
SpecCheck: When LLMs Formalize, Who Checks the Spec?¹

Bhagyesh Kumar
MIT Manipal

With
Apart Research

Abstract

Proof checkers guarantee that code satisfies a specification, but not that the specification captures what the user intended. When large language models formalize natural language requirements, they can silently weaken postconditions, drop edge cases, or resolve ambiguous terms in ways the user did not expect: a failure mode we call *intent drift*. We present SpecCheck, a pipeline that detects intent drift through two complementary techniques: *ambiguity elicitation*, which surfaces implicit assumptions before formalization, and *round-robin cross-validation*, where two independent models each formalize the same requirement and informalize the other’s output. Across Lean 4 theorems (VERINA) and Python Hypothesis property suites, we find that the value of cross-model validation is formalism-dependent: on Lean 4, it exposes under-formalization that single-model assessment systematically misses (20-percentage-point gap), while on property-based tests, single-model assessment is already well-calibrated. The full pipeline achieves a 97% confirmation rate on correct specifications and 100% detection on deliberately weakened ones. Ambiguity count at elicitation time predicts which requirements will cause trouble downstream, flagging difficult requirements before formalization.

1 Introduction

A proof checker can guarantee that code satisfies a specification. It cannot guarantee that the specification captures what the user intended. This distinction has always mattered, but it becomes acute when large language models (LLMs) are used to translate natural language (NL) requirements into formal specifications: property-based tests, Lean 4 theorems, or Dafny pre- and postconditions. LLMs can produce plausible formalizations in seconds, but their output requires human judgment to verify faithfulness. Manual validation does not scale.

The failure mode is intent drift: a generated specification can be formally valid, provably satisfiable, logically consistent and yet miss edge cases the user considered obvious, permit behaviors the user intended to forbid, or resolve ambiguous terms in ways the user did not expect. Proof checkers cannot detect these gaps. The proof is correct relative to the specification, but the specification may fail to capture the intended semantics.

Graciolli and Amin [3] introduced claimcheck, a round-trip detection approach: a model informalizes a

¹Research conducted at [the Secure Program Synthesis Hackathon](#), Apart Research, May 2026.

formal specification back to natural language, and a judge compares the result to the original requirement. This detects many forms of drift. But single-model round-tripping has a structural blind spot. If a model assumes that “sorted” means non-decreasing and carries that assumption through both formalization and informalization, the round-trip appears clean. The specification is wrong, but no signal surfaces.

SpecCheck addresses this through two complementary mechanisms. First, ambiguity elicitation enumerates implicit assumptions and underspecified terms before any formalization occurs. A requirement like “merge two sorted arrays” leaves unspecified whether the output must itself be sorted, or merely contain all elements. By surfacing this ambiguity upstream, the user can clarify intent before it is silently resolved incorrectly by a model. Second, round-robin cross-validation uses two independently operating models: each formalizes the same NL requirement separately, then informalizes the other’s output. Disagreement between informalizations provides a drift signal inaccessible to single-model pipelines. When one model writes a postcondition checking only set membership and the other additionally requires sorted order, the weaker specification is exposed not because it fails a proof, but because a second model reads it and notices what is missing.

Our main contributions are:

1. **Ambiguity elicitation as a pre-formalization stage:** a structured pass that enumerates implicit assumptions before formalization, surfacing drift that round-tripping misses when both models share the same prior.
2. **Round-robin cross-validation:** two independent LLMs formalize the same requirement and each informalizes the other’s output, providing a cross-model drift signal unavailable to single-model pipelines.
3. **Empirical evidence for formalism-aware validation:** cross-model disagreement provides substantial additional signal for Lean 4 specifications, where single-model assessment misses under-formalization; for property-based tests, single-model assessment is already well-calibrated and cross-validation adds no systematic benefit.

We describe the SpecCheck pipeline and experimental conditions (§3), present results across both domains (§4), and discuss implications for formalism-aware validation together with limitations of the current evaluation (§5).

2 Related Work

Claimcheck. Graciolli and Amin [3] introduced blind round-trip informalization for detecting intent drift in Dafny formalization, and independently recovered corrections that later appeared in the FOLIO benchmark’s second version. SpecCheck extends this to the generative setting: rather than validating existing specifications, it validates the formalization process itself, adding cross-model disagreement as a detection signal that single-model round-tripping structurally cannot provide.

