Spec*Bench: Benchmarking Specification Generation with Automated Verification

Baris Bayazit[1], Xujie Si[1]

[1]Department of Computer Science, University of Toronto

Computer Science
UNIVERSITY OF TORONTO

The Problem

Spec checking lacks guarantees (soundness, completeness, independence).

- Property-based testing (PBT) compares specs by examples, providing no correctness guarantees [4]:

PBT checks only $D' \subset D$: $S_{\text{gen}} = S_{\text{ref}}$ on D'
Equivalence requires: $\forall x \in D,\ S_{\text{gen}}(x) = S_{\text{ref}}(x)$

Samples are *evidence*, not a *guarantee*.

- LLM-assisted ITP proofs (e.g., Lean, Rocq) offload substantial effort to LLMs, resulting in 0% accuracy on state-of-the-art models [2]:

```
-- Goal: forall x, Sgen x <-> Sref x
theorem equiv : forall x, Sgen x <-> Sref x := by
  -- Generated equivalence proof - LLMs achieve 0% accuracy!
  sorry  -- proof not found => no guarantee
```

- **Conclusion:** Sample agreement $\not\Rightarrow$ equivalence; we need $\forall x.\, S_{\text{gen}}(x) \equiv S_{\text{ref}}(x)$ or a counterexample. LLM + ITP checking is not feasible.

Contributions

- Spec*Bench introduces **the first benchmark and protocol for spec generation with verifiable and automated checking**, considering only specifications whose proofs can be offloaded to an automated verifier.

Method	Sound?	LLM-independent Verification?	Successful in Practice?
LLM as a Judge	×	×	?
PBT	×	✓	✓
LLM + ITP	✓	×	×
Spec*Bench	✓	✓	✓

Table 1. Trade-offs among existing specification generation benchmarks.

Background

- **Specification.** A predicate that formalizes a natural-language description into precise mathematical conditions.
- **Correctness.** A generated specification S_{gen} is *correct* if the base specification S_{ref} can be formally proven equivalent to it using an automated (ATP) or interactive (ITP) theorem prover (e.g., SMT solvers such as Z3, or Coq).
- **Soundness.** ITPs and ATPs are sound, meaning if they accept a proof of equivalence between S_{gen} and S_{ref}, then that equivalence is indeed valid.

Method

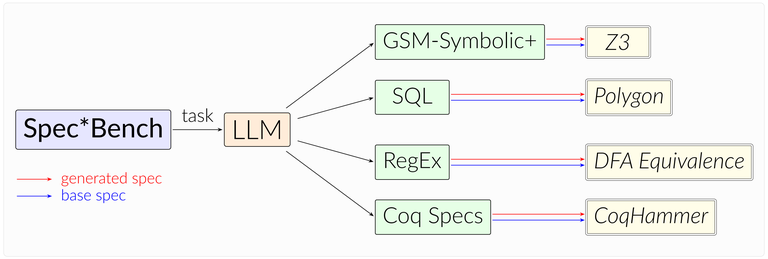
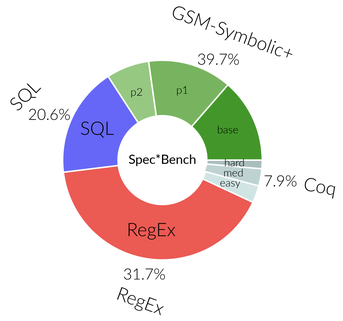


Figure 1. Overview of Spec*Bench methodology. Each natural language task is given to an LLM, which produces a candidate specification. The generated specification is compared against the base specification using domain-specific verifiers.



- **Input.** A natural-language task picked from one of the domains and a hidden base specification S_{ref}.
- **Generation.** The LLM proposes a candidate specification S_{gen}.
- **Verification.** Check $S_{\text{gen}} \equiv S_{\text{ref}}$ with the respective solver.
- **Verdict.** *Verified*, *Refuted* (counterexample), or *Unknown* (timeout/failure).

Figure 2. Spec*Bench task composition across four domains. The chart shows both main domain percentages and their sub-domain breakdown: GSM-Symbolic+ contains three variants (base, p1, p2), while Coq Specifications have three difficulty levels (easy, medium, hard).

Example

Domain: SQL **Solver:** Polygon (equivalence/refutation)

Task. Largest number that appears exactly once.

Base (S_{ref}).
```
SELECT MAX(num) AS num
FROM (SELECT num FROM MyNumbers GROUP BY num HAVING COUNT(*)=1) t;
```

LLM (S_{gen}).
```
SELECT MAX(num)
FROM MyNumbers GROUP BY num HAVING COUNT(*)=1;
```

Counterexample. `num`: {1, 2, 2, 5} **Outputs.** S_{ref}: 5, S_{gen}: 1, 5

Verdict. *Refuted.*

Evaluation

- **Protocol.** For each task×model, generate top-k candidates, and verify each with the corresponding solver under fixed time limits. With r *optional refinement rounds*, if *Refuted*, feed the counterexample to the LLM, and repeat.
- **Unknown policy.** If the limits are exceeded or a proof is not found/reconstructed, the verdict is *Unknown*. *Unknowns* are reported separately and not as correct.
- **Reporting.** The Verdict, Time2Verdict, Resource Limits, and Counterexamples.
- **Overall Accuracy.** Accuracy is calculated by *total Success verdicts divided by total items* across domains.

Results

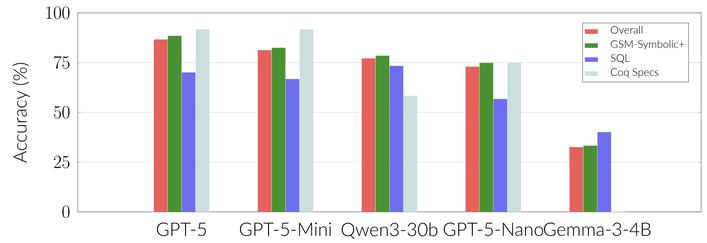


Figure 3. Spec*Bench performance across models and domains for a subset of the benchmark, with no refinement rounds and a 4 second timeout. RegEx was omitted for this experiment.

Analysis

- **SQL evaluation gap.** Most SQL failures are *still accepted* by LeetCode. Counterexamples are valid and fail LeetCode's own "Run Test." ⇒ **Tests are insufficient to evaluate specifications.**
- **Numbers vs. variables.** Accuracy drops from numeric variations are known [1], but GSM-Symbolic+ reveals a *further decline* when templating with variables.
- **Scaling for Coq.** Smaller models underperform at writing Coq specifications.

References

[1] Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models, 2024.

[2] Amitayush Thakur, Jasper Lee, George Tsoukalas, Meghana Sistla, Matthew Zhao, Stefan Zetzsche, Greg Durrett, Yisong Yue, and Swarat Chaudhuri. Clever: A curated benchmark for formally verified code generation, 2025.

[3] Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. Benchmarking multimodal regex synthesis with complex structures, 2020.

[4] Zhe Ye, Zhengxu Yan, Jingxuan He, Timothe Kasriel, Kaiyu Yang, and Dawn Song. Verina: Benchmarking verifiable code generation, 2025.

[5] Pinhan Zhao, Yuepeng Wang, and Xinyu Wang. Polygon: Symbolic reasoning for sql using conflict-driven under-approximation search. *Proceedings of the ACM on Programming Languages*, 9(PLDI):1315–1340, June 2025.