

SpecFault-Dafny: A Class-Stratified Scorecard for Verifier-Passing Specification Faults

Research conducted at The Secure Program Synthesis Hackathon, May 22-24, 2026

Hugo Nguyen, Independent

May 2026

Abstract

Formal verification can approve a Dafny program whose contract omits, shifts, or hides the user's intended behavior. SpecFault-Dafny targets that gap for secure AI coding. It contributes a newly built 45-item Dafny corpus: 30 verifier-passing intent/specification mismatches and 15 clean controls, stratified across six failure classes and nine small domains. The package also includes five baseline validators, a reproducible runner, generated scorecards, and evidence cards for every scored item. On the final run, verifier-only has 0.00% bad recall, static escape scanning 13.33%, symbolic checks 86.67%, round-trip intent/spec comparison 93.33%, and a fixed hybrid 100.00%, with 0.00% observed clean-control false-positive rate for all validators. The LLM baseline replays compact frozen summaries from run 20260524T205027Z-llmfinal, so reported coverage is reproducible without publishing raw prompts or responses. The takeaway is a compact benchmark for comparing specification-validation signals by failure class, rather than a deployment-ready detector, for reviewers, future tools, and repeatable regression tests.

1. Introduction & Motivation

Secure program synthesis depends on more than code that verifies. A verifier checks an implementation against the contract it was given. The open question is whether that contract captured the intended behavior, and turning a stated intent into a correct formal specification is itself a hard problem for reliable AI coding [10]. The specification-validation track of the Secure Program Synthesis Hackathon asks exactly this: whether a candidate specification captures intended behavior, with a working artifact and research report as the target output.

Dafny gives a clear experimental unit. A candidate program runs through `dafny verify`, while its benchmark metadata stores a natural-language intent and a reviewed label. Every bad item in this corpus passes verification, so the failure lies in the relation between intent and contract.

SpecFault-Dafny contributes:

1. A 45-item Dafny benchmark with 30 verifier-passing intent/specification mismatches, 15 clean controls, six bad classes, and nine domains.
2. Five baseline validators: verifier-only acceptance, static escape-hatch scanning, Claimcheck-style round-trip comparison, symbolic validation checks, and a fixed-order hybrid.
3. Reproducible scorecard, corpus audit, and evidence cards that make each item inspectable from intent to validator signal.

All corpus items, validators, scorecards, the evidence-card generator, and the report artifacts were newly built during the May 22-24, 2026 hackathon window. Prior work was used only for framing, validator baselines, and comparison points; no prior benchmark payload was imported into the scored corpus.

The goal is a narrow comparison artifact. It gives future secure-synthesis systems a regression target: a system should pass verification and also leave evidence about whether the contract says what the user meant.

2. Related Work

Claimcheck motivates the round-trip baseline: informalize a formal Dafny claim without showing the original requirement, then compare the back-translation to the requirement [1]. SpecFault-Dafny keeps the same structural idea, but uses it as one baseline inside a corpus-level scorecard with clean controls and per-class recall.

SPOTs, small proof-oriented tests, motivate the symbolic-check baseline [2]. SpecFault-Dafny packages each item with positive and negative Dafny harnesses where available, then scores those harnesses as a validator. SPOTs are a proof-engineering technique; this project is an evaluation artifact for comparing multiple validation styles on the same items.

Prior Dafny specification-evaluation work uses symbolic tests to evaluate whether a formal specification captures user intent [3]. IronSpec studies specification reliability through sanity checks, Spec-Testing Proofs, and spec mutation testing [4]. MutDafny mutates Dafny implementations and treats surviving mutants as potential weak-spec signals [5]. Clover checks consistency among code, docstrings, and annotations for Dafny artifacts [6]. DafnyBench focuses on making Dafny verification succeed and discusses proof-bypass filters, which informed the static escape-hatch baseline [7]. RESpecBench motivates the ground-truth boundary: LLM judgments are treated as validator signals, while labels come from reviewed metadata and verifier-backed checks [8]. Recent Alloy validation work shows how positive and negative formal test cases can help detect wrong specifications, with LLMs generating the test cases while final judgments stay with the formal checks [9].

The contribution is the benchmark unit: class-stratified, verifier-passing Dafny intent faults with clean controls, scored across five validators by coverage, clean-control false-positive rate, and bad-class recall.

3. Approach / Methodology

Each benchmark item bundles its metadata, a candidate Dafny program, an optional corrected version, and optional positive and negative proof harnesses. The score command is the only code path that reads labels; each validator reads only its permitted inputs.

Bad items use six primary classes:

Class	Meaning in this corpus
weak_postcondition	The contract permits behavior outside the stated intent.
wrong_property	The contract proves the wrong semantic property.
narrowed_scope	The contract covers a narrower case than the intent.
boundary_gap	Edge cases are omitted or mishandled by the contract.
precondition_shift	The contract rejects inputs that the intent includes.
escape_hatch	Trust or proof-bypass constructs make obligations vacuous.