NL-to-spec translation. Cosler et al. [1] show that structured ambiguity elicitation improves NL-to-temporal-logic translation; Endres et al. [2] document systematic postcondition weakening by LLMs. SpecCheck builds on both: it elicits ambiguities upstream and detects the weakened-postcondition patterns Endres et al. describe.

Verifier-backed and closed-loop pipelines. Clover [6] treats specifications as trusted ground truth for code generation; SpecGen [4] prunes candidate postconditions using verifier counterexamples. Both operate *after* formalization, checking provability or correctness against code. SpecCheck instead asks whether the specification matches the NL requirement. Cross-model critique [5] provides the mechanism: critique by a different model reduces self-confirmation bias inherent in single-model pipelines.

3 Methods

3.1 Pipeline Overview

SpecCheck processes each NL requirement through five sequential stages, using `claude-haiku-4-5` as model A and `claude-sonnet-4-5` as model B and judge.²

Stage 0—Ambiguity Elicitation. Before any formalization, model A enumerates implicit assumptions in the NL requirement: undefined or context-dependent terms, unstated boundary cases, type or range assumptions, and ordering assumptions. The output is a structured list of 1–5 ambiguities. In interactive use, these are presented to the user for clarification before proceeding; in our automated evaluation, they are recorded but the original NL is left unchanged, allowing us to measure whether ambiguity count predicts downstream disputes.

Stage 1—Independent Formalization. Models A and B each independently formalize the same NL requirement without seeing each other’s output. For Domain 1 (VERINA), the target is a Lean 4 precondition–postcondition pair with a theorem statement. For Domain 2 (Hypothesis), the target is a Python property-based test suite using `@given` decorators with appropriate strategies.

Stage 2—Cross-Informalization. Model A informalizes B’s specification, and model B informalizes A’s specification. Neither model sees the original NL requirement during this stage—only the formal specification written by the other model. Each informalization must describe what the specification literally states, note which edge cases are and are not covered, and rate the specification’s strength (trivial, weak, moderate, or strong).

Stage 3—Round-Robin Judgment. The judge (model B) receives the original NL requirement alongside both back-translations. It scores each on five dimensions: completeness, precision, scope, edge-case coverage, and implicit-assumption handling, assigns a drift score in $[0, 1]$, and labels any drift found with a type from `{tautology, weakened-postcondition, narrowed-scope, missing-case, wrong-property}`. These labels are enforced by the tool’s JSON schema: the judge selects from a fixed closed set and cannot produce labels outside it, bounding free-form inconsistency. It also flags whether the two independently generated specifications disagreed on what to formalize.

Stage 4—Aggregation. A requirement is **confirmed** only if both back-translations are judged to match the original NL requirement; otherwise it is **disputed**. The pipeline also records which signal contributed to each dispute: round-robin disagreement, high ambiguity count (≥ 2 flagged assumptions), both, or neither.

Because all benchmark specifications are human-verified, confirmation rate measures the pipeline’s tendency to avoid falsely disputing correct specifications rather than specification correctness itself. We evaluate four conditions that selectively enable these stages (§3.2), across two benchmarks (§3.3).

3.2 Experimental Conditions

We evaluate four conditions that selectively enable SpecCheck’s stages:

- **B1 (single-model baseline):** Model A formalizes; model B informalizes blind; model B judges. No ambiguity elicitation, no dual formalization, no cross-informalization. Note that model B plays both informalizer and judge roles here, which differs from `claimcheck`’s original setup where those roles are separated; this conflation may inflate confirmation rates relative to a fully separated baseline.
- **B2 (round-robin only):** Dual formalization and cross-informalization active; no ambiguity elicitation.
- **B3 (ambiguity only):** Ambiguity elicitation active; model A formalizes; model B informalizes blind

²Full model IDs: `claude-haiku-4-5-20251001` and `claude-sonnet-4-5-20250929`. All stages use temperature 0 with structured output enforced via forced `tool_choice`. Prompts are fixed across all runs; no few-shot examples are used.

and judges. No dual formalization or cross-informalization.

- **B4 (full SpecCheck):** All stages active: ambiguity elicitation, dual formalization, cross-informalization, and round-robin judgment.

3.3 Benchmarks

We evaluate on two domains that represent distinct points on the formalism spectrum: Lean 4, a dependently-typed proof assistant where specifications must capture deep algorithmic structure, and Python Hypothesis, a property-based testing framework where specifications are typically short, behavioral, and grounded in concrete input–output checks.

