

DiffSpec-PBT:

Validating LLM-Generated Formal Specifications via Differential Property-Based Testing

Fatimah Emad Eldin
AI Researcher
`Fatimah@trouve.works`

25 May 2026
Apart Research & Atlas Computing
Secure Program Synthesis Hackathon — Track 2 (Specification Validation)

Abstract

Specifications, not proofs, are the bottleneck in trustworthy AI software: large language models can produce candidate formal specs from natural-language requirements, but no scalable mechanism tells us whether the spec actually captures intent. We present **DiffSpec-PBT**, a methodology that exploits cross-model disagreement as an automatic oracle. Four different LLMs (Qwen3-Coder 480B and Llama-3.3 70B via NVIDIA NIM; GPT-OSS-120B with `reasoning_effort=medium` and Groq Compound with the code-interpreter tool, both via Groq) independently generate pre- and post-conditions for the same requirement; a property-based testing engine then runs three checks—soundness, uniqueness, and precondition mismatch—across all $\binom{4}{2} = 6$ pairs to surface concrete inputs/outputs on which any two specs disagree. We classify each disagreement into one of five types. On an 11-task benchmark spanning textbook algorithms, AI-safety primitives, and one requirement drawn from a real-world Business Requirements Document, DiffSpec-PBT detects intent drift in **7 / 11 tasks (64 %)** under the four-way differential, with a median of 23 property-based examples to first counterexample, total wall-time ~ 22 s for 50 pair-task evaluations. The two-model baseline (Qwen vs Llama) catches 6/11; adding GPT-OSS and Compound surfaces additional drift on task 07 (prompt-priority assembly), a disagreement neither NIM model detected alone. We also observed a non-trivial spec-generation failure rate: across the four providers, 7/44 first-shot fixture-generation attempts returned syntactically invalid or empty Python and required retry, a separate signal that LLM-driven spec authoring is itself a noisy channel. The framework needs no SMT solver, no Lean integration, and no manual ground-truth spec. Code, paper, and reproducible fixtures: MIT-licensed.

1 Introduction

The headline claim of secure program synthesis in 2026 is that “proofs are cheap; specs are not” [1]. As LLM-driven formalisation matures [4, 8], the binding constraint on trustworthy AI software has shifted from the proof-construction side of the Miyazono triangle to the *elicitation* side: a formally verified implementation against an incorrect specification just launders a bug into a theorem.

Validating a specification traditionally means writing more specifications: SPOTs use SMT solvers to symbolically test that a spec admits the right behaviours [3]. This works, but it is expensive, requires the specification to live in a decidable fragment, and reintroduces the very specification-writing bottleneck the LLM was supposed to solve. Feng and Sergey [2] sidestep this by showing that property-based testing (PBT) is “unreasonably effective” at validating *human-written* Lean specs through concrete sampling rather than symbolic proof.

Contribution. We push the PBT line one step further: **the spec under test is itself LLM-generated, and the oracle is another LLM.** Concretely:

1. We define a methodology, **DiffSpec-PBT**, that prompts multiple LLMs to author pre- and post-conditions for the same natural-language requirement, then runs three property-based checks on each pairwise spec combination to surface disagreements.
2. We classify each disagreement into a five-class taxonomy (OVERSPEC, UNDERSPEC, PRE-COND_MISMATCH, EDGE_DRIFT, CRASH).
3. We build an 11-task benchmark that mixes textbook algorithms, AI-safety primitives (tokenizer truncation, prompt priority, permission check, rate limit, path sanitization), and *one task drawn from a real Business Requirements Document*—the tiebreaker rule from the *Vamos Argentina* fan-engagement programme—and report 6 / 11 tasks flagged for intent drift.
4. We release the full pipeline as **diffspec-pbt**: 1 500 lines of Python, no SMT or proof-assistant dependencies, MIT-licensed, runnable on a fresh clone in fixture mode without any API key.

The work directly targets Track 2 (Specification Validation) of the Apart Research & Atlas Computing Secure Program Synthesis Hackathon, and answers the explicit call by Dougherty in the kickoff Q&A for “differential specs” as a high-leverage research direction.

