ICLR 2026

HardTestGen: A High-Quality RL Verifier Generation Pipeline for LLM Algorithmic Coding

Zhongmou He[1*] · Yee Man Choi[1*] · Kexun Zhang[1*†] · Jiabao Ji[2] · Junting Zhou[1] · Dejia Xu[3] · Ivan Bercovich[2] · Aidan Zhang[1] · Yixiao Zeng[1,4] · Lei Li[1]

[1]Carnegie Mellon University · [2]UC Santa Barbara · [3]UT Austin · [4]Meracode · [*]Equal contribution · [†]Project lead

Carnegie Mellon University · UC Santa Barbara · UT Austin · Meracode

1 Motivation

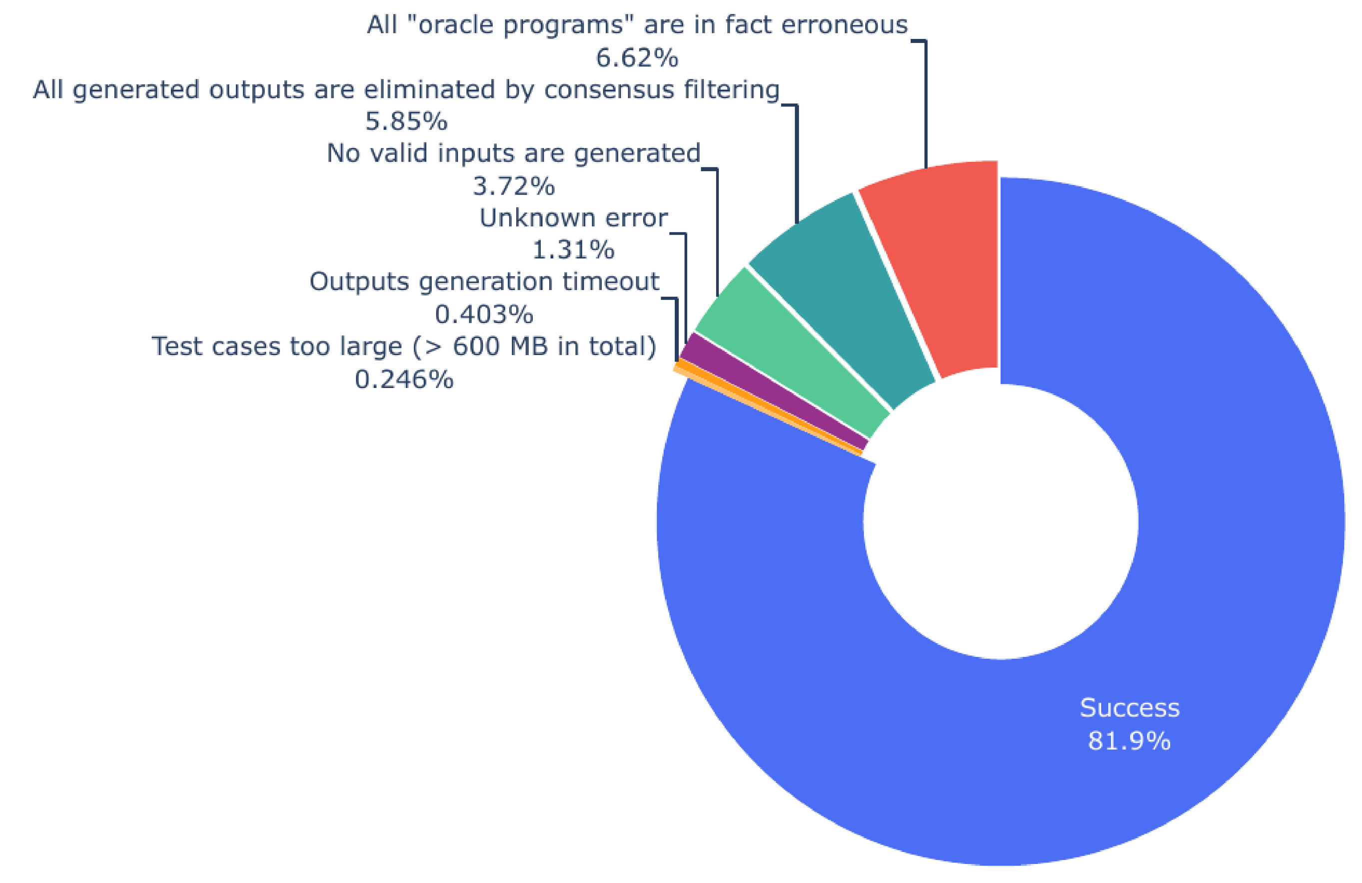
- RL for code LLMs needs **reliable verifiers** — tests *are* the reward.
- **APPS:** 60% of programs that pass its tests are **wrong**.
- **CodeContests:** 46% that pass are **too slow**.
- Human-written hard tests don't scale; we need auto-synthesis.