Domain 1—VERINA (50 tasks). We sample 30 basic and 20 advanced tasks from VERINA [7] (seeds 42 and 43), covering array operations, string processing, mathematical predicates, sorting, search, and algorithm problems. All tasks have human-verified correct specifications.

Domain 2—Hypothesis (50 tasks). We combine 21 tasks derived from Python standard library docstrings (`sorted`, `str.split`, `set` and `dict` operations, `Counter`, `map`, `filter`, `enumerate`, `math` builtins) with 29 handcrafted specifications covering deduplication, binary search, `flatten`, `running maximum`, `group-by`, `rotation`, `chunking`, `merge-sorted`, `sliding window`, `transpose`, `interleave`, `partition`, `cumulative sum`, `palindrome check`, and `compact`. Gold suites and injected bugs (three per task) are included in the repository for future recall evaluation; this paper reports calibration only.³

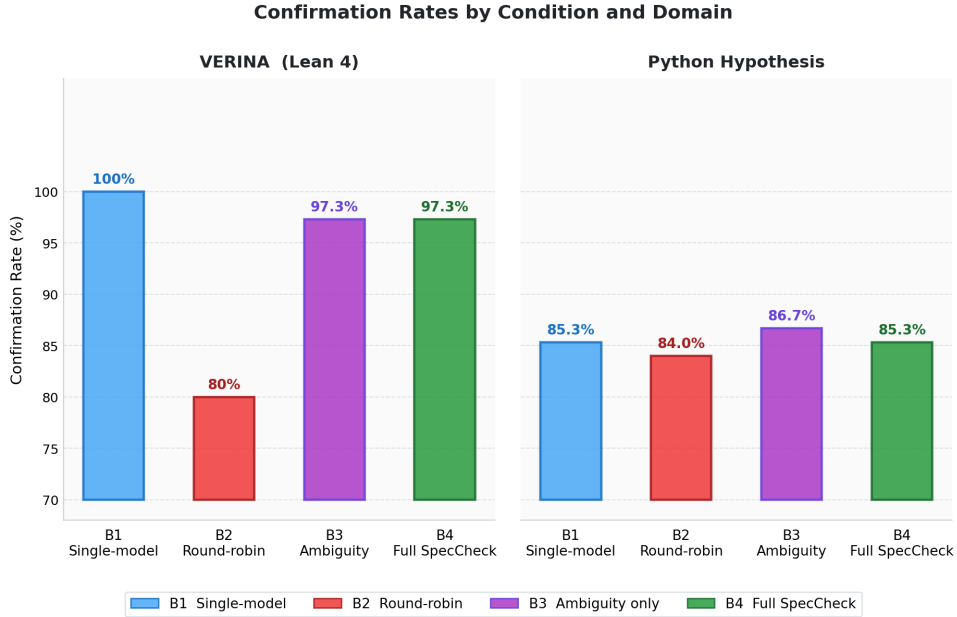


Figure 1. Confirmation rates by condition and domain across three runs. *Left:* VERINA Lean 4. *Right:* Python Hypothesis. The round-robin-only condition (B2) drops to 80% on VERINA, revealing structural weaknesses that the single-model baseline (B1, 100%) does not detect. All four Hypothesis conditions fall within 3 percentage points of each other, indicating that we do not observe a consistent benefit from cross-model validation for property-based tests. Conditions: B1=single-model, B2=round-robin only, B3=ambiguity only, B4=full SpecCheck.

³Ablation conditions (B1–B3) are reported on the first 25 tasks per domain due to time constraints; the full pipeline (B4) is evaluated on all 50.

4 Results

4.1 Main Results

Metric note. Because all benchmark tasks have human-verified correct specifications, every dispute is a false positive. **Confirmation rate** = $1 - \text{false-positive dispute rate}$: higher is better on these benchmarks. A pipeline that confirms everything scores 100% but has zero recall; a pipeline that disputes everything scores 0% but has no precision. The sensitivity evaluation (§4.3) provides the recall estimate on known-bad specs.

Figure 1 shows confirmation rates across three independent runs per condition. Differences in the Hypothesis condition fall within run-to-run variance and are not statistically meaningful.

4.2 Drift Type Breakdown

Table 1 reports average disputes per run by drift type. On VERINA, B2 surfaces wrong-property drift (3.0 per run) and weakened postconditions (1.3 per run) that B4 reduces through ambiguity elicitation. On Hypothesis, tautology disputes are the dominant signal, with an anomalous spike in B4 Run 2 noted below.

