

Verified but Wrong: Auditing When Vericoding Verifies the Wrong Target

Philip Nilsson
Independent Researcher

Abstract

Vericoding can prove that generated code satisfies a supplied formal specification, but not that the specification is the right target. We study target validity: whether a formal target preserves natural-language or policy intent before candidate selection. We audit a public Dafny vericoding benchmark snapshot with 2,334 records and 1,448 useful natural-language descriptions. A scanner found 128 candidate target-validity gaps; adjudicating all 41 high-confidence candidates produced 11 verified-but-wrong demonstrations, including 3 direct-Dafny cases where deliberately wrong implementations verify with zero errors and fail intended examples. Repaired targets block all three Tier 1 cases. A controlled 12-task benchmark isolates the mechanism: naive specs produced 12/12 verified-but-wrong selections, checklist-critic specs 5/12, and oracle/full specs 0/12. These results are demonstration evidence, not prevalence estimates. They motivate target-validity auditing before using formal specs for vericoding selection.

1. Introduction

Formal verification can tell us whether code satisfies a specification. Vericoding extends this into program synthesis: generate candidate implementations, verify them against a formal target, and select code that passes.

But verification against a formal target does not prove that the target was the right one.

If the formal target omits an intent-critical requirement, then verification may certify a program that is correct relative to the written spec and wrong relative to the user's intent. We call this outcome *verified-but-wrong*: an implementation passes the public formal specification used for candidate selection, but fails the natural-language, policy, or product intent that the formal specification was meant to capture.

Example. In DA0208, the natural-language task asks for the minimum uniform box size needed to pack cowbells. The reconstructed public target only requires a nonnegative result. A candidate that always returns 0 verifies with Dafny reporting zero errors, but fails intended examples expecting 5, 10, and 10. Adding the missing target constraint blocks the same bad candidate.

The claim is not that weak specifications exist. That is already known. The vericoding-specific failure is that weak specifications become trusted selection targets. Verification can then amplify confidence in code that is only correct relative to an incomplete target.

Existing vericoding benchmarks primarily evaluate implementation validity against a supplied formal target. We evaluate target validity itself: whether the supplied formal target preserves natural-language or policy intent before it is used for candidate selection.

Contributions.

- **Failure model.** Defines verified-but-wrong as implementation validity passing while target validity fails.
- **External replication suite.** Audits 1,448 useful NL/Dafny tasks from a public benchmark and validates 11 externally sourced demonstrations, including 3 direct-Dafny cases.
- **Controlled mechanism benchmark.** Shows that incomplete public specs select wrong implementations in 12/12 naive tasks, reduced to 5/12 with checklist-critic specs and 0/12 with oracle/full specs.
- **Target-validity audit baseline.** Evaluates a simple pre-selection Policy Audit Gate that catches 17/17 dangerous specs in the controlled setting.

The main contribution is not the gate. The main contribution is the missing evaluation layer: target validity before implementation selection.

Scope of claims. This paper does not claim that verified-but-wrong failures are common across all vericoding benchmarks, nor that Dafny verification is unsound. The narrow claim is that target-validity failures can occur when a formal public target omits requirements present in the natural-language or policy intent, and that vericoding pipelines can amplify this failure by using the incomplete target for candidate selection. The three direct-Dafny cases establish the strongest evidence for simple underconstraint classes, while the Tier 2 adapted demonstrations support broader omission patterns such as missing simulation semantics, missing minimality, missing exact-value constraints, and related target gaps.

2. Background and Failure Model

Vericoding benchmarks evaluate whether models can synthesize programs and proofs that satisfy formal specifications. DafnyBench, FVAPPS, and the Vericoding benchmark all test whether generated artifacts verify against supplied formal targets.

This paper studies a complementary question. Instead of asking whether a model can satisfy a formal target, we ask whether the formal target preserves the natural-language or policy intent it is supposed to represent.

The issue is related to specification validation, CEGIS, CEGAR, executable oracles, and spec-conditioned decoding. However, the failure mode is different in a vericoding pipeline. In ordinary implementation debugging, an incomplete spec may cause missed bugs. In vericoding, the incomplete spec can become the object of optimization and selection. A candidate can be selected because it satisfies the wrong target.

