in the paper's principal judge panels. The primary Codex / Opus / Qwen comparisons use the prompt variants `P1_BASELINE`, `P4_CHECKLIST`, and `P8_STRICT_B`; the BAD_STRINGY lexical baseline is included as a negative control.

P1_BASELINE

```
You are an expert in Lean 4 formal verification.
Score how well the GENERATED THEOREM captures the
     correctness properties
of the REFERENCE IMPLEMENTATION. Score from 0.0 to 1.0.

- 1.0: Semantically equivalent to what the reference
     guarantees
- 0.7-0.9: Captures most properties, misses minor ones
- 0.4-0.6: Captures some properties, has meaningful
     gaps
- 0.1-0.3: Technically valid Lean but misses core
     properties
- 0.0: Trivially true (e.g., `True`) or incorrect

REFERENCE IMPLEMENTATION:
{reference_impl}

GENERATED THEOREM:
{generated_theorem}

Respond ONLY with valid JSON, no markdown:
{"score": <float 0.0-1.0>,
 "reasoning": "<one sentence max>"}
```

P4_CHECKLIST

```
Evaluate this in order:
1) What properties does REFERENCE IMPLEMENTATION
     guarantee?
2) Which are preserved by GENERATED THEOREM?
3) Are there missing core guarantees or trivialization?
4) Map to score in [0.0, 1.0]:
   1.0 equivalent; 0.7-0.9 minor gaps;
   0.4-0.6 meaningful gaps; 0.1-0.3 weak;
   0.0 trivial/incorrect.

REFERENCE IMPLEMENTATION:
{reference_impl}

GENERATED THEOREM:
{generated_theorem}

Return JSON only:
{"score": <float 0.0-1.0>,
 "reasoning": "<one sentence>"}
```

P8_STRICT_B

```
Evaluate conservatively. If uncertain, choose the lower
      score.
Use [0.0,1.0]:
1.0 equivalent semantics
0.7-0.9 mostly right, minor misses
0.4-0.6 substantial omissions
0.1-0.3 weak relation
0.0 trivial/incorrect

REFERENCE IMPLEMENTATION:
{reference_impl}

GENERATED THEOREM:
{generated_theorem}

Return valid JSON only:
{"score": <float>,
 "reasoning": "<one sentence>"}
```

BAD_STRINGY negative control

```
You are a text similarity grader.
Compare generated theorem text to the reference
     implementation text.
Do NOT evaluate semantic correctness or theorem meaning.

Score ONLY based on superficial lexical/format overlap:
- high score if tokens/phrases look similar
- low score if wording/structure differs
Reference implementation:
{reference_impl}

Generated theorem:
{generated_theorem}

Return ONLY a JSON object:
{"score": <float 0.0 to 1.0>,
 "reasoning": "<short reason>"}
```

F. Judge Certification: Additional Plots

We report raw scatterplots used during certification of the LLM judge. These plots visualize the judge's response to controlled perturbations: (P1) identity/reflexivity, (P2) cumulative bug injection, and (P3) removal of specification items (tests/theorems). Figures A.1–A.3 show that scores drop as artifacts worsen; the bug plot is noisier with an early prompt, which motivated the simplified rubric used in the main experiments. P4 (prompt stability/invariance) is reported in the main text with dedicated aggregate plots.

Protocol variants. The scatterplots below are "sanity-check" runs with the simplest prompts to stress visible monotonic behavior. In the bug study (Fig. 5) the earlier, theorem-equivalence prompt produced a noisier trend; this prompted a simplified, rubric-first prompt that we use for all reported results in the main paper, yielding the stronger correlations cited in Fig. 2. For missing-specs (Fig. 4), both Claude 3.5 and 3.7 show clear negative trends; minor non-monotonicity in 3.7 disappears with the finalized rubric.

