Towards LLM-Driven Open-ended Research

Discovery, Evaluation, and Data Scaling

Qiuyang Mang
University of California, Berkeley

qmang@berkeley.edu

SWE-bench Verified ≈ **80%**

AIME ≈ **100%**

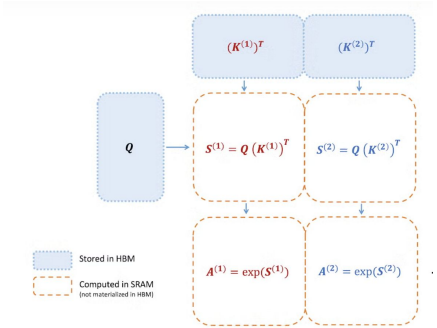
Humanity Last Exam ≈ 60%

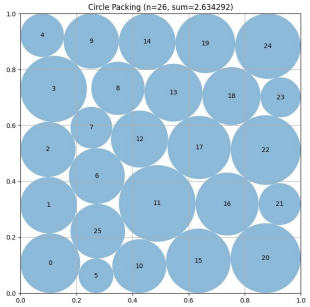
LLMs now **saturate** real exams and exam-style benchmarks.

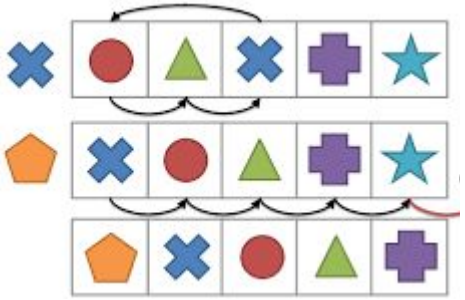
What's next?

Open-ended Problems

Beyond Passing Exams: Open-ended Problems







LLM as a performance optimizer

How much speedup can we achieve with a GPU kernel?

LLM as an algorithm designer

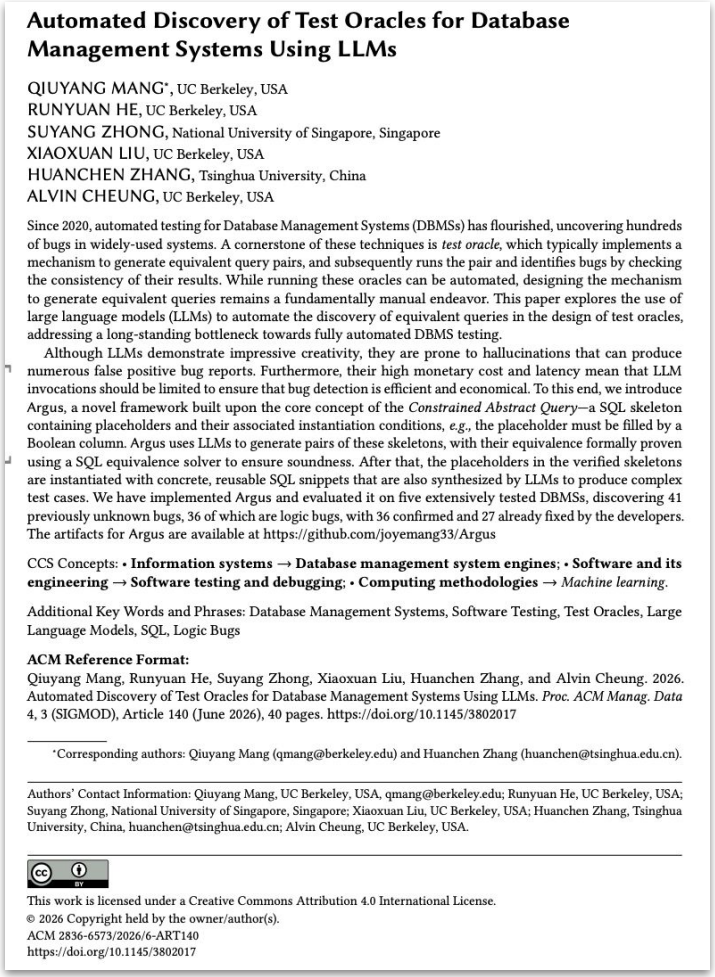
How many circles can we pack into a fixed region?

LLM as a system researcher

How much cache miss cost can we reduce with a replacement policy?

No optimal solutions, only better frontiers.

Topic #1: If LLMs can tackle long-standing research problems by modeling them as open-ended coding tasks?