The five validators are intentionally simple:

Validator	Input and rule
verifier_only	Runs <code>dafny verify</code> ; a passing candidate is predicted clean.
static_escape	Scans <code>candidate.dfy</code> for explicit trust or proof-bypass patterns.
roundtrip_claimcheck	Summarizes <code>candidate.dfy</code> , compares summary to intent, and uses frozen confidence thresholds.
symbolic_checks	Runs item-provided positive and negative Dafny harnesses.
hybrid	Applies <code>static -> symbolic -> roundtrip -> clean</code> in a fixed order.

The round-trip baseline uses gpt-5.1 at temperature 0 with a fixed 900-token output limit and frozen prompts. The public artifact keeps compact parsed summary JSON for the final LLM run, including input hashes, prompt hashes, model parameters, parsed stage outputs, and classifier summaries. Every scored item replays from those summaries, so the scorecard reproduces without live API calls or public raw prompt/response transcripts. The run identifier and Dafny build are recorded with the generated artifacts.

4. Results

The scorecard covers all 45 scored items. No item was invalid, every prediction came from run 20260524T205027Z-llmfinal, and every validator covered every scored item. Table 1 keeps the main PDF readable by reporting the two headline safety metrics; the full scorecard, per-class recall table, and corpus composition are in the Appendix.

Table 1. Headline validator metrics.

Validator	Bad recall	Clean FPR	Main signal
hybrid	100.00%	0.00%	Fixed union of static, symbolic, then round-trip signals.
roundtrip_claimcheck	93.33%	0.00%	LLM summary compared with the natural-language intent.
symbolic_checks	86.67%	0.00%	Positive and negative Dafny harnesses.
static_escape	13.33%	0.00%	Explicit trust or proof-bypass patterns.
verifier_only	0.00%	0.00%	Dafny verification success only.

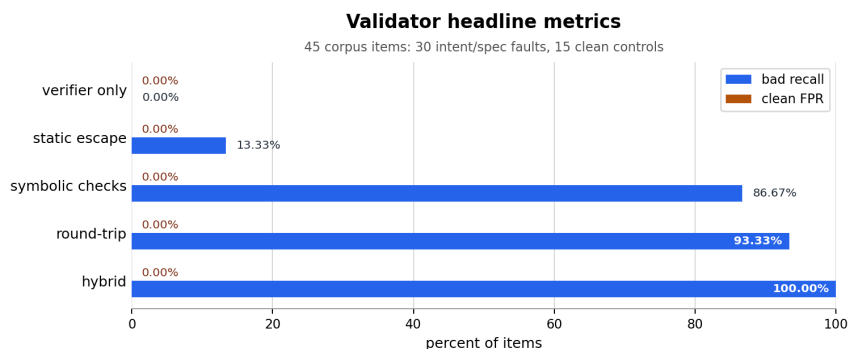


Figure 1. Bad-item recall and clean-control false-positive rate by validator on the 45-item SpecFault-Dafny corpus. The bars show complementary validator behavior: static scanning catches only explicit escape hatches, symbolic checks and round-trip comparison catch different semantic subsets, and the fixed hybrid combines them.

The main observation is signal complementarity. The verifier-only baseline accepts all verifier-passing candidates and therefore misses all 30 bad items. The static escape-hatch scanner has narrow coverage of semantic classes but captures all four escape-hatch items. Symbolic checks catch every bad class except escape-hatch in this run. Round-trip comparison covers every scored item, flags 28 of 30 bad items, and produces no observed clean-control false positives. Its two misses fall in the escape-hatch class, where it flags two of the four items for 50.00% recall. The fixed hybrid combines static escape scanning, symbolic checks, and round-trip signals, which explains its observed 100.00% bad recall on this corpus.

5. Evidence Cards

Three cards illustrate the corpus: a weak-postcondition fault, an escape-hatch fault, and a clean control. Each card pairs the stated intent with the candidate contract, explains why Dafny accepts it, and names the mismatch and the minimal repair.

Card A, weak postcondition:

Field	Value
Intent	Return one of the two inputs whose value is at least the other input.
Candidate contract	ensures $m \geq a$; ensures $m \geq b$
Why Dafny accepts	The implementation returns the larger branch value, so both lower-bound postconditions are provable.
Mismatch	The contract permits a value greater than both inputs even though that value was not supplied.
Validator signals	Verifier-only: clean. Static escape-hatch: clean. Symbolic checks: bad. Round-trip: bad. Hybrid: bad.
Minimal repair	Require the result to equal one of the two inputs.