Successful formal verification may also increase user trust. A program that is merely untested can be questioned. A program that is formally verified may be trusted too quickly, even when the formal target omitted the requirement that mattered.

Layer	Question	Failure mode
Implementation validity	Does the code satisfy the supplied formal/public spec?	The implementation fails the written target.
Target validity	Does the supplied spec preserve the intended requirement?	The spec is too weak, so verified code can still be wrong.

A verified-but-wrong outcome occurs when implementation validity passes but target validity fails.

Verified-but-Wrong failure path

Formal verification can pass while target validity fails.

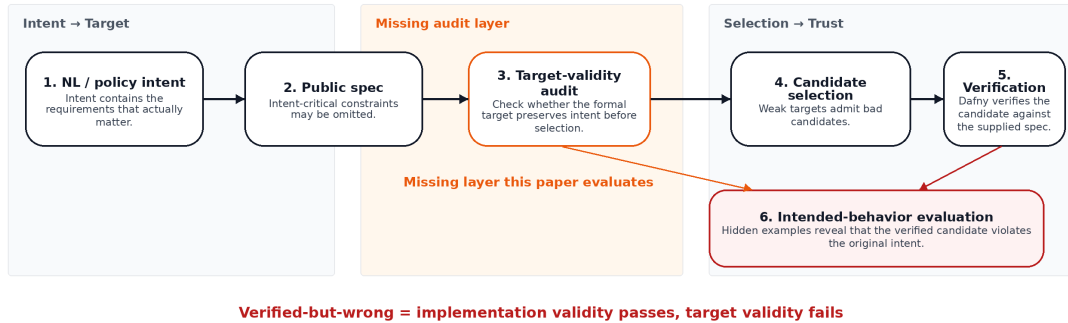


Figure 1. Verified-but-wrong failure path. Formal verification can establish implementation validity against the supplied public target, while target validity still fails if the target omits intent-critical requirements. The missing layer is a pre-selection audit that checks whether the formal target preserves natural-language or policy intent before candidate selection.

Artifact separation.

Artifact	Purpose	Visible to audit gate?	Used for selection?	Used for evaluation?
Public spec	Candidate-selection target	Yes	Yes	No
NL intent or policy pack	Requirement inventory	Yes	Audit only	No
Hidden oracle	Intended-behavior evaluator	No	No	Yes
Selected candidate	Output of candidate selection	No before selection	Yes	Yes

The target-validity audit is not allowed to read the hidden oracle or inspect which candidate will be selected. It only checks whether the public spec appears to preserve requirements already present in the natural-language intent or policy inventory.

3. Methods

We evaluate verified-but-wrong in three parts: an external audit pipeline, a scanner-ranked external replication suite, and a controlled benchmark with a simple pre-selection target-audit baseline.

3.1 External audit pipeline

Field	Value
Source	Beneficial-AI-Foundation/vericoding-benchmark
Commit	349ee510deeb6e15e1eb0d98174914e7093252fd
Dafny records	2,334
Useful natural-language descriptions	1,448
APPS useful descriptions	676
HumanEval useful descriptions	157

We treat `vc-description` as the natural-language intent and `vc-spec` plus `vc-preamble` as the formal target.

Stage	Input	Rule	Output
Benchmark parse	<code>vc-description</code> , <code>vc-spec</code> , <code>vc-preamble</code>	Retain useful NL/Dafny pairs.	1,448

Stage	Input	Rule	Output
Gap scan	NL intent + formal target	Detect cases where intent appears stronger than formal constraints.	128
High-confidence filter	Candidate gaps	Require clear NL intent, clear formal omission, plausible bad candidate.	41
Frozen adjudication	All high-confidence cases	Apply fixed Tier / reject / deferred labels.	11 validated

Candidate gaps were categorized as trivial postconditions, missing exact-value constraints, missing optimization requirements, missing simulation semantics, missing tie-breaks, output-format gaps, or related target omissions. A case was counted only if manual inspection found clear natural-language intent, a clear formal-spec gap, plausible bad candidate behavior, and executable intended examples.