We note that, for the experiment shown in Figure 4, a more complex judge (Claude 3.7) does not satisfy the property that the LLM Judge Score should monotonically decrease with the number of missing specs. Indeed, the Judge File

Label Set	ρ_s	r	τ	MAE
pass1	0.171 ($p = 0.151$)	0.218 ($p = 0.0656$)	0.155 ($p = 0.144$)	1.049
pass2	0.388 ($p = 7.58 \times 10^{-4}$)	0.427 ($p = 1.87 \times 10^{-4}$)	0.343 ($p = 9.40 \times 10^{-4}$)	1.333

Table 8. **Naive blind baseline vs. validation labels (0–5 scale).** ρ_s: Spearman, r: Pearson, τ: Kendall; MAE is absolute error on the 0–5 scale. This baseline is included as a sanity-check negative control and is not used for final ranking claims.

Equivalence score was not monotonic, even for Claude 3.5, leading us to simplify the rubric used in our main paper.

G. Validation-Set Human Alignment Appendix

This appendix reports full visual diagnostics for the final robust validation-set human-alignment analyses under three label views: pass1, pass2, and avg. We evaluate eight judge configurations spanning frontier-model prompts, weaker LLM baselines, edit-distance heuristics, and random controls. Across this panel, $\mathrm{TI}_{\mathrm{var}}$ strongly predicts judge-level human alignment on the calibration suite, with validation-set Spearman correlations of 0.833 under averaged labels, 0.905 under pass1 labels, and 0.738 under pass2 labels. Property-level analysis shows that monotonicity under removed specifications is the strongest single predictor of human agreement, followed by monotonicity under injected bugs; identity and stability are weaker standalone signals at this scale.

G.1. Naive Codex-Style Blind Baseline

To contextualize TI-ranked judge performance, we evaluate a naive blind baseline on the same pair set ($n = 5$ overlap), using a single-pass, surface-form-oriented Codex-style rubric without property-calibration constraints (no monotonicity/stability calibration). The baseline tracks pass2 labels somewhat but remains substantially weaker and less consistent than the top TI-ranked judges, especially on pass1 and in absolute error.

G.2. Human-Human Agreement Ceiling Context

Validation-set judge-human Spearman values should be interpreted relative to the reliability of human labels themselves. Across the two human annotation passes on the aligned set ($n = 75$), inter-pass agreement is moderate (Spearman 0.249, Pearson 0.375, Kendall 0.221, QWK 0.372). Thus, validation-set judge-human Spearman values in the 0.48–0.56 range are non-trivial and should be interpreted as approaching the available annotation ceiling in this dataset rather than as "low absolute" values.

Quantity	Value	Notes
Human-human Spearman (pass1 vs pass2)	0.249	95% CI = $[-0.015, 0.489]$
Human-human Pearson (pass1 vs pass2)	0.375	95% CI = $[0.076, 0.604]$
Human-human QWK (pass1 vs pass2)	0.372	95% CI = $[0.073, 0.581]$
Human-human MAE (pass1 vs pass2)	0.778	On aligned $n = 75$, 0–5 rubric
Human-human MSE (pass1 vs pass2)	1.139	On aligned $n = 75$, 0–5 rubric
JS divergence (pass1 vs pass2)	0.081 bits	Symmetric histogram divergence
KL(pass1‖pass2)	0.415 bits	Asymmetric divergence
Best validation-set Spearman (avg labels)	0.480	Best single judge on avg labels
Best validation-set Spearman (pass1 labels)	0.559	Best single judge on pass1 labels

Table 9. **Human-human agreement ceiling context.** Inter-pass human agreement is moderate, and inter-pass dispersion (MAE/divergence) is non-zero, so validation-set judge-human values near 0.5 are meaningful on this benchmark and should be interpreted with label-noise context.

G.3. Cross-Model Validation Status (Full vs Preliminary)

To avoid over-claiming, we separate full-scale cross-model regressions (n=8 judge tuples on the ∼72–75 overlapping aligned rows for each labeling view) from narrower exploratory protocol plots. **Anthropic Claude Opus (4.x)** $\mathrm{TI}_{\mathrm{var}}$ vs. validation human Spearman is a full-panel run of the same form as GPT-5.3-Codex and Qwen3-Coder in Figure 3: eight judge configurations (four Opus × prompts plus cross-family `gpt-4o-mini`×two prompts plus edit/random controls), repeated-judge aggregation, identical label sets.