Card B, escape hatch:

Field	Value
Intent	Advance a counter by one step.
Candidate construct	assume { :axiom } false; next := c; under ensures next == c + 1
Why Dafny accepts	The axiom-annotated assume statement makes the remaining proof obligations vacuous.
Mismatch	The body assigns c to next, so the stated transition is absent from the executable assignment.
Validator signals	Verifier-only: clean. Static escape-hatch: bad. Symbolic checks: clean. Round-trip: bad. Hybrid: bad.
Minimal repair	Remove the assume statement and assign next := c + 1.

Card C, clean control:

Field	Value
Intent	Return one of the two inputs whose value is at least the other input.
Candidate contract	Adds ensures $m == a \mid \mid m == b$ to the lower-bound postconditions.
Validator signals	All validators predict clean, including round-trip comparison.
Role	Checks that the scorecard includes clean controls alongside bad items.

6. Limitations and Dual-Use Considerations

The corpus is compact, synthetic, Dafny-only, and designed for inspection rather than statistical generalization. Its class balance is explicit: 30 bad items, 15 clean controls, six bad classes, and nine domains. Labels were authored and reviewed inside the project workflow, but not adjudicated by an external panel of Dafny experts.

False positives remain possible even though this run observed 0.00% clean FPR. Round-trip comparison can overread intent or summaries, static scanning can flag benign controlled uses of suspicious constructs, symbolic harnesses can encode the wrong expectation, and the hybrid inherits false positives from whichever upstream signal fires first.

False negatives are also visible. `verifier_only` misses all bad items by construction, `static_escape` misses ordinary semantic mismatch, symbolic checks miss all four `escape_hatch` items in this run, and `roundtrip_claimcheck` misses two of those four escape-hatch items. Boundary, precondition, and narrowed-scope edge cases are especially sensitive to how the natural-language intent and harness examples phrase the included inputs.

The main scalability constraint is item authorship. Every new item needs a reviewed intent, a verifying candidate, a label rationale, and preferably a fixed version plus positive and negative harnesses. The LLM baseline is also model-sensitive: the reported 93.33% bad recall is a cached `gpt-5.1` result on this corpus, not a general claim about LLM judgment. Future improvements should add external label adjudication, mutation-derived variants, larger domain coverage, and repeated model snapshots before treating the scorecard as representative.

The dual-use risk is narrow but real. Examples that expose weak contracts can also teach how to craft contracts that evade simple validators. The mitigation is to publish the artifact as a transparent benchmark with clean controls, limitations, and reproducible metrics, and to avoid presenting any baseline as a deployment gate for high-assurance systems. The project uses toy, synthetic, or pattern-authored Dafny examples and avoids live software systems or fresh vulnerability disclosure.

7. Future Work

Useful next improvements are external label adjudication, larger waves from real Dafny projects, multiple independently authored symbolic-check suites, model- and prompt-stability studies for the round-trip baseline, and comparison against mutation-based Dafny signals such as MutDafny-style surviving mutants. Future scorecards should track false positives and false negatives over new corpus waves instead of relying on the current zero-observed-clean-FP result.

8. Conclusion

SpecFault-Dafny turns the intent/specification gap into a small, inspectable benchmark. Every scored candidate verifies, yet 30 items encode reviewed intent/specification mismatches. The scorecard shows that verifier acceptance alone gives no bad-item signal on this corpus, static escape scanning is useful for explicit trust escapes, symbolic checks cover most semantic classes, and a fixed hybrid combines those signals on the current item set.

The artifact is useful because it is concrete: judges can inspect a candidate that verifies, read the intended behavior, see why the contract misses it, and compare validator outputs in one scorecard.

Code and Data

- **Code repository:** <https://github.com/SystemicVoid/secure-synth-hackathon>.
- **Data/Datasets:** the 45-item annotated Dafny corpus is under `specfault-dafny/bench/items`; final public artifacts include `artifacts/results/predictions.jsonl`, `artifacts/results/scorecard.json`, `artifacts/results/verify.jsonl`, and `artifacts/cards/*.md`.
- **Other artifacts:** the generated corpus audit and report tables are under `specfault-dafny/notes/` and `specfault-dafny/report/tables/`; the optional demo script is in `submission/demo_script.md`. Raw API prompt/response transcripts are not required for checking the submitted scorecard and are not claimed as public data.

Author Contributions

Hugo Nguyen directed project scope and reviewed priorities. Codex agents implemented the benchmark tooling, validators, corpus, review loops, generated artifacts, and report under that direction. CodeRabbit CLI reviews served as the independent review gate where available, with Codex review as a fallback.