3.2 External replication suite and evidence tiers

To reduce cherry-picking risk, we adjudicated all 41 high-confidence scanner candidates under a frozen labeling protocol. Each candidate was labeled as Tier 1 direct-Dafny verified-but-wrong, Tier 1 with repaired-target blocking, Tier 2 adapted executable demonstration, rejected, or deferred. The external suite is scanner-ranked demonstration evidence, not a prevalence estimate.

Tier	Meaning	Strength
Tier 1: Direct original-target Dafny verification	A deliberately wrong Dafny implementation verifies against the reconstructed original benchmark target and fails intended-behavior examples.	Strongest evidence.
Tier 2: Adapted executable demonstration	A bad candidate passes an adapted spec check and fails intended-behavior examples.	Supportive evidence, weaker than Tier 1.

Only Tier 1 cases are claimed as direct verification against the reconstructed original benchmark target. The three Tier 1 cases establish simple underconstraint demonstrations: DA0003 and DA0208 are trivial-postcondition cases, while DA0293 is a range-only target case. Tier 2 cases support broader omission patterns, including missing simulation semantics, missing minimality, missing exact-value constraints, and related target gaps.

For Tier 1 cases, we also test whether adding the missing target-validity condition blocks the same bad candidate. A repaired-target check passes if the original reconstructed target verifies the bad candidate with zero errors, and the repaired target fails verification on the same bad candidate because the repaired constraint is not satisfied. These are local mechanism checks, not complete repaired benchmark specifications.

3.3 Controlled benchmark and baseline mitigation

The controlled benchmark contains 12 executable tasks: 8 development tasks and 4 heldout tasks. Each task contains a public spec, a separate policy pack, deterministic candidate implementations, and a hidden executable oracle representing intended behavior.

Each task is evaluated under three spec modes: naive incomplete specs, checklist-critic specs with common omissions addressed, and oracle/full specs containing the intended requirements. Candidate selection is deterministic: the harness runs public-spec tests against all candidates, keeps candidates that pass, and selects the first passing candidate in a fixed task order. The hidden oracle runs only after selection. Because deterministic ordering could otherwise explain the results, we also analyze the full public-spec-passing candidate set.

The Policy Audit Gate is a deliberately simple pre-selection baseline. It compares a public spec against a separate policy-card inventory and returns ALLOW, REVIEW, or BLOCK. Both REVIEW and BLOCK stop automatic implementation selection. We use the gate only as a simple known-policy baseline. It is not the main proposed solution and does not discover unknown requirements.

The Policy Audit Gate is a deterministic pre-selection checker that audits public spec text against task-linked policy cards from configured policy packs. For each policy card, it assigns covered, unclear, or missing using explicit keyword/term matching, then applies fixed decision rules: missing critical or load-bearing requirements trigger BLOCK, unclear high/critical requirements or low high-severity coverage trigger REVIEW, otherwise ALLOW. In controlled calibration, hidden-oracle outcomes are used only as external labels for metrics; hidden-oracle signals are not used to compute pre-selection decisions.

4. Results

Main result. We find verified-but-wrong target-validity failures in public benchmark artifacts and reproduce the mechanism in a controlled benchmark. In the external audit, 11 demonstrations were validated, including 3 direct-Dafny cases where bad candidates verify with zero errors. In the controlled benchmark, naive specs produced 12/12 verified-but-wrong selections, while oracle/full specs produced 0/12.

Evidence stack for verified-but-wrong target-validity failures

External benchmark audit + controlled mechanism benchmark + mitigation baseline.