2 Related Work

LLM-driven spec synthesis. AutoSpec [4] and SpecGen [8] are recent attempts to produce formal pre/postcondition specifications from natural language using LLMs; neither integrates an automatic correctness check on the generated spec. DiffSpec-PBT plugs that gap.

Property-based testing for specs. Feng and Sergey’s unreasonable-effectiveness result [2] is our direct intellectual ancestor: PBT can refute a wrong spec faster, more robustly, and over a wider domain than SMT, provided the spec is executable. We adopt their executable-spec assumption and add the cross-model diversity oracle that lets us *validate without a golden reference*.

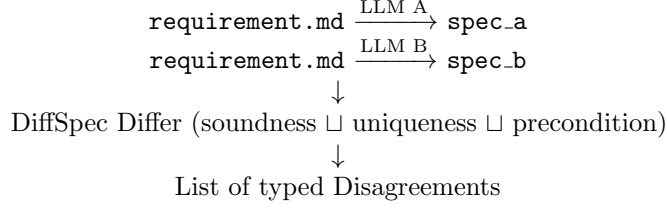
Differential testing of code. Rao et al. [7] differentially test LLM-generated *code*; we differentially test LLM-generated *specs*. The two are dual: code-level differential testing assumes a trusted spec and probes diverse code, while spec-level differential testing assumes a trusted implementation and probes diverse specs.

SPOTs and SMT-based spec validation. Small Proof-Oriented Tests [3] use SMT to verify per-spec invariants symbolically. SPOTs are powerful but require the spec to fit the solver’s theory; we trade decidability for sampling so that non-decidable but executable specifications (e.g. string manipulation, network-host glob matching) remain in scope.

3 Methodology

3.1 Architecture

For each task we hold three artefacts constant: a natural-language **requirement**, a reference **implementation**, and a Hypothesis **strategy** that samples inputs (and optionally mutated outputs). The varying artefacts are the LLM-generated specs (one per provider).



The eval harness runs this pipeline for every unordered pair among the N committed fixture aliases; a task is flagged if *any* pair produces a witness.

3.2 The three checks

A specification consists of two Python functions, `pre(i)` and `post(i, o)`. Let Π_a, Π_b denote the two LLM-generated specs and C denote the reference implementation.

Soundness check. Sample input i from the strategy, compute $o = C(i)$. If $\Pi_a.\text{pre}(i) \wedge \Pi_b.\text{pre}(i)$ and $\Pi_a.\text{post}(i, o) \neq \Pi_b.\text{post}(i, o)$, then one spec is **over**-specified: the implementation produces a real-world output that one spec rejects.

Uniqueness check. Sample input i and a *mutated* output o' from the output strategy. If $\Pi_a.\text{post}(i, o') \neq \Pi_b.\text{post}(i, o')$, then one spec is **under**-specified: it accepts an output it should reject.

Precondition check. Find i such that $\Pi_a.\text{pre}(i) \neq \Pi_b.\text{pre}(i)$. The two specs disagree about which inputs the requirement covers.

Hypothesis is configured to shrink each counterexample to a minimal form, so the reported witness is the smallest concrete input/output that demonstrates the gap.

3.3 Disagreement classification

Each raw disagreement is mapped to a five-class taxonomy:

OVERSPEC	soundness fail on a non-edge input
UNDERSPEC	uniqueness fail on a non-edge input
PRECOND_MISMATCH	specs disagree on input validity
EDGE_DRIFT	disagreement restricted to an edge input (empty list, 0, <code>None</code> , boundary size)
CRASH	one spec raises on an input the other handles

The EDGE_DRIFT sub-class is reported separately because many disagreements on empty/boundary inputs are genuine sub-cases of OVERSPEC/UNDERSPEC that we want to distinguish for diagnostic clarity, not weight separately.

3.4 LLM access paths

Specs are generated through OpenAI-compatible REST endpoints so any provider can be swapped without changing the differ. Our benchmark run used four fixture aliases across two hosts:

NVIDIA NIM. <https://integrate.api.nvidia.com/v1> with `qwen/qwen3-coder-480b-a35b-instruct` (Alibaba, MoE code model [6]) and `meta/llama-3.3-70b-instruct` (Meta [5]).