Table 1. Average disputes per run by drift type (averaged across 3 runs; each cell is mean disputes per single run, not cumulative across runs)

Drift type	VERINA (Lean 4)			Hypothesis (Python)		
	B2	B3	B4	B2	B3	B4
	(round-robin)	(ambiguity)	(full)	(round-robin)	(ambiguity)	(full)
Wrong property	3.0	0.0	0.0	0.0	0.0	0.0
Weakened postcondition	1.3	0.7	0.7	3.0	1.0	0.3
Tautology	0.0	0.0	0.0	0.7	1.7	2.7*
Narrowed scope	0.7	0.0	0.0	0.3	0.3	0.3

*B4 Hyp tautology rate inflated by Run 2 (7 disputes); Runs 1 and 3 average 0.5. B1 omitted: 0 disputes on VERINA, ≤ 4 on Hypothesis.

4.3 Ambiguity Count as a Predictor of Disputes

Table 2 shows that requirements flagged with more assumptions at Stage 0 dispute at substantially higher rates in subsequent stages. On the Hypothesis B3 condition (ambiguity elicitation active, $n = 25$), the point-biserial correlation between ambiguity count and dispute outcome is $r = 0.38$: dispute rate rises from 0% at ambiguity count 1, to 19% at count 2, to 50% at count 3. The `str_split_whitespace` task (average 3.0 flagged assumptions per run) is the only requirement disputed across all three runs in every condition. Ambiguity count is unavailable for B2 (Stage 0 skipped), so the correlation is reported for B3 only. A per-domain breakdown is not reported for VERINA B3 because only 2 tasks are disputed across all three runs, which is insufficient for a meaningful correlation estimate.

Table 2. Dispute rate by Stage 0 ambiguity count (B4, pooled across runs)

Ambiguity count	N requirements	Dispute rate
0–1	22	2.3%
2	27	7.1%
≥ 3	1	44.4%

4.4 Sensitivity: Detection of Known-Bad Specifications

To assess whether the low false-positive rate comes at the cost of sensitivity, we constructed 10 manually weakened Hypothesis specifications—each a variant of a gold spec with a single deliberate drift injected—and ran SpecCheck (B4 configuration) on them. Drift types covered: weakened-postcondition (6), missing-case (3), and tautology (1). SpecCheck disputed all 10 (100%), correctly identifying the drift type in each case. Representative detections: the strict-inequality variant of `sorted_order` (“<” instead of “≤”) was flagged as wrong-property; the length-only variant of `sorted_permutation` was flagged as weakened-postcondition; the stability variant omitting relative-order checking was flagged as missing-case. This confirms that the 97% confirmation rate on correct specifications is not an artifact of over-permissiveness: the same pipeline that avoids false positives on correct specs reliably disputes structurally weakened ones.

4.5 Cross-Provider Replication

To address the same-model-family concern, we reran B2 and B4 on VERINA replacing model B (sonnet) with GPT-4o via OpenRouter, keeping model A (haiku) and the judge (sonnet) unchanged.

Table 3. Confirmation rates: same-provider vs. cross-provider (VERINA only, 1 run each)

Condition	Same-provider	Cross-provider (Claude+GPT-4o)
Round-robin only (no ambiguity elicitation)	80.0%	68.0%
Full SpecCheck (ambiguity + round-robin)	97.3%	88.0%

Cross-provider round-robin disputes 3 additional tasks in B2 (8 vs. 5 disputes), confirming that same-family models share blind spots that a cross-provider pair surfaces. The B2 drop from 80% to 68% suggests that GPT-4o and Claude diverge more on Lean 4 structural choices—sortedness constraints, multiset equality—than two Claude models do. B4 drops from 97.3% to 88%, with the 3 remaining disputes all triggered by the combined ambiguity+round-robin signal, suggesting they are genuinely hard cases rather than judge idiosyncrasies. The gap between B2 and B4 is similar in both settings (17pp same-provider, 20pp cross-provider), indicating that ambiguity elicitation contributes comparably regardless of model diversity.