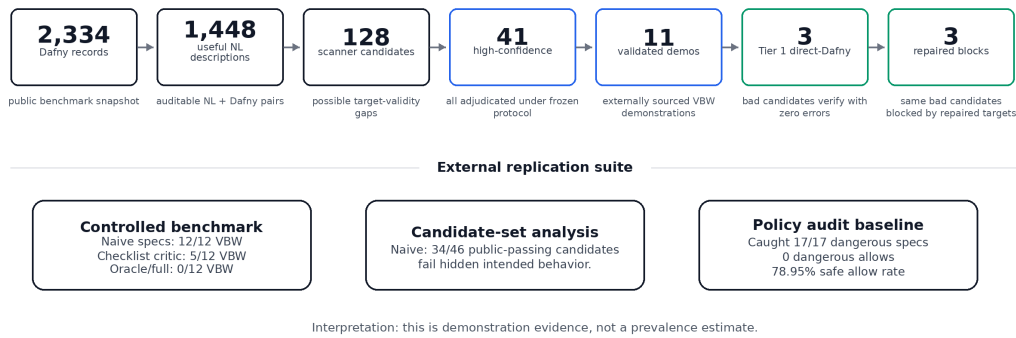


Figure 2. Evidence stack for verified-but-wrong target-validity failures. The external replication suite provides demonstration evidence from public benchmark artifacts, including three direct-Dafny cases and three repaired-target blocking checks. The controlled benchmark isolates the mechanism, while the candidate-set analysis shows that incomplete targets underconstrain the candidate pool rather than merely selecting one unlucky candidate.

4.1 External replication results

The scanner found 128 candidate target-validity gaps among 1,448 useful-description Dafny tasks. We adjudicated all 41 high-confidence scanner candidates under the frozen external replication protocol. DA0293 was subsequently promoted from Tier 2 adapted demonstration to Tier 1 direct-Dafny after a bounded direct-Dafny evidence pass. Final submission counts are 11 validated demonstrations: 3 Tier 1, 8 Tier 2, and 3 repaired-target blocks.

Manual adjudication outcome	Count
Tier 1 direct-Dafny demonstrations	3
Tier 2 adapted demonstrations	8
Rejected	7
Deferred	23
Total high-confidence adjudicated	41

The validated cases include DA0003, DA0010, DA0157, DA0208, DA0244, DA0293, DA0306, and four additional Tier 2 cases recorded in the external replication report. Rejected cases were not counted when manual inspection showed that the formal target was stronger than the scanner assumed, the natural-language intent was ambiguous, or the bad candidate was not clean enough for a defensible demonstration.

Deferred cases are not interpreted as likely positives or likely negatives. In this bounded pass, they required additional manual proof or reconstruction work before they could be defensibly promoted to Tier 1, Tier 2, or rejection. We report them separately to avoid converting unresolved cases into evidence.

4.2 Representative external demonstrations

Case	NL intent	Formal target gap	Bad candidate	Tier	Examples / repair
DA0003	Maximize total chocolates under divisibility and overlap rules.	Postcondition only requires result ≥ 0 .	Always returns 0.	Tier 1	Fails expected 13, 28, 32; repaired target blocks.
DA0010	Simulate champagne overflow and count full glasses after t seconds.	Range and edge-case constraints, but no transition semantics.	Returns 0 for $t=0$, else 1.	Tier 2	Fails expected 1, 3, 4.
DA0157	Find minimum rectangle area satisfying area and growth constraints.	Validity constraints but no minimality condition.	Outputs oversized valid rectangle.	Tier 2	Fails expected 6, 30, 63.
DA0208	Find minimum uniform box size for packing cowbells with at most 2 per box and at most k boxes.	Postcondition only requires result ≥ 0 .	Always returns 0.	Tier 1	Fails expected 5, 10, 10; repaired target blocks.
DA0293	Maximize consecutive 1s after up to K contiguous flips.	Postcondition only constrains $0 \leq \text{result} \leq N$; no optimization/flip semantics.	Always returns 0.	Tier 1	Fails expected 3, 5, 4; repaired target blocks.

These are externally sourced target-validity demonstrations. They are not claims that benchmark authors acted incorrectly. The narrow claim is that in these cases, the paired formal target is weaker than the natural-language intent it is meant to capture.

4.3 Tier 1 direct-Dafny case card

Tier 1 example: verified against the wrong target

DA0208: a trivial candidate verifies against the public target but fails intended behavior.