Automated Discovery of Test Oracles for Database Management Systems Using LLMs

QIUYANG MANG*, UC Berkeley, USA
RUNYUAN HE, UC Berkeley, USA
SUYANG ZHONG, National University of Singapore, Singapore
XIAOXUAN LIU, UC Berkeley, USA
HUANCHEN ZHANG, Tsinghua University, China
ALVIN CHEUNG, UC Berkeley, USA

Since 2020, automated testing for Database Management Systems (DBMSs) has flourished, uncovering hundreds of bugs in widely-used systems. A cornerstone of these techniques is *test oracle*, which typically implements a mechanism to generate equivalent query pairs, and subsequently runs the pair and identifies bugs by checking the consistency of their results. While running these oracles can be automated, designing the mechanism to generate equivalent queries remains a fundamentally manual endeavor. This paper explores the use of large language models (LLMs) to automate the discovery of equivalent queries in the design of test oracles, addressing a long-standing bottleneck towards fully automated DBMS testing.

Although LLMs demonstrate impressive creativity, they are prone to hallucinations that can produce numerous false positive bug reports. Furthermore, their high monetary cost and latency mean that LLM invocations should be limited to ensure that bug detection is efficient and economical. To this end, we introduce Argus, a novel framework built upon the core concept of the *Constrained Abstract Query*—a SQL skeleton containing placeholders and their associated instantiation conditions, *e.g.*, the placeholder must be filled by a Boolean column. Argus uses LLMs to generate pairs of these skeletons, with their equivalence formally proven using a SQL equivalence solver to ensure soundness. After that, the placeholders in the verified skeletons are instantiated with concrete, reusable SQL snippets that are also synthesized by LLMs to produce complex test cases. We have implemented Argus and evaluated it on five extensively tested DBMSs, discovering 41 previously unknown bugs, 36 of which are logic bugs, with 36 confirmed and 27 already fixed by the developers. The artifacts for Argus are available at https://github.com/joyemang33/Argus

CCS Concepts: • **Information systems** → **Database management system engines**; • **Software and its engineering** → **Software testing and debugging**; • **Computing methodologies** → *Machine learning*.

Additional Key Words and Phrases: Database Management Systems, Software Testing, Test Oracles, Large Language Models, SQL, Logic Bugs

ACM Reference Format:
Qiuyang Mang, Runyuan He, Suyang Zhong, Xiaoxuan Liu, Huanchen Zhang, and Alvin Cheung. 2026. Automated Discovery of Test Oracles for Database Management Systems Using LLMs. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 140 (June 2026), 40 pages. https://doi.org/10.1145/3802017

*Corresponding authors: Qiuyang Mang (qmang@berkeley.edu) and Huanchen Zhang (huanchen@tsinghua.edu.cn).

Authors' Contact Information: Qiuyang Mang, UC Berkeley, USA, qmang@berkeley.edu; Runyuan He, UC Berkeley, USA; Suyang Zhong, National University of Singapore, Singapore; Xiaoxuan Liu, UC Berkeley, USA; Huanchen Zhang, Tsinghua University, China, huanchen@tsinghua.edu.cn; Alvin Cheung, UC Berkeley, USA.

ACM 2836-6573/2026/6-ART140
https://doi.org/10.1145/3802017

Argus: Automated Discovering Test Oracles for Database Management Systems with LLMs

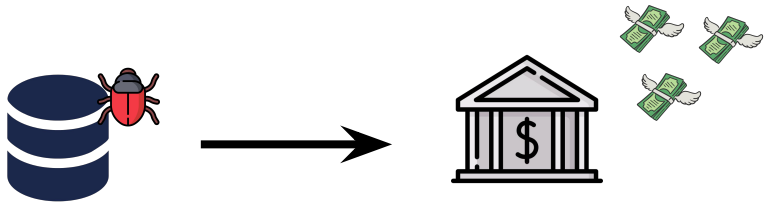
To be appear in SIGMOD 2026

***Qiuyang Mang**, Runyuan He, Suyang Zhong, Xiaoxuan Liu, Huanchen Zhang, and Alvin Cheung*

TLDR: $10 of LLM usage generates millions of reliable DBMS test cases and uncovers unknown logic bugs.

DBMS Can Return Incorrect Results

```
CREATE TABLE t(c INT);
INSERT INTO t VALUES (1);
SELECT sub.c FROM (
    SELECT
        json_array_length(json_array(3, 2, t.c))
    AS c FROM t
) AS sub
RIGHT JOIN t ON FALSE; -- {2}
```