APPS — 60% wrong
CodeContests — 46% slow

Can we auto-synthesize tests *hard* enough to reward-shape modern code LLMs — at scale?

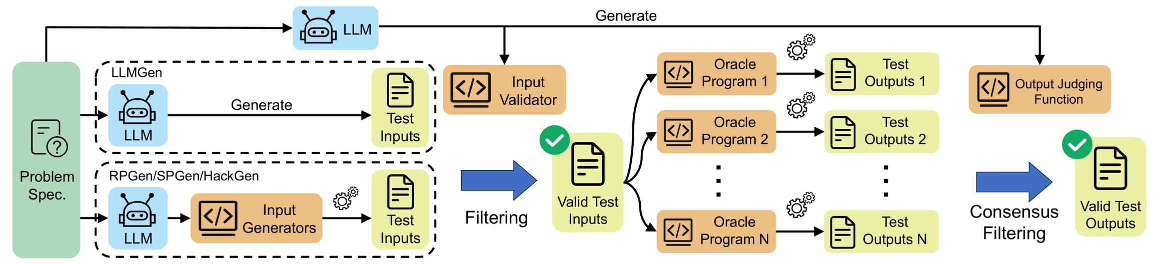
2 Key Insights

- **Programs, not prose.** LLMs can't write large valid inputs directly — but they can write **generator programs**.
- **Diverse hypotheses → diverse tests.** Four generator types probe four failure modes.
- **Oracles decide outputs.** 68% of contest problems ship with human-written oracles; we run them all and keep only **≥90% consensus** outputs.



81.9% of problems get valid tests end-to-end.

3 HardTestGen Pipeline



LLMGen — LLM *directly* writes small inputs mimicking samples.

RPGen — Random generator respecting types & ranges.

SPGen — Stratified — one generator per **output class**.

HackGen — Adversarial — force **WA / TLE** on flawed algos.

4 Example · AtCoder ABC139C

Problem

N squares with heights H_i. From any square, step right while next height ≤ current. **Find max moves.**

Wrong greedy: for each i, scan right greedily — O(N²) on strict chains.

✗	LLMGen	N ≤ 20 · greedy passes. **Bug hidden.**
✗	RPGen	Random H ∈ [1,10⁹]. **Not worst-case.**
✗	SPGen	Output-class split. Still fast inputs.
✓	HackGen	**Strict descending H · N=10⁵ → TLE!**

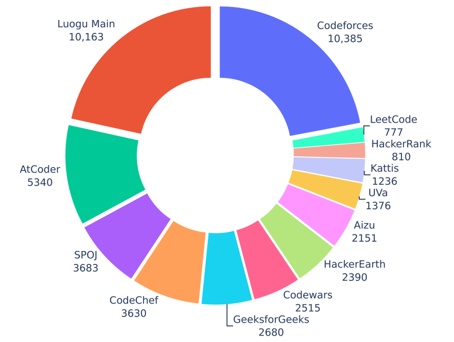
HACKGEN GENERATOR · LLM-SYNTHESIZED (PAPER APP. B)

```
def gen_hack_monotonic_decreasing():
    N = 100000                          # max allowed
    H = list(range(10**9, 10**9 - N, -1))  # strict chain
    print(N); print(*H)                 # greedy → O(N²) TLE
```

Wrong greedy fails the official tests only after HackGen supplies the worst-case input.