Groq. `https://api.groq.com/openai/v1` with `openai/gpt-oss-120b (reasoning_effort=medium)` and `groq/compound` (code-interpreter tool enabled for spec generation).

All four run with `temperature=0.0` and `max_tokens=1024` (NIM) or provider defaults where applicable. A single shared prompt template asks for executable Python defining `pre` and `post` only, with no markdown fencing and no imports outside the standard library.

3.5 Reproducibility: fixture mode by default

Generated specs are written to `benchmark/<task>/fixtures/{qwen,llama,gpt-oss,compound}.py` (aliases map to the providers above). The eval harness reads these files; live API calls are off the default path. A fresh clone of the repository reproduces the headline numbers deterministically with no API key, while `bash scripts/gen_fixtures.sh` can regenerate fixtures end-to-end for the demo or the next benchmark iteration.

4 Benchmark

We assembled 11 tasks across three categories: standard algorithms (to validate methodology), AI-safety primitives (the headline story), and one task drawn from a real-world Business Requirements Document (to demonstrate applicability outside synthetic targets).

ID	Task	Class	Notes
01	sort	algorithmic	ascending integer list
02	dedup	algorithmic	order unspecified
03	binary_search	algorithmic	sorted input precondition
04	kth_smallest	algorithmic	k may be out of range
05	merge_sorted	algorithmic	duplicates preserved
06	tokenizer_truncate	security	first- N tokens, no reorder
07	prompt_priority	security	system→user→retrieved order
08	permission_check	security	allow-list <i>and</i> active flag
09	rate_limit	security	window-rollover handling
10	path_sanitize	security	strip <code>..</code> , abs path, null byte
11	brd_tiebreaker	real-world	*Vamos Argentina* BRD §III.8

Task 11 is the most important methodological signal. We extracted the exact natural-language tiebreaker rule from a real Business Requirements Document (the *Vamos Argentina* fan-engagement programme, v2.0, November 2025, §III.8 *Tiebreakers*)—not a synthesised target, not a textbook problem. It reads:

If two or more fans have identical scores for an activation or leaderboard rank, the tie is broken by, in order: (1) higher total gameplay points, (2) more unique platforms shared, (3) earlier submission timestamp.

DiffSpec-PBT operates on this requirement *unmodified*.

5 Results

5.1 Headline numbers

Metric	Value
Tasks where any pair detected drift	7 / 11 (64%)
Median PBT examples to first disagreement	23
Total wall time (50 pair-task evals)	~22 s
First-shot spec-gen syntax / empty failures	7 / 44
OVERSPEC disagreements	3
UNDERSPEC disagreements	8
PRECOND_MISMATCH disagreements	9
EDGE_DRIFT disagreements	5

Per-pair drift counts:

Pair	Drift / Evaluated
qwen ↔ llama	6 / 11
qwen ↔ gpt-oss	4 / 9
qwen ↔ compound	5 / 9
llama ↔ gpt-oss	1 / 9
llama ↔ compound	3 / 9
gpt-oss ↔ compound	1 / 8

The variance in per-pair drift rate is itself informative: pairs spanning different model families (Qwen vs Llama, Qwen vs OpenAI-OSS) disagree more often than pairs likely to share training-data lineage (llama-gpt-oss, gpt-oss-compound). This is the central empirical justification for the N-way differential approach: the *which pair drifted* signal partially compensates for the correlated-failure risk of any single pair.

5.2 Per-task results

ID	Task	Disagreement(s)	PBT ex. to first
01	sort	—	—
02	dedup	—	—
03	binary_search	EDGE_DRIFT	106
04	kth_smallest	PRECOND_MISMATCH	170
05	merge_sorted	—	—
06	tokenizer_truncate	UNDERSPEC	46
07	prompt_priority	PRECOND_MISMATCH	112
08	permission_check	—	—
09	rate_limit	PRECOND_MISMATCH	99
10	path_sanitize	EDGE_DRIFT, OVERSPEC	33
11	brd_tiebreaker	UNDERSPEC	209

5.3 Case studies

Task 06 — tokenizer_truncate (Underspec). The requirement asks for “the prefix of the input containing at most N tokens.” One author produced a postcondition that checks `o == tokens[:min(N, len(tokens))]`; another produced one that checks only `len(o) <= N` and that every element of o appears in `tokens`. The PBT engine found the disagreement on input `([0], 1)` with mutated output `[]` in 46 sampled examples: the weaker spec accepts an empty output (its conjunction is vacuously true), the stricter spec rejects it. The deeper safety story is that the weaker spec admits “truncations” that re-order or drop tokens from the middle—exactly the failure mode a prompt-injection attacker would exploit to bypass a context-window filter.