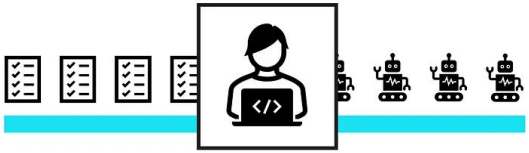


Buggy DBMS

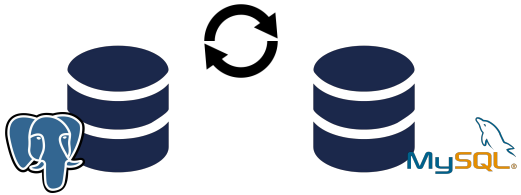
Downstream Application

Detect such **logic bugs** in DBMS

Existing DBMS Testing Methodologies



Manual Crafted Test Cases



Reference Engine

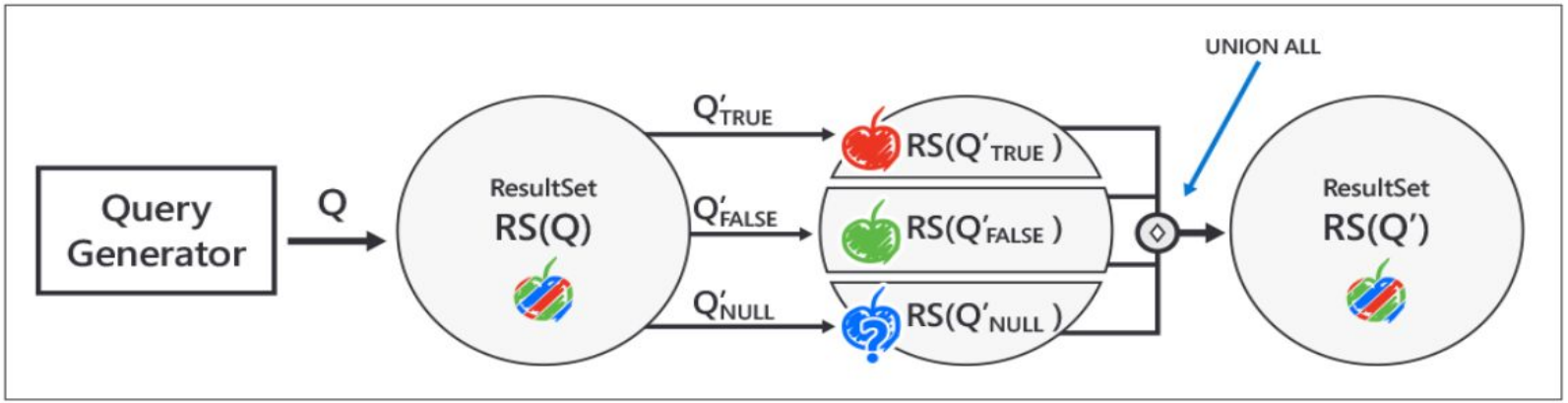
SQLANCER

Test Oracles

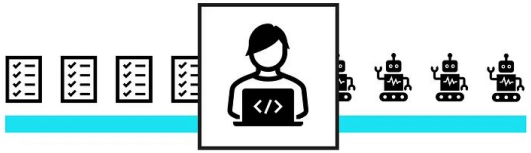
⇒ *Pairs of equivalent queries*

Before 2020 — 2020

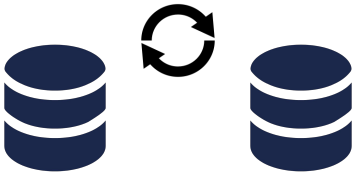
Example: Ternary Logic Partitioning



Existing DBMS Testing Solutions



Manual Crafted Test Cases



Reference Engine

SQLANCER

Test Oracles

⇒ *Pairs of equivalent queries*

Researcher

New Oracles

OSDI'24

ACM SIGMOD

VLDB

New Papers

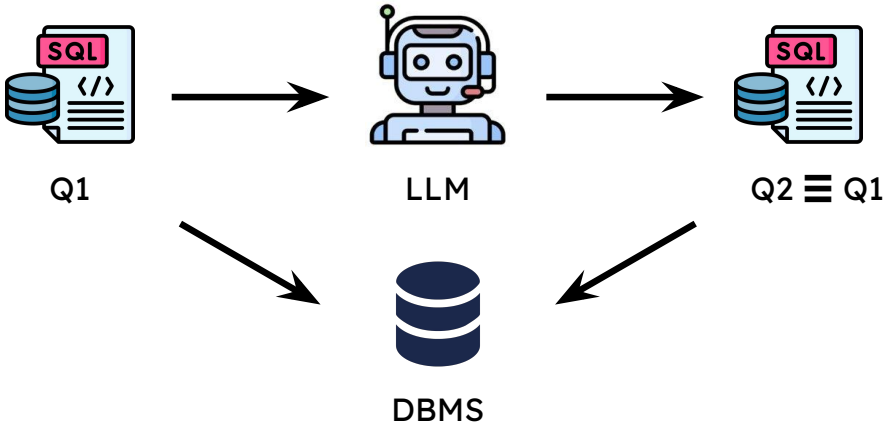
New Bugs

Before 2020 — 2020 — 2020 – 2025

Limited Test Oracles Hand-made by Human Experts

Using LLM to Break the Endless Cycle

Automated Oracles



Inconsistent Results ⇒ Bugs

C1: LLMs are slow and expensive