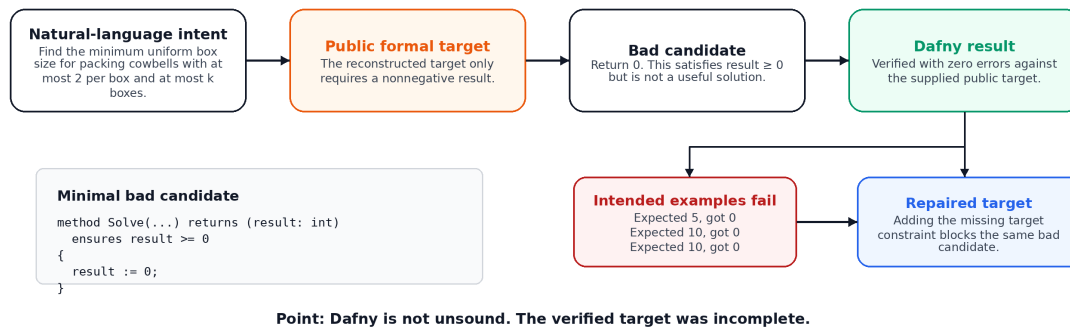


Figure 3. Tier 1 direct-Dafny example. In DA0208, a deliberately wrong candidate that always returns 0 verifies against the reconstructed public target because the target only requires a nonnegative result. The same candidate fails intended-behavior examples and is blocked by the repaired target.

DA0003, DA0208, and DA0293 are the three Tier 1 direct-Dafny cases. The Tier 1 cases remain simple underconstraint demonstrations: DA0003 and DA0208 are trivial-postcondition cases, while DA0293 is a range-only target case that admits the same deliberately wrong candidate pattern.

Case	Formal target gap	Bad candidate	Original target	Intended examples	Repaired target
DA0003	Requires only nonnegative result.	Return 0.	Verifies with zero errors.	Fails expected 13, 28, 32.	Blocks same bad candidate.
DA0208	Requires only nonnegative result.	Return 0.	Verifies with zero errors.	Fails expected 5, 10, 10.	Blocks same bad candidate.
DA0293	Requires only $0 \leq \text{result} \leq N$; omits optimization/flip semantics.	Return 0.	Verifies with zero errors.	Fails expected 3, 5, 4.	Blocks same bad candidate.

Minimal shape of the verified bad candidate:

```

method Solve(...) returns (result: int)
  ensures result >= 0
{
  result := 0;
}

```

The candidate is intentionally trivial. That is the point. If the formal target only requires weak range or nonnegativity conditions, a useless implementation can verify. The failure is not in Dafny's implementation-validity check. The failure is in the target used for verification.

We attempted to construct a direct-Dafny Tier 1 case for DA0157, a missing-minimality target gap. The local verification run timed out, so DA0157 is not counted as Tier 1 evidence. This limitation matters: the confirmed Tier 1 cases directly verify the mechanism for simple underconstraint classes, while Tier 2 cases support broader omission patterns.

4.4 Controlled benchmark and candidate-set analysis

Spec mode	Selected verified-but-wrong outcomes
Naive	12/12
Checklist critic	5/12
Oracle/full	0/12

Naive specs fail by construction as a stress-test condition. This is not a prevalence estimate. It shows a structural possibility: when public specs omit intent-critical requirements, public-spec selection can choose implementations that are locally valid and globally wrong.

Spec mode	Public-passing candidates	Hidden-failing among public-passing	Mean underconstraint risk	Selected VBW
Naive	46	34	0.7361	12/12
Checklist critic	24	12	0.2917	5/12
Oracle/full	-	0	0.0000	0/12

Under naive specs, 34 of 46 public-spec-passing candidates failed hidden intended behavior. This shows that the public spec underconstrained the candidate set itself. The selected candidate was not the only problem; many candidates that satisfied the public target were wrong under intended behavior.

4.5 Baseline mitigation: pre-selection target audit

Actual class	ALLOW	REVIEW	BLOCK	Total
Dangerous	0	12	5	17
Safe	15	0	4	19

The default gate allowed 0 dangerous specs and caught 17/17 dangerous specs. It allowed 15/19 safe specs, for a safe allow rate of 78.95%. The cost is that 4/19 safe specs were blocked.

Method	Naive VBW after repair	Critic VBW after repair	Interpretation
No repair	12	5	Original failure rate.
Generic repair	11	5	Generic patch barely helps.
Checklist repair	10	5	Checklist patch helps little.
Policy-targeted repair	0	0	Adds missing policy-card requirement.
Oracle repair	0	0	Upper-bound full-spec repair.

Robustness checks support the same narrow interpretation. Irrelevant low-severity noise did not change the aggregate outcome. Outdated wording preserved safety but collapsed usability, and removing 50% of relevant policy cards allowed 2 dangerous specs. The gate helps only when relevant policies are represented.