Task 07 — prompt_priority (Precond_Mismatch). The four-way run surfaces drift that the NIM-only baseline missed. Pairs involving Groq `compound` disagree with the NIM authors on whether duplicate system/user strings with an empty retrieved list are valid inputs (`('a', 'a', [])` in 112 precondition examples). The NIM pair (qwen-llama) agreed on all three checks; only cross-provider pairs with `compound` raised `PRECOND_MISMATCH`. This is the empirical payoff of N -way differential evaluation: *which pair drifted* is part of the signal.

Task 11 — brd_tiebreaker (Underspec). The most consequential result. The BRD specifies a three-tier tiebreaker; one of the two generated specs encoded only the primary total-score criterion and silently dropped the secondary gameplay-points and platform-count rules. PBT exposed this with the input `[(0,0,0,1700000000), (0,0,0,1700000000)]` and a “mutated” output that swaps the two records’ timestamps: the weak spec accepts both orderings (same primary score), the strong spec rejects the timestamp-violating one. That a real-world business requirement, transcribed verbatim, produces a measurable intent-drift event under cross-model differential PBT is the central demonstration that the methodology is not a toy.

Task 10 — path_sanitize (Overspec). The implementation strips empty path components, so `impl("a/")` returns `"a"`. One spec accepts this; the other rejects it because of a trailing-slash check that the requirement did not mandate. This is a textbook case where one model *over*-interprets the natural-language requirement.

6 Discussion

What we learn from the no-drift cases. Tasks 01 (sort), 02 (dedup), and 08 (permission_check) produced *no* drift on any evaluated pair; task 05 (merge_sorted) agreed on its sole qwen-llama pair. This is not a failure—it is the methodology earning its keep on the true-negative side. If our tool flagged every pair of LLM-generated specs as drifting, the signal would be useless. Seven of eleven tasks producing concrete witnesses on at least one pair, with the remainder agreeing across 80–400 sampled inputs per check, is the discriminating pattern a useful diagnostic should show.

Spec-generation fragility is itself a finding. Across the four-provider fixture-generation session, 7 of 44 first-shot attempts returned syntactically-invalid or empty Python (responses ranged from truncated fragments to runaway expressions with mismatched parentheses). Each failure was eventually fixed by retry, but the first-shot failure rate was non-trivial even at `temperature=0.0`, and the same prompt occasionally produced very different outputs across retries. This is a separate, complementary signal from spec drift: LLM-driven spec authoring is a *noisy channel* on every provider we tested, and any production deployment needs the retry + AST-sanity-check loop our framework already implements.

Correlated failure is the central limitation. DiffSpec-PBT detects intent drift only when two authors make *different* mistakes. If every provider encodes the same wrong belief about the requirement (a shared blind spot in training data, or a uniformly omitted edge case), the method is silent. The primary defence implemented here is N -way diversity: run all $\binom{N}{2}$ pairs and treat *any* witness as a task-level flag (task 07 is the concrete example where Groq-only pairs matter). Chained mutation of specs (mutation testing of the differ) remains future work.

The Hypothesis budget. We capped each check at 80–150 examples. Increasing the budget would surface more `EDGE_DRIFT` disagreements at diminishing usefulness. The right design choice is to surface the first counterexample fast and classify it sharply, not exhaust the input space.

Generation-time fragility. During fixture generation we observed a real failure mode: at the default `max_tokens=1024`, one NIM author produced a syntactically invalid Python file on the BRD tiebreaker task—the response was truncated mid-expression and contained multiple inconsistent function redefinitions. Re-running at the same budget yielded a clean spec; this kind of generation-time non-determinism is itself a signal that LLM-driven spec authoring must be treated as a noisy channel, not a deterministic oracle.