We needs about **100K** test cases to detect one unique bugs in mature DBMS :0

C2: Hallucination ⇒ False Alarms

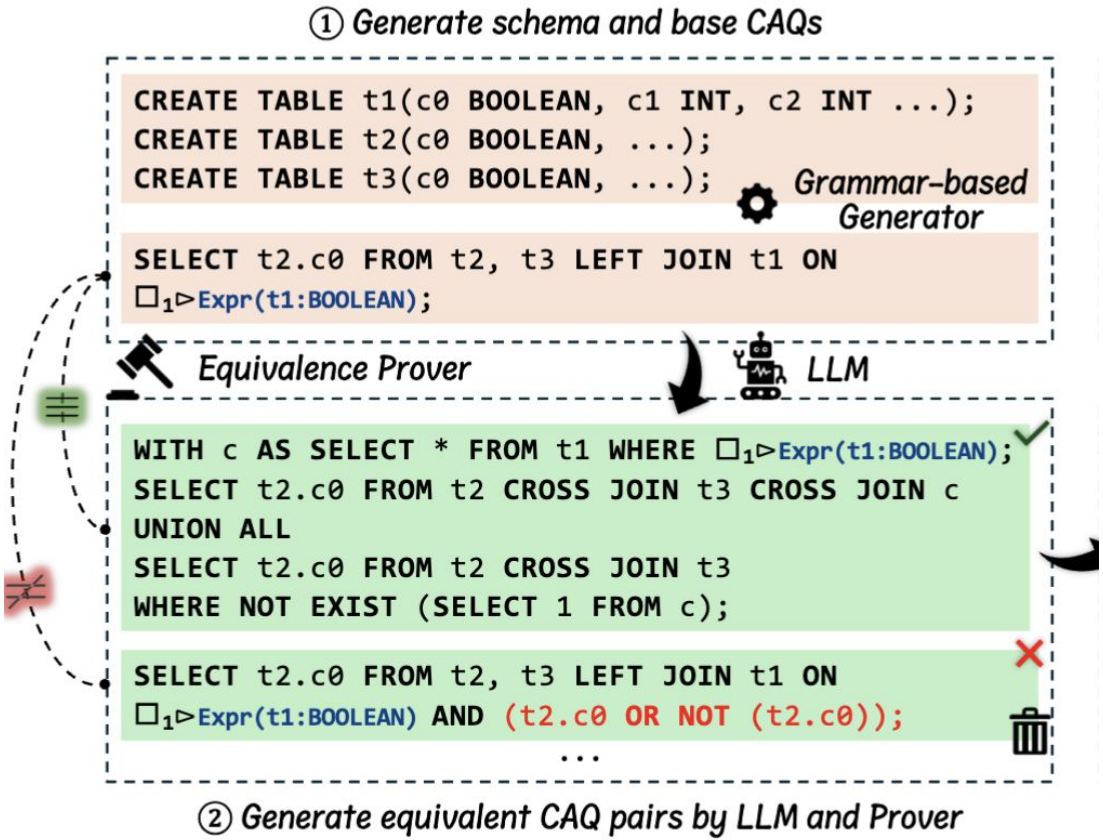
Filtering true bug reports from a lot of **false positives** is crazy for developers

Generating Test Oracles, Not Test Cases

Constrained Abstract Query (CAQ) can represent a set of SQL queries that can be instantiated from a template. We use Equivalent CAQS to represent test oracles.

```
CREATE TABLE t1(c0 VARCHAR, ...);
CREATE TABLE t2(...);
SELECT * FROM t1, □1 ▷ Table(...);                    -- Q1
SELECT * FROM t1, □1 ▷ Table(...)
WHERE (□2 ▷ Expr(t1:BOOLEAN) IS TRUE) UNION ALL
SELECT * FROM t1, □1 ▷ Table(...)
WHERE (□2 ▷ Expr(t1:BOOLEAN) IS FALSE) UNION ALL
SELECT * FROM t1, □1 ▷ Table(...)
WHERE (□2 ▷ Expr(t1:BOOLEAN) IS NULL);                -- Q2
□1 ▷ Table(...) ↦ t1 ASOF JOIN t2
□2 ▷ Expr(t1:BOOLEAN) ↦ json_valid(t1.c0)
```

Using Verification to Avoid Inequivalent CAQs