5. Discussion and Limitations

Why this is not just "weak specs are weak"

The claim is not that incomplete specifications exist. The claim is that vericoding pipelines can turn incomplete specifications into trusted selection targets.

A weak informal requirement is visibly weak. A weak formal target can look strong because it is associated with formal verification. This creates a specific evaluation risk: a pipeline may report that an implementation verifies, while the more important fact is that the target did not encode the intended behavior.

The missing question is therefore not only whether the implementation satisfies the spec. It is also whether the spec preserves the intent it is supposed to formalize.

What the results do and do not show

The results show that verified-but-wrong failures can be realized in public benchmark artifacts and that the mechanism can be isolated in a controlled benchmark. The external replication suite reduces cherry-picking risk by adjudicating all high-confidence scanner candidates under a frozen labeling protocol.

The results do not estimate prevalence. The external replication suite is scanner-ranked and enriched for likely gaps. It is not a random sample of all benchmark tasks. Only DA0003, DA0208, and DA0293 are direct-Dafny demonstrations against reconstructed original benchmark targets. These cases cover simple underconstraint classes, so broader omission patterns remain supported primarily by Tier 2 adapted demonstrations and the controlled benchmark.

The repaired-target checks are local mechanism checks. They show that adding the omitted target constraint blocks the same bad candidate in the three Tier 1 cases. They are not full repaired benchmark specifications and do not imply that all external tasks have easy repairs.

The controlled benchmark is author-constructed. It isolates the mechanism and mitigation behavior under controlled conditions, but it is not an external prevalence estimate. The Policy Audit Gate is also not a complete solution. It depends on a maintained policy inventory, can over-block when wording is outdated, and does not discover unknown requirements.

Why this is not oracle leakage or only candidate ordering

The hidden oracle is used only after candidate selection to evaluate intended behavior. The pre-selection audit sees the public spec and policy inventory, but not the hidden oracle and not the selected candidate.

The failure is also not merely caused by selecting the first passing candidate. Under naive specs, 34 of 46 public-spec-passing candidates failed hidden intended behavior. This shows that the public target underconstrained the candidate pool, not just the first selected candidate.

Practical recommendation and dual-use considerations

Vericoding pipelines should add a target-validity check before implementation selection: start with natural-language or policy intent, translate it into a formal/public spec, audit whether the spec preserves intent-critical requirements, repair missing or unclear requirements, then generate and select candidate implementations.

The target-validity check should not be treated as a fully autonomous rejection mechanism. A safer deployment posture is human-in-the-loop review: ALLOW specs proceed, REVIEW specs are repaired by a human, and BLOCK specs require explicit policy-owner approval before candidate selection.

A tool that identifies missing specification constraints could be misused to craft under-specified tasks that evade weak selection pipelines. The defensive use is to audit specs before implementation selection, surface missing requirements, and route risky specs to review. If future audits identify live production issues, those issues should be disclosed privately to maintainers before publication.

6. Conclusion

Vericoding can verify that code satisfies a supplied specification. It cannot, by itself, guarantee that the supplied specification was the right target.

We introduced verified-but-wrong as a target-validity failure mode: a selected implementation passes the public formal target but violates intended behavior. Implementation validity asks whether the code satisfies the spec. Target validity asks whether the spec

preserves the intent.

In an external Dafny vericoding benchmark audit, we scanned 1,448 useful-description tasks and identified 128 candidate target-validity gaps. We then adjudicated all 41 high-confidence candidates under a scanner-ranked external replication protocol. This yielded 11 externally sourced demonstrations: 3 Tier 1 direct-Dafny cases and 8 Tier 2 adapted demonstrations. In the Tier 1 cases DA0003, DA0208, and DA0293, trivially wrong implementations verify against reconstructed original benchmark targets while failing intended-behavior examples. Repaired-target checks block the same bad candidates in all three cases.

In a controlled 12-task benchmark, naive specs produced verified-but-wrong selections in 12/12 tasks, critic specs in 5/12, and oracle/full specs in 0/12. Candidate-set analysis showed that the issue is not merely fixed ordering: under naive specs, 34/46 public-spec-passing candidates failed hidden intended behavior. A deterministic target-audit baseline caught 17/17 dangerous specs with 0 dangerous allows and a 78.95% safe allow rate.