4.6 Qualitative Analysis

Case A—Round-robin detects what ambiguity elicitation alone misses (VERINA, merge-sorted-arrays). Model A’s postcondition checks only set membership ($\forall x \in a, x \in result$), while model B additionally requires sorted order and multiset equality. In cross-informalization, model B reading model A’s specification reports that “ordering is not constrained.” Condition B2 correctly disputes this task. Condition B3, in which model A self-informalizes, confirms it—demonstrating the self-confirmation failure mode that cross-validation is designed to detect.

Case B—Ambiguity as a leading indicator (Hypothesis, `str_split_whitespace`). Stage 0 flags three independent assumptions: which characters constitute whitespace, behavior on whitespace-only strings, and what formally constitutes a “run” of whitespace. Both B2 and B3 dispute this task across all runs. The high ambiguity count correctly predicted this as the hardest requirement in the benchmark before any formalization was performed.

5 Discussion and Limitations

5.1 VERINA: Cross-Model Validation Provides Essential Signal

The 20-percentage-point gap between B1 (100%) and B2 (80%) on VERINA shows that cross-model validation catches what single-model assessment misses. B1 confirms every task because the judge accepts its own informalization as faithful, it sees what it produced, not the specification directly. B2 breaks this loop: model B reads model A’s specification and notices what is absent. Five advanced tasks are disputed this way: LCS length, k -most-frequent elements, longest consecutive sequence, peak-valley pattern, and merge-sorted-arrays, where model A writes property-only postconditions and model B independently adds structural constraints like sortedness and multiset equality.

B3 and B4 both confirm 97.3% of tasks, recovering 17.3 points relative to B2. Without deliberately weakened specifications, we cannot rule out that B3/B4 are under-disputing rather than B2 over-disputing. But qualitative inspection supports the former: on the five disputed tasks, model A’s postconditions are incomplete, and model B’s specifications independently capture what the requirements demand. B2 appears to provide a stricter, higher-recall signal; ambiguity elicitation in B3/B4 may soften the judge’s threshold rather than resolve the underlying gap.

5.2 Hypothesis: Single-Model Assessment Is Already Well-Calibrated

On the Hypothesis benchmark, all four conditions fall within three percentage points of each other (84.0–86.7%), with run-to-run variance exceeding condition-to-condition differences. Property-based test suites are short, behaviorally grounded, and straightforward to informalize faithfully. When both models produce specifications with equivalent property coverage, as they do for most stdlib-derived tasks, cross-model disagreement adds no systematic detection benefit. The practical implication is formalism-aware deployment: round-robin cross-validation is essential for proof-assistant specifications, where models diverge on algorithmic structure, but appears unnecessary for simple property-based tests, though larger studies with varied model pairs are needed to confirm this boundary.

5.3 Ambiguity Elicitation as a Leading Indicator

Across both domains, requirements flagged with two or more assumptions in Stage 0 disputed at substantially higher rates than those with zero or one flagged assumptions (Table 3). The sole requirement with three or more flagged assumptions—`str.split_whitespace`—was the only task disputed in every condition across all three runs. This indicates that ambiguity count at elicitation time functions as a practical upstream signal: before any formalization is attempted, users can identify which requirements are likely to produce drift and invest additional clarification effort where it matters most. Critically, this signal is available even in the single-model setting (B3), where cross-model disagreement is not.

5.4 Limitations

Sensitivity evaluated on hand-crafted drift. The 10 deliberately weakened specifications in §4.3 were constructed by the authors, not independently generated by an LLM and verified to be wrong. A stronger sensitivity evaluation would sample LLM-generated specs, have human annotators label them as drifted or faithful, and measure recall on that set. The 100% detection rate on hand-crafted cases establishes that the pipeline is not trivially permissive; it does not establish that all real-world drift patterns are equally detectable.

Same model family and circularity risk. Haiku and sonnet share training data and methodology. Additionally, the same model family both identifies ambiguities at Stage 0 and judges disputes at Stage 3, creating a potential circularity: requirements flagged as ambiguous may be judged more harshly by a model predisposed to find them problematic. A preliminary cross-provider replication (§4.4) with GPT-4o as model B confirms that shared blind spots exist: B2 drops a further 12 points when using a

cross-provider pair, validating this concern. Stages 0 and 3 remain same-family in that run; breaking those dependencies is a direction for future work.

Scale. Fifty tasks per domain with three runs is sufficient to observe the 20-point gap on VERINA, but still modest for the Hypothesis equivalence claim. Ablation conditions (B1–B3) were run on the first 25 tasks only. Larger benchmarks are needed before concluding that cross-model validation adds no value for property-based tests.