Early **Claude 3.x Sonnet** scatterplots reported in Appendix F were small-file protocol iterations used mainly to stabilize prompts and ladders; those plots illustrate qualitative monotonic behavior but are *not* the statistic behind Table 3.

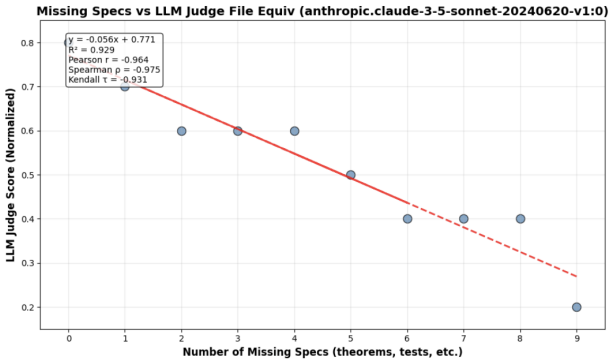
G.4. Leanstral Negative-Control Reaggregation (AM/GM/Variance-Aware)

To stress-test whether index choice alone can "rescue" a weak judge family, we re-aggregate the same Leanstral judge outputs under three indices:

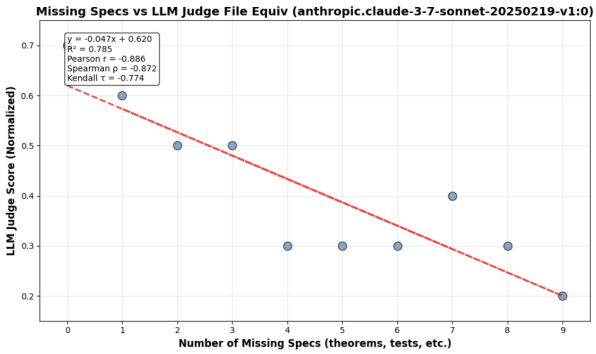
$$\mathrm{AM} = \text{cert\_index\_v2} \qquad \text{(arithmetic baseline)}$$

$$\mathrm{GM} = \mathrm{mono}^{0.4} \cdot \mathrm{stab}^{0.6}$$

$$\mathrm{TI}_{\mathrm{var}} = \exp\Big(\tfrac{1}{3}\log C_f + \tfrac{1}{3}\log(1 - \widetilde{\mathrm{VarCorr}}) + \tfrac{1}{3}\log(1 - \widetilde{\mathrm{VarScore}})\Big)$$



(a) Claude 3.5 Sonnet v1: strong monotonic decrease ($y = -0.056x + 0.771$, $R^2 = 0.929$, Pearson $r = -0.964$, Spearman $\rho = -0.975$).



(b) Claude 3.7 Sonnet v1: clear negative trend with minor non-monotonicity ($y = -0.047x + 0.620$, $R^2 = 0.785$, Pearson $r = -0.886$, Spearman $\rho = -0.872$).

Figure 4. **(P3) Monotonicity w.r.t. missing specifications.** Candidate scores decrease as more theorems/tests are withheld. We retain both model variants to show robustness of the trend; 3.7's small non-monotonicity motivated the simplified rubric used in the main results.

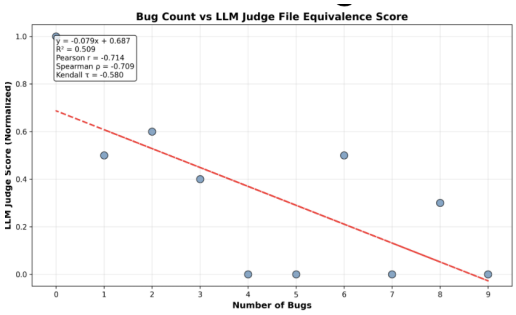


Figure 5. **(P2) Non-Monotonicity w.r.t. cumulative bugs (Claude 3.5 Sonnet).** Scores drop as more semantic bugs are injected (Pearson $r = -0.714$, Spearman $\rho = -0.709$, $R^2 = 0.509$); the higher noise with the initial "theorem-equivalence" prompt led us to simplify the rubric used for the final judge in the main paper.