References

- [1] Apart Research. 2026. The Secure Program Synthesis Hackathon. <https://apartresearch.com/sprints/secure-program-synthesis-hackathon-2026-05-22-to-2026-05-24>.
- [1] Fernanda Graciolli and Nada Amin. 2026. claimcheck: Narrowing the Gap between Proof and Intent. Midspiral blog. <https://midspiral.com/blog/claimcheck-narrowing-the-gap-between-proof-and-intent>.
- [2] Nik Swamy and Shuvendu Lahiri. 2026. Spotting Specification Gaps with Small Proof-Oriented Tests. RiSE blog. <https://risemr.github.io/blog/2026-04-16-spotting-specs>.
- [3] Shuvendu K. Lahiri. 2024. Evaluating LLM-driven User-Intent Formalization for Verification-Aware Languages. arXiv:2406.09757. <https://arxiv.org/abs/2406.09757>.

- [4] Eli Goldweber, Weixin Yu, Seyed Armin Vakil Ghahani, and Manos Kapritsos. 2024. IronSpec: Increasing the Reliability of Formal Specifications. OSDI 2024. <https://www.usenix.org/conference/osdi24/presentation/goldweber>.
- [5] Isabel Amaral, Alexandra Mendes, and Jose Campos. MutDafny: A Mutation-Based Approach to Assess Dafny Specifications. arXiv:2511.15403. <https://arxiv.org/abs/2511.15403>.
- [6] Chuyue Sun, Oded Padon, Ying Sheng, and Clark Barrett. 2023. Clover: Closed-Loop Verifiable Code Generation. arXiv:2310.17807. <https://arxiv.org/abs/2310.17807>.
- [7] Chloe Loughridge et al. 2024. DafnyBench: A Benchmark for Formal Software Verification. arXiv:2406.08467. <https://arxiv.org/abs/2406.08467>.
- [8] Anonymous authors. 2026. RESpecBench: How Reliable is LLM-as-a-Judge? Rigorous Evaluation of Specification Generation with Automated Verification. OpenReview, ICLR 2026 submission. <https://openreview.net/forum?id=eFwJZIN9eI>.
- [9] Alcino Cunha and Nuno Macedo. 2026. Validating Formal Specifications with LLM-Generated Test Cases. FM 2026. <https://arxiv.org/abs/2510.23350>.
- [10] Shuvendu K. Lahiri. 2026. Intent Formalization: A Grand Challenge for Reliable Coding in the Age of AI Agents. arXiv:2603.17150. <https://arxiv.org/abs/2603.17150>.

Appendix

Full Scorecard

Validator	Coverage	Bad recall	Clean FPR	Precision	F1	Macro bad-class recall	Invalid count	Mean seconds/item
hybrid	100.00%	100.00%	0.00%	100.00%	100.00%	100.00%	0	0.0001
roundtrip_claimcheck	100.00%	93.33%	0.00%	100.00%	96.55%	91.67%	0	0.0002
static_escape	100.00%	13.33%	0.00%	100.00%	23.53%	16.67%	0	0.0001
symbolic_checks	100.00%	86.67%	0.00%	100.00%	92.86%	83.33%	0	0.9825
verifier_only	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0	0.4750

Per-Class Bad Recall

Validator	weak_postcondition	wrong_property	narrowed_scope	boundary_gap	precondition_shift	escape_hatch
hybrid	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
roundtrip_claimcheck	100.00%	100.00%	100.00%	100.00%	100.00%	50.00%
static_escape	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%
symbolic_checks	100.00%	100.00%	100.00%	100.00%	100.00%	0.00%
verifier_only	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%

The full corpus-composition table is tracked at `specfault-dafny/report/tables/corpus_composition.md`.

Reproduction

From the repository root, run the pipeline from the package directory:

```
cd specfault-dafny
uv sync --extra dev
uv run specfault lint-items bench/items
uv run specfault verify bench/items --out artifacts/results/verify.jsonl
uv run python scripts/validate_roundtrip_cache.py \
```

```
20260524T205027Z-llmfinal artifacts/results/predictions.jsonl
uv run specfault score bench/items artifacts/results/predictions.jsonl \
  --out artifacts/results/scorecard.json
uv run specfault cards bench/items artifacts/results/predictions.jsonl --
out artifacts/cards
uv run specfault audit bench/items artifacts/results/predictions.jsonl --
out notes/corpus_audit.md
```

The submitted predictions are tracked. If you want to rerun the LLM-backed round-trip validator, `make run` can provision a live cache from `OPENAI_API_KEY`, but live calls are not required for reproducing the submitted scorecard.

LLM Usage Statement

LLM agents implemented the runner, validators, corpus, generated artifacts, reviews, and report drafts under human direction. Every numeric claim in this report was checked against the generated scorecard, the corpus audit, and the round-trip outputs, with verification and review evidence recorded in the repository. The final submission text was edited against the event template and checked against the rendered PDF.

Public repository: <https://github.com/SystemicVoid/secure-synth-hackathon>.