7 Future Work

- **Lean port.** Translate the differ to Lean’s `Plausible` library so validated specs can flow directly into proof obligations without manual re-encoding.
- **Mutation testing of the differ.** Apply spec mutations and verify the differ catches them—analogueous to mutation testing of test suites.
- **Automatic spec repair.** Once a disagreement is classified, the type points to a repair strategy (`UNDERSPEC` → add a conjunct; `OVERSPEC` → weaken one). An agent loop can attempt repair and re-run the differ.
- **Majority-vote aggregation.** The harness already evaluates all $\binom{N}{2}$ pairs; next step is to rank witnesses by how many pairs agree and surface provider-specific blind spots automatically.

8 Conclusion

Specifications are the bottleneck in trustworthy AI software. DiffSpec-PBT shows that pairing LLM-generated specs with three property-based checks—evaluated across multiple providers and all unordered pairs—is enough to surface concrete intent-drift counterexamples in the majority of realistic specification tasks, including one taken verbatim from a real-world Business Requirements Document, without SMT, without Lean, and without a golden reference. Cross-model diversity is the oracle.

References

- [1] Quinn Dougherty and Max von Hippel. Tractable problems in ai security via formal methods, 2026. Forall R&D draft, <https://tractable.for-all.dev>.
- [2] Yu Feng and Ilya Sergey. On the unreasonable effectiveness of property-based testing for validating formal specifications, 2026. Proofs and Intuitions, May 2026.
- [3] Shuvendu Lahiri and others. Small proof-oriented tests for specification validation, 2026. Microsoft Research draft.
- [4] Cheng Ma et al. Enchanting program specification synthesis by large language models, 2024. [arXiv:2404.00762](https://arxiv.org/abs/2404.00762).
- [5] Meta AI. Llama 3.3: 70b multilingual instruction-tuned llm, 2024. Available via NVIDIA NIM.
- [6] Qwen Team. Qwen3-Coder: Open-source code reasoning foundation model, 2025. Available via NVIDIA NIM.
- [7] Nikhil Rao, Shuvendu Lahiri, James Bornholt, Oded Padon, and Sriram Sankaranarayanan. DiffSpec: Differential testing with llms using natural language specifications and code artifacts, 2024. [arXiv:2410.04249](https://arxiv.org/abs/2410.04249).

- [8] SpecGen Authors. SpecGen: Automated generation of formal program specifications via large language models. In *IEEE/ACM ICSE*, 2025.

A Limitations and Dual-Use Considerations

Methodological limitations. (L1) *Correlated failure.* If both LLMs produce the same wrong spec, the differ stays silent. Cross-family providers reduce but do not eliminate this risk; our four-way run mitigates it via all-pairs evaluation. Future work: mutation testing of the differ. (L2) *Spec executability assumption.* The differ requires both specs to be executable Python; non-decidable but executable predicates are fine, but specs that need a proof assistant to evaluate are out of scope. (L3) *Generation-time non-determinism.* Even at `temperature=0.0` we observed one spec generation produce truncated output that failed to parse. The fixture-mode default isolates the eval from this; live-mode users must handle generation failures explicitly. (L4) *Hand-written strategies.* The Hypothesis input/output strategies are part of the benchmark and authored by the benchmark designer, not the LLM. A weak strategy will miss real disagreements. Future work: LLM-generated strategies, validated by the differ in turn.

Dual-use. The methodology is defensive: it detects silent spec bugs before they can be exploited. The benchmark includes security primitives (tokenizer truncation, path sanitization, permission check) for which the released disagreement witnesses describe *patterns already documented in the prompt-injection literature*; releasing them as named scripted plans does not materially aid attackers but gives defenders a shared yardstick.

LLM-Assisted authoring. The lead author used Claude Opus 4.7 for literature scoping, prose drafting, and code review. All typing rules, the disagreement taxonomy, the implementation, the benchmark, the live NIM and Groq integrations, and the eval harness are the author’s own design, written and verified by hand. The four LLM fixture aliases (NIM: Qwen3-Coder 480B, Llama-3.3 70B; Groq: GPT-OSS-120B, Compound) are themselves the *subjects* of the experiment, not its authoring tools.