5 Dataset Coverage

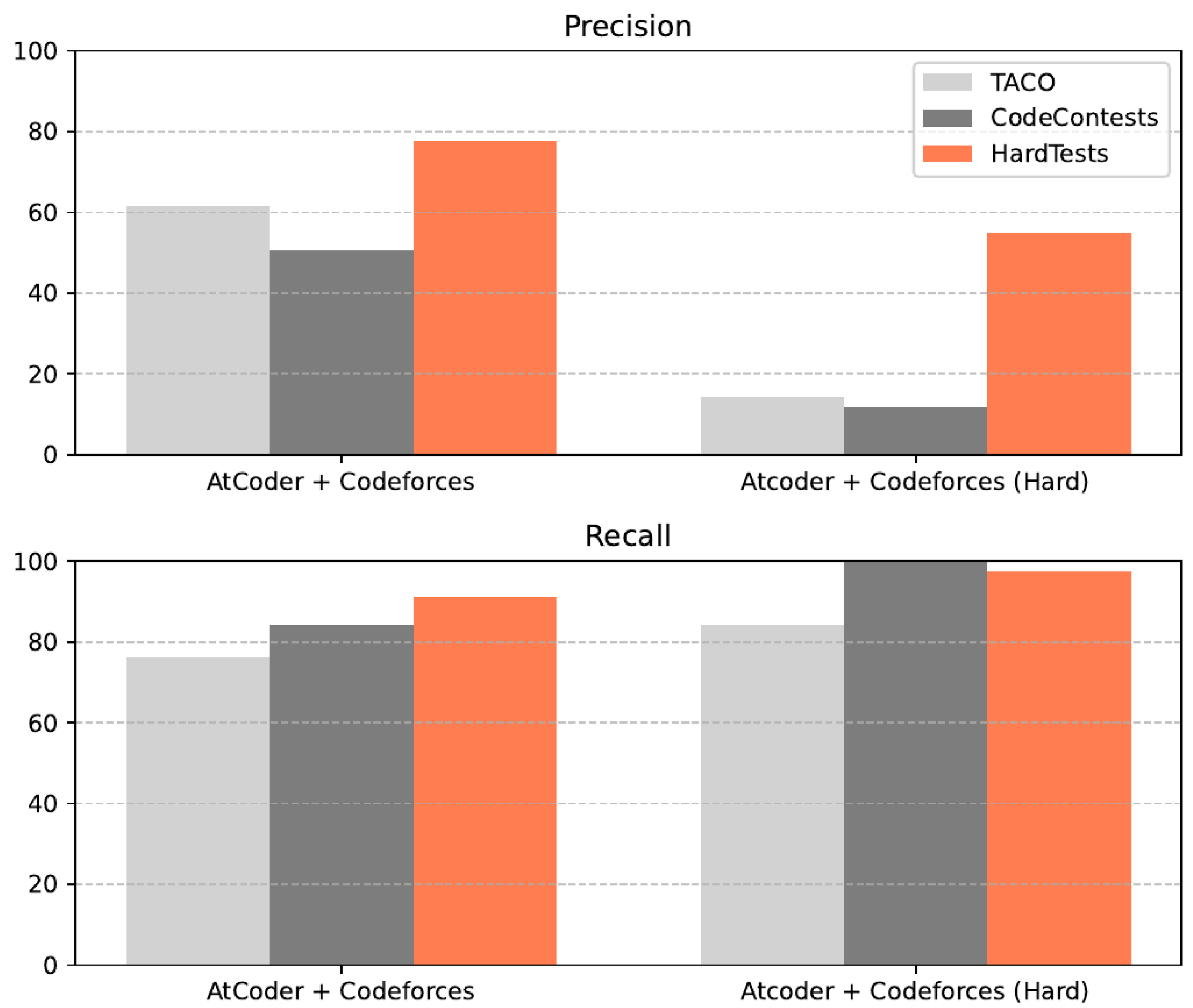
- **26.6k problems** across **13 OJ platforms.**
- Source languages span **English, Chinese, Japanese** — all translated to English.



Problems per online judge — long-tail by design.

6 Headline Result

+11.2	+11.0	3×
pts precision avg. over baselines	pts recall avg. over baselines	precision on Diff 4+ vs. TACO