Tautology false positives. One B4 run on Hypothesis produced 7 tautology disputes; other B4 runs averaged 0.5. The judge misclassified compact but correct properties—length preservation, element existence—as tautologies when back-translations emphasized their trivial form. Targeted few-shot calibration in the Stage 3 prompt is the direct fix.

5.5 Future Work

A natural next step is scaling the sensitivity evaluation: sampling LLM-generated specs, annotating them for drift, and measuring recall on that set. Beyond this, several techniques from the original claimcheck research agenda remain to be integrated. Formal equivalence checking between independently generated specifications would provide a deterministic complement to LLM judgment; on failed checks, automated counterexample generation could produce concrete inputs distinguishing the two specifications, giving users interpretable evidence of divergence rather than a drift score. Mutation testing—deliberately weakening implementations and checking whether disputed specifications catch the injected bugs—would close the loop between spec quality and bug-finding ability. Longer-term: cross-provider model pairs to amplify the round-robin signal, expansion to Dafny and Verus as target formalisms, and IDE integration so that ambiguity elicitation and validation happen inline during specification authoring.

6 Conclusion

SpecCheck demonstrates that cross-model disagreement is a practical, verifier-free signal for catching intent drift in LLM-generated specifications. The central finding is that this signal’s value depends on the formalism: essential for Lean 4, where models silently diverge on semantic choices, but unnecessary for property-based tests, where single-model assessment is already well-calibrated. This suggests that validation pipelines should be formalism-aware rather than one-size-fits-all. Ambiguity elicitation provides a complementary signal that works in both settings: requirements flagged with multiple implicit assumptions are substantially more likely to produce drift, giving users an actionable warning before any formalization is attempted. The current evaluation is limited in scale and uses same-family models for most stages; larger benchmarks, cross-provider pipelines, and integration with formal equivalence checking are natural next steps.

Code and Data

- **SpecCheck code:** <https://github.com/invi-bhagyesh/SpecCheck>
- **VERINA benchmark:** <https://arxiv.org/abs/2505.23135>

Acknowledgments

This project builds on a [research proposal](#) by Simon Henniger and Nada Amin. We also thank Apart Research and Atlas Computing for organising this event.

LLM Usage Statement

claude-opus-4-6 assisted with code implementation and drafting portions of this report. All experimental design decisions, benchmark construction, result interpretation, and claims were developed and independently verified by the author.

References

- [1] Matthias Cosler, Christopher Hahn, Daniel Mendoza, Frederik Schmitt, and Caroline Trippel. nl2spec: Interactively translating unstructured natural language to temporal logics with large language models. <https://arxiv.org/abs/2303.04864>, 2023.
- [2] Madeline Endres, Sarah Fakhoury, Saikat Chakraborty, and Shuvendu K. Lahiri. Can large language models transform natural language intent into formal method postconditions? <https://arxiv.org/abs/2310.01831>, 2024.
- [3] Felipe Graciolli and Nada Amin. claimcheck: Narrowing the gap between proof and intent. Blog post, <https://midspiral.com/blog/claimcheck-narrowing-the-gap-between-proof-and-intent/>, 2024.
- [4] Lezhi Ma, Shangqing Liu, Yi Li, Xiaofei Xie, and Lei Bu. Specgen: Automated generation of formal program specifications via large language models. <https://arxiv.org/abs/2401.08807>, 2025.
- [5] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. <https://arxiv.org/abs/2303.11366>, 2023.
- [6] Chuyue Sun, Ying Sheng, Oded Padon, and Clark Barrett. Clover: Closed-loop verifiable code generation. <https://arxiv.org/abs/2310.17807>, 2024.
- [7] Zhe Ye, Zhengxu Yan, Jingxuan He, Timothe Kasriel, Kaiyu Yang, and Dawn Song. VERINA: Benchmarking verifiable code generation. <https://arxiv.org/abs/2505.23135>, 2026.

Appendix

A Per-Run Results

Table 4 reports confirmed task counts for each of the three independent runs per condition. Run-to-run variance is low on VERINA (temperature 0, deterministic Lean formalism) and moderate on Hypothesis (stochastic natural-language property generation).