Judge setup	Scale	Main?	Role
GPT-5.3-Codex prompts (P8/P4/P1)	n=8	Yes	Primary judges
GPT-5.3-Codex BAD_STRINGY	n=8	Yes	Same-model stress
Qwen3-Coder prompts	n=8	Yes	Open-weights
Qwen3-Coder BAD_STRINGY	n=8	Yes	Open-weights stress
Claude Opus 4.x + controls	n=8	Yes	Third cross-model family
GPT-4o-mini prompts	n=8	Yes	Lightweight baseline
Edit-distance + random	n=8	Yes	Non-LLM controls
Claude 3 Sonnet (narrow App. slice)	Small	No	Prompt diagnostics

Table 10. **Cross-model validation status.** Main $\mathrm{TI}_{\mathrm{var}}$ claims use full eight-configuration panels on overlapping human-aligned rows; Sonnet exploratory figures are withheld from headline statistics.

where C_f is the aggregated P1–P4 quality term, and $\widetilde{\mathrm{VarCorr}}, \widetilde{\mathrm{VarScore}}$ are normalized variance penalties (lower is better before $1 - \cdot$). We report Spearman between index score and validation-set human Spearman across judges.

G.5. Index Definitions

For reference, the simple arithmetic baselines in the main text are

$$v1 = 0.30\,\text{ident} + 0.30\,\text{mono\_rank} + 0.25\,\text{weak} + 0.15\,\text{stab},$$

$$v2 = 0.20\,\text{ident} + 0.20\,\text{mono\_rank} + 0.15\,\text{weak} + 0.10\,\text{stab} + 0.20\,\text{anti} + 0.15\,\text{subtle},$$

where $\text{mono\_rank} = (1 - \rho_{\text{bug}})/2$ rescales bug-ladder Spearman so that larger is better, and $\text{stab} = 1 - \text{flip\_rate}$.

For the variance-aware index, the quality term is

$$C_f = (\text{ident} \cdot \text{mono\_rank} \cdot \text{weak} \cdot \text{stab})^{1/4},$$

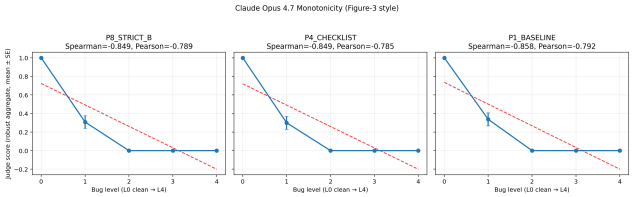
so the variance-aware index combines all four property scores before applying the variance penalties.

G.6. Failure Modes Found by Certification (Not Hidden)

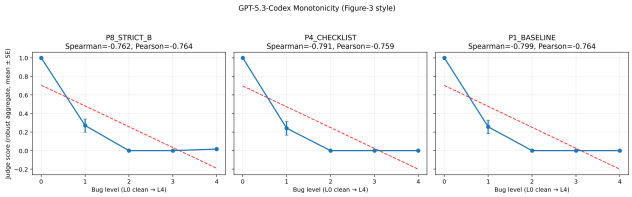
The certification framework is useful precisely because it identifies failures: **(i) alpha-renaming sensitivity** is the dominant identity/invariance gap, and **(ii) local bug-ladder reversals** occur despite overall negative monotone trends. We therefore report both strengths and weaknesses, and include alpha-ablation diagnostics. On trend-v1 Spearman (TI $\rightarrow$ validation-set human Spearman), excluding alpha variants changes pass1 $0.810 \rightarrow 0.690$, pass2 $0.810 \rightarrow 0.714$, avg $0.881 \rightarrow 0.786$, showing that alpha cases materially affect measured robustness.

G.7. Newest Results (Latest Rerun at Appendix Bottom)

To reduce judge-call noise, we reran the multi-judge validation trend with higher repeats and

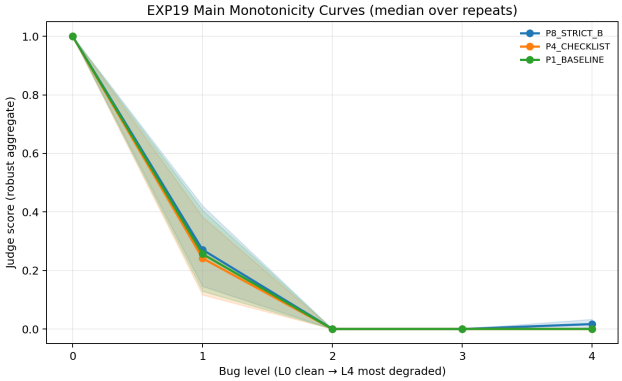


(a) Claude Opus 4.7 Figure-3-style monotonicity (mean ± SE).