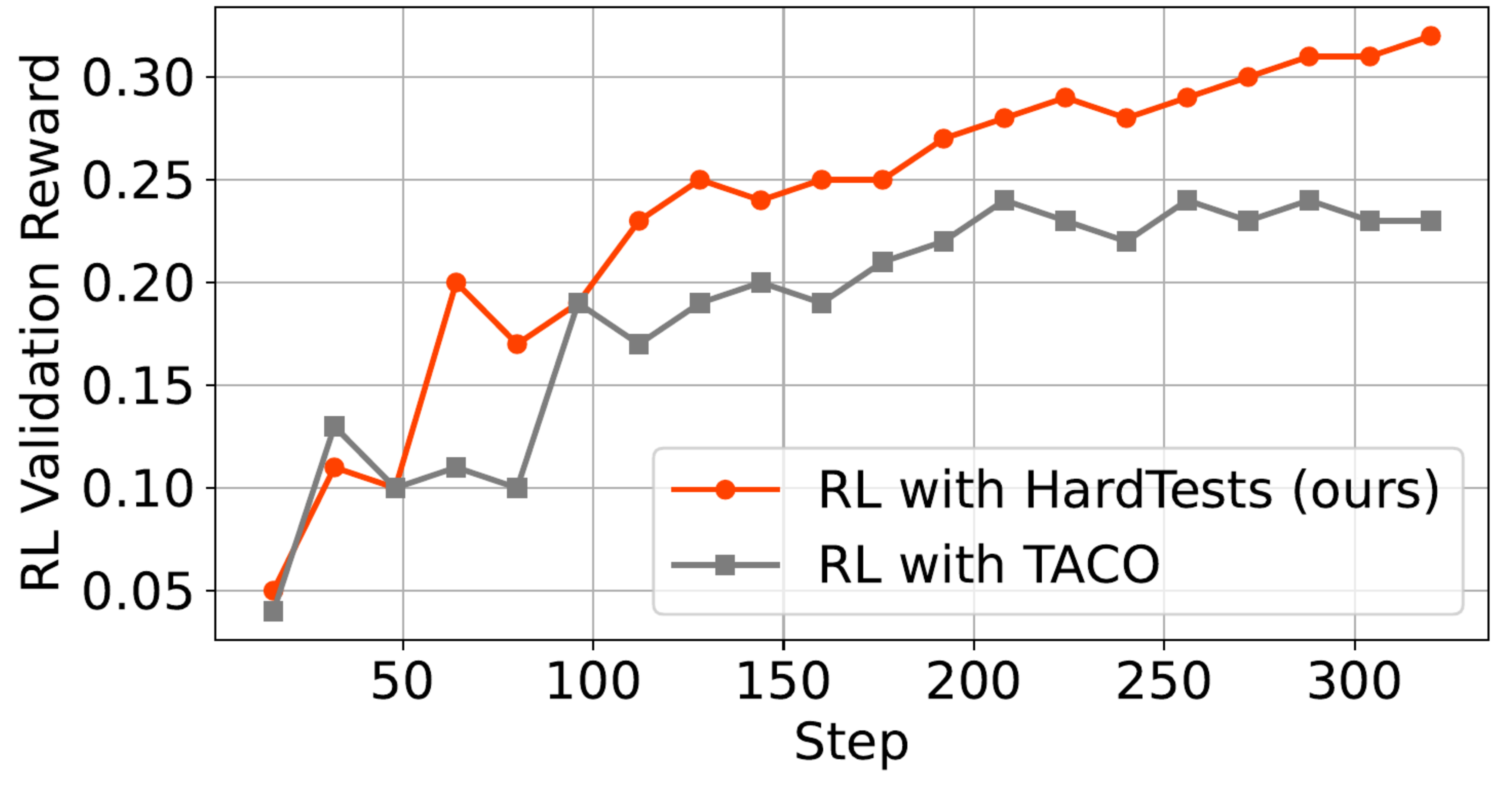


- Gap **grows with difficulty** — up to +40 pts on Diff 4+.
- Recall stays high: HardTests does not over-reject correct code.

7 Precision by Difficulty

Method · Qwen2.5-Coder-7B	Diff 3	Diff 4+	Avg.
TACO	53.8	17.8	61.0
CodeContests	39.3	10.8	51.6
HT − LLMGen only	29.0	8.0	42.4
HT − L+R/S	62.8	47.7	64.7
HardTests (full)	**70.8**	**55.9**	**79.2**

8 Better Tests → Better RL



- RL + HardTests: **+8.6 pt pass@10** on LiveCodeBench.
- RL + TACO: **hurts pass@1** — bad rewards are worse than no RL.
- Same story for **rejection-sampling (StaR)**: pass@10 = 60.0.
- Harder tests = **tighter reward signal** for policy training.

Qwen3-4B · LiveCodeBench	pass@1	pass@10
Base	38.5	56.2
RL + TACO	37.0	57.1
RL + HardTests	**39.4**	**64.8**

★ Takeaways & Links

- Hard tests come from **generator programs**, not LLM-written prose.
- Adversarial **HackGen** is the missing piece — alone it closes the worst-case gap.
- Only **$0.23 API cost per problem** — scalable to new OJs.
- Better verifiers ⇒ better RL: **not just more RL.**

Project · Code · Dataset · Meracode