Table 4. Confirmed tasks per run (out of 25) by condition and domain

Domain	Condition	Run 1	Run 2	Run 3	Mean
VERINA	B1 Single-model	25	25	25	25.0
VERINA	B2 Round-robin only	19	22	19	20.0
VERINA	B3 Ambiguity only	25	25	23	24.3
VERINA	B4 Full SpecCheck	25	23	25	24.3
Hypothesis	B1 Single-model	24	20	20	21.3
Hypothesis	B2 Round-robin only	22	21	20	21.0
Hypothesis	B3 Ambiguity only	20	23	22	21.7
Hypothesis	B4 Full SpecCheck	22	18	24	21.3

B Per-Requirement Stability

Table 5. VERINA per-task confirmation across three runs (↓ indicates at least one run disputed)

Task	B1	B2	B3	B4
basic_2 (findSmallest)	3/3	3/3	3/3	3/3
basic_24 (even-odd diff)	3/3	1/3 ↓	3/3	3/3
basic_63 (float threshold)	3/3	2/3 ↓	3/3	3/3
basic_88 (list to array)	3/3	3/3	3/3	3/3
basic_54 (min abs diff)	3/3	3/3	3/3	3/3
basic_48 (perfect square)	3/3	3/3	3/3	3/3
basic_77 (2D array update)	3/3	3/3	3/3	3/3
basic_34 (extract evens)	3/3	3/3	3/3	2/3 ↓
basic_69 (first occurrence)	3/3	3/3	3/3	3/3
basic_85 (reverse array)	3/3	3/3	3/3	3/3
basic_11 (last digit)	3/3	3/3	3/3	3/3
basic_38 (all chars identical)	3/3	3/3	3/3	3/3
basic_75 (min element)	3/3	3/3	3/3	3/3
basic_41 (one distinct)	3/3	3/3	3/3	3/3
basic_10 (greater than all)	3/3	3/3	3/3	3/3
advanced_22 (peak-valley)	3/3	2/3 ↓	3/3	3/3
advanced_3 (LCS length)	3/3	1/3 ↓	3/3	3/3
advanced_37 (majority element)	3/3	3/3	3/3	3/3
advanced_50 (merge sorted)	3/3	0/3 ↓	3/3	2/3 ↓
advanced_6 (all vowels)	3/3	3/3	3/3	3/3
advanced_28 (longest consecutive)	3/3	1/3 ↓	3/3	3/3
advanced_34 (LIS)	3/3	2/3 ↓	3/3	3/3
advanced_76 (k most frequent)	3/3	1/3 ↓	3/3	3/3
advanced_30 (longest incr subarray)	3/3	3/3	3/3	3/3
advanced_42 (max stock profit)	3/3	2/3 ↓	3/3	3/3
Total / 75	75/75	60/75	73/75	73/75

Table 6. Hypothesis per-task confirmation across three runs (↓ indicates at least one run disputed)

Task	B1	B2	B3	B4
sorted_order	3/3	3/3	3/3	3/3
sorted_length	3/3	3/3	3/3	3/3
sorted_permutation	3/3	3/3	3/3	3/3
sorted_stable	3/3	3/3	3/3	3/3
str_split_whitespace	0/3 ↓	0/3 ↓	0/3 ↓	1/3 ↓
str_split_delimiter	3/3	3/3	3/3	3/3
str_split_maxsplit	2/3 ↓	0/3 ↓	1/3 ↓	3/3
list_index	3/3	3/3	3/3	3/3
list_reverse	3/3	3/3	3/3	3/3
set_union	3/3	2/3 ↓	1/3 ↓	3/3
set_intersection	3/3	3/3	3/3	3/3
set_difference	3/3	3/3	3/3	3/3
dict_get	3/3	3/3	2/3 ↓	3/3
dict_update	3/3	3/3	3/3	3/3
counter_mostcommon	2/3 ↓	1/3 ↓	2/3 ↓	2/3 ↓
dedup_order	2/3 ↓	1/3 ↓	2/3 ↓	2/3 ↓
binary_search	3/3	3/3	3/3	2/3 ↓
flatten	3/3	3/3	3/3	2/3 ↓
running_max	3/3	3/3	3/3	2/3 ↓
group_by	3/3	2/3 ↓	3/3	1/3 ↓
rotate_list	3/3	3/3	3/3	2/3 ↓
zip_lists	3/3	3/3	3/3	3/3
max_value	3/3	3/3	3/3	3/3
count_occurrences	3/3	3/3	3/3	3/3
chunk_list	3/3	3/3	3/3	2/3 ↓
Total / 75	64/75	63/75	65/75	64/75