Before using a formal spec as a vericoding selection target, we should audit whether the spec is the right target.

If vericoding pipelines inherit the blind spots of their specifications, then target-validity auditing should happen before implementation selection.

Code, Data, and Demo

Code repository: <https://github.com/Philipn106/Verified-But-Wrong>

Demo source: <https://github.com/Philipn106/Verified-But-Wrong/tree/main/demo>

Live demo: <https://verified-but-wrong-demo.vercel.app/>

The demo replays repository reproduction logs and does not execute Dafny in the browser.

External audit dataset: Beneficial-AI-Foundation/vericoding-benchmark at commit 349ee510deeb6e15e1eb0d98174914e7093252fd, using `jsonl/dafny_tasks.jsonl`.

Primary reproduction commands: `python run_all_repro.py` and `python run_external_replication.py`.

Direct-Dafny verification logs, repaired-target checks, external replication summaries, and paper assets are included under `external_vericoding_cases/candidate_DA0003/`, `external_vericoding_cases/candidate_DA0208/`, `external_vericoding_cases/candidate_DA0293/`, `results/external_replication/`, and `paper_assets/`.

The direct-Dafny runs use Dafny 4.11.0. Main artifacts include the controlled benchmark, candidate-set analysis, target-audit baseline, repair baselines, robustness checks, external replication suite, direct-Dafny logs, repaired-target checks, CI/CD demo, and reproduction hashes.

References

1. 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. 2025. A Benchmark for Vericoding: Formally Verified Program Synthesis. *arXiv:2509.22908*.
2. Beneficial-AI-Foundation. Vericoding Benchmark repository. <https://github.com/Beneficial-AI-Foundation/vericoding-benchmark>
3. Loughridge, C. R., Sun, Q., Ahrenbach, S., Cassano, F., Sun, C., Sheng, Y., Misu, M. R. H., Amin, N., and Tegmark, M. 2024. DafnyBench: A Benchmark for Formal Software Verification. *arXiv:2406.08467*.
4. Dougherty, Q., and collaborators. 2025. A Benchmark for Formally Verified Code Generation / FVAPPS. *arXiv:2502.05714*.
5. Clarke, E., Grumberg, O., Jha, S., Lu, Y., and Veith, H. 2000. Counterexample-Guided Abstraction Refinement. *Computer Aided Verification*.
6. Wang, R. Counterexample-Guided Inductive Synthesis. Online primer.
7. Raghathan, M., Udupa, A., Kuraj, I., and Solar-Lezama, A. 2014. Synthesizing Finite-state Protocols from Scenarios and Requirements. *arXiv:1402.7150*.
8. Melcer, D., and collaborators. 2024. Approximately Aligned Decoding. *arXiv:2410.01103*.
9. Regehr, J. 2026. Zero-Degree-of-Freedom LLM Coding using Executable Oracles.
10. Dodds, M. 2025. Specifications Don't Exist. *Galois*.
11. Greenblatt, R., Shlegeris, B., Sachan, K., and Roger, F. 2024. AI Control: Improving Safety Despite Intentional Subversion. *arXiv:2312.06942*.
12. Elastic Kibana GitHub Issue #192346. 2024. [Serverless RBAC] Custom Role Fails to Override Admin Role for Logged-In User Even After Refresh.

Appendix A. Additional Ablations and LLM Usage

Additional gate ablations are reported in the repository. In summary, an allow-all gate catches 0/17 dangerous specs. Keyword-only and no-severity variants catch 17/17 dangerous specs but over-block safe specs with a 42.11% safe allow rate. Category-only and no-unclear-handling variants allow 10/17 dangerous specs. Generic-policy-only variants allow 14/17 dangerous specs. The full target-audit baseline catches 17/17 dangerous specs with 0 dangerous allows and a 78.95% safe allow rate.

LLM assistance was used for brainstorming, implementation support, debugging, and drafting portions of this report. All reported quantitative results were produced by the project's reproduction scripts and checked against generated logs and reports. The author is responsible for all claims, framing, and final submission content.