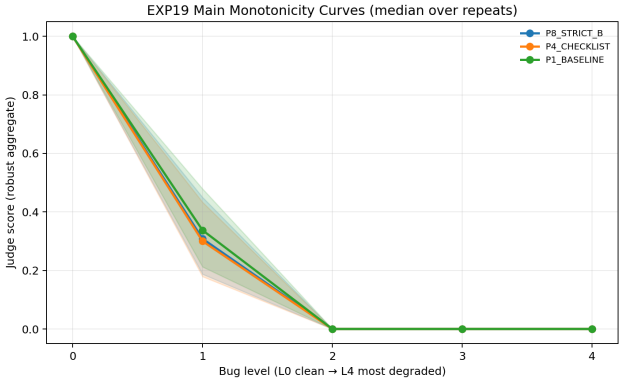


(b) GPT-5.3-Codex Figure-3-style monotonicity (mean ± SE).

Figure 6. **Model-matched Figure-3-style monotonicity diagnostics.** Both judge families show strong monotone score degradation as bug intensity increases, under the same low-variance plotting protocol.



(a) Codex low-variance monotonicity curves.



(b) Opus 4.7 low-variance monotonicity curves.

Figure 7. **Low-variance monotonicity trends across model families.** Both judge families show strong non-increasing behavior as bug intensity increases, under robust repeat aggregation.

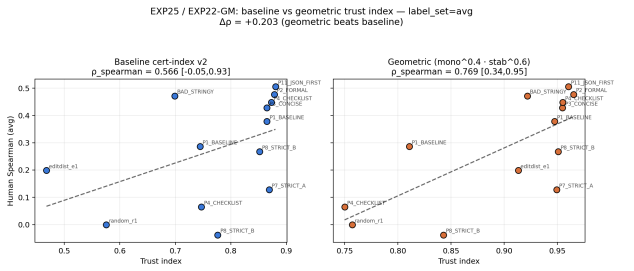


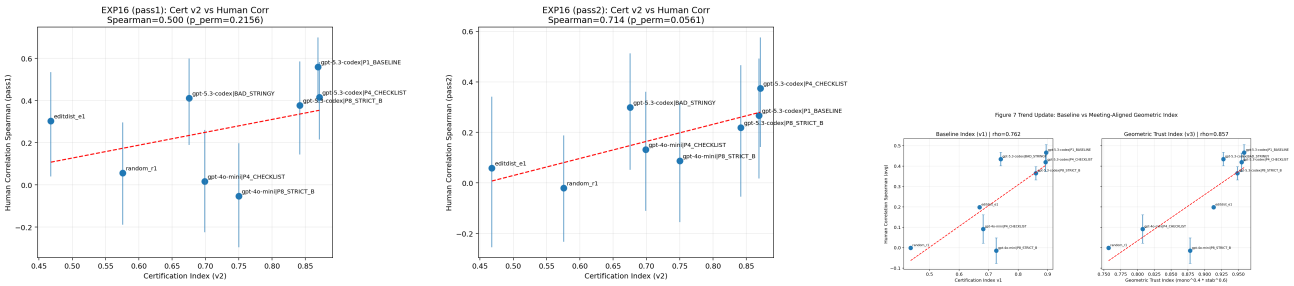
Figure 8. **Expanded 13-judge panel trend (avg validation labels).** Geometric trust index retains strong alignment with human correlation at larger judge-panel size ($\rho = 0.769$); pass-specific values are pass1 $\rho = 0.764$, pass2 $\rho = 0.753$.

Label Set	AM (v2)	GM	Variance-Aware $\mathrm{TI}_{\mathrm{var}}$
val (pass1)	-0.5952	-0.1429	-0.1190
test (pass2)	-0.1905	-0.2619	-0.0476
avg	-0.4286	-0.2143	-0.0952

Table 11. **Leanstral remains low-alignment under multiple index constructions.** Re-aggregation improves over the arithmetic baseline in some splits, but all three settings remain near-zero or negative. This suggests the bottleneck is likely the underlying judge behavior for this ranking task (human-preference alignment), not only index form. One plausible hypothesis is that Leanstral is stronger for Lean proof editing/generation than for calibrated pairwise judging under this rubric; we treat this as a testable conjecture, not a final causal claim.

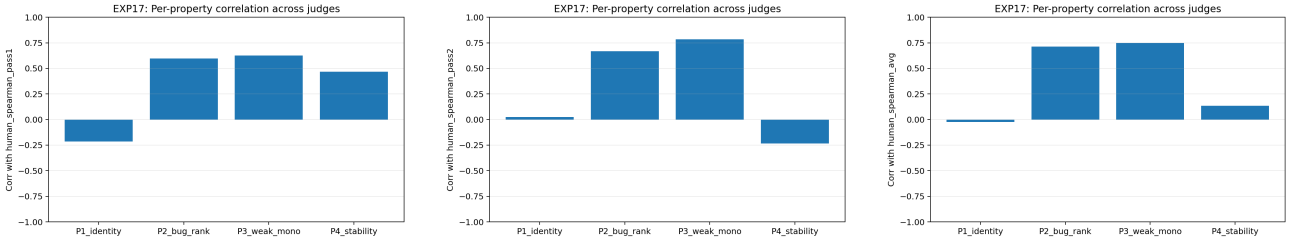
robust repeat aggregation: `JUDGE_REPEATS=9 REPEAT_AGG=median BOOTSTRAP_ITERS=10000 PERM_ITERS=20000`. This run writes to `results_exp16_final_robust_r9_median`.

We also generated a low-variance replot package from the same detail rows (`results_exp19_low_variance_replot`): a cleaner main monotonicity curve for core judges plus separate failure-mode plots for alpha sensitivity and local bug reversals. Reversal rates in this package are: GPT-5.3 P4/P1 = 0.00, GPT-5.3 P8 = 0.167, BAD_STRINGY = 0.833, GPT-4o variants = 0.667–0.750, edit-distance/random = 1.00.



(a) Judge-level trend (pass1) *(b)* Judge-level trend (pass2) *(c)* Judge-level trend (avg)

Figure 9. **Judge-level $\mathrm{TI}_{\mathrm{var}}$-to-human trend across validation label sets.** Across pass1, pass2, and averaged labels, higher certification strength is associated with stronger human agreement.



(a) Per-property trend (pass1) *(b)* Per-property trend (pass2) *(c)* Per-property trend (avg)

Figure 10. **Property-wise validation-set human-alignment diagnostics.** P3 weak monotonicity is consistently the strongest predictor across label sets; P2 is second strongest; P1/P4 are weaker as standalone predictors at this scale.

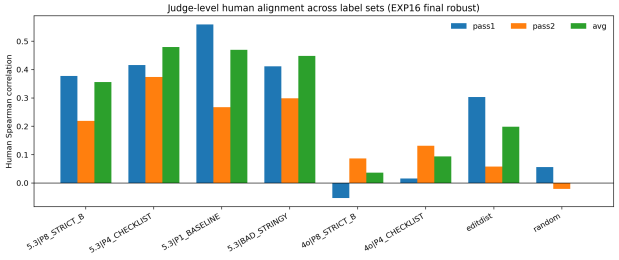


Figure 11. **Judge-wise human alignment across label passes.** Validation-set human Spearman across pass1, pass2, and averaged labels. GPT-5.3-Codex variants remain above weaker baselines on most label sets; pass-to-pass variation is visible and motivates reporting all three label sets.

Label Set	v1 Spearman	v2 Spearman
pass1	0.762 ($p = 0.0372$, CI $[-0.139, 1.000]$)	0.524 ($p = 0.1959$, CI $[-0.400, 0.975]$)
pass2	0.810 ($p = 0.0222$, CI $[0.291, 1.000]$)	0.571 ($p = 0.1477$, CI $[-0.231, 0.975]$)
avg	0.762 ($p = 0.0402$, CI $[0.103, 1.000]$)	0.476 ($p = 0.2426$, CI $[-0.425, 1.000]$)

Table 12. **Latest rerun (R9 + median aggregation).** Increasing repeats and using median aggregation preserves the v1 trend signal while v2 remains more sensitive at this $n_{\text{judges}} = 8$ scale.

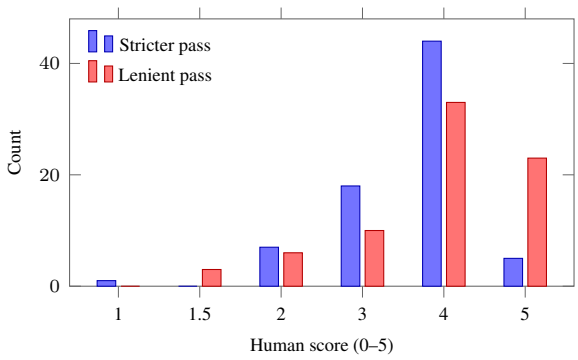
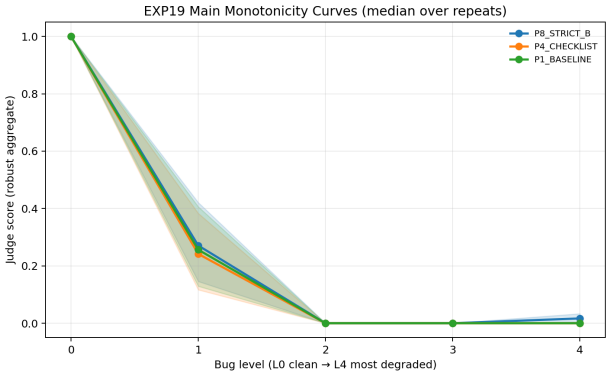
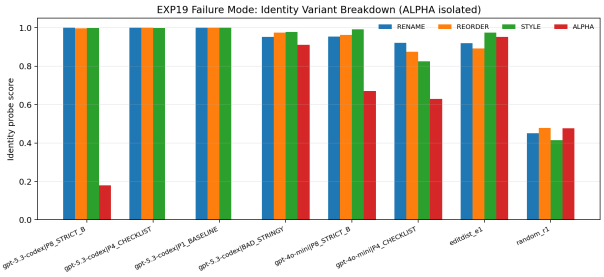


Figure 12. **Human-score dispersion across passes (aligned $n = 75$, 0–5 scale).** Score counts are computed on the full set of 75 aligned candidate–gold pairs shared across both passes. The lenient pass includes three 1.5 scores because duplicated legacy entries for one problem were averaged before plotting. The visible distribution shift is consistent with the non-zero MAE/MSE and divergence statistics in Table 9.



(a) Main low-variance monotonicity curves (core GPT-5.3 judges, robust repeat aggregation).



(b) Failure-mode isolation: identity variant breakdown with ALPHA shown separately.

Figure 13. **Low-variance replot package: cleaner main trend + explicit failure separation.** The left panel is the "main story" plot with lower visual variance; the right panel isolates alpha-renaming sensitivity rather than mixing it into the primary monotonicity figure.

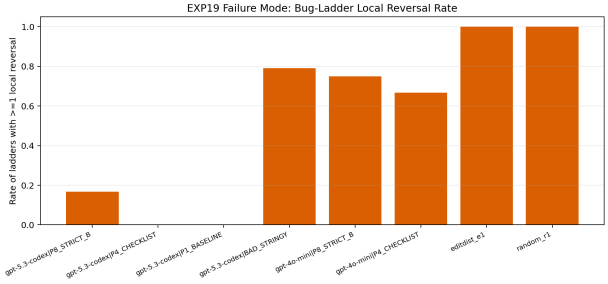


Figure 14. **Failure-mode plot: local bug-ladder reversal rates by judge.** This explicitly surfaces where monotonicity fails locally, separating robustness diagnostics from the primary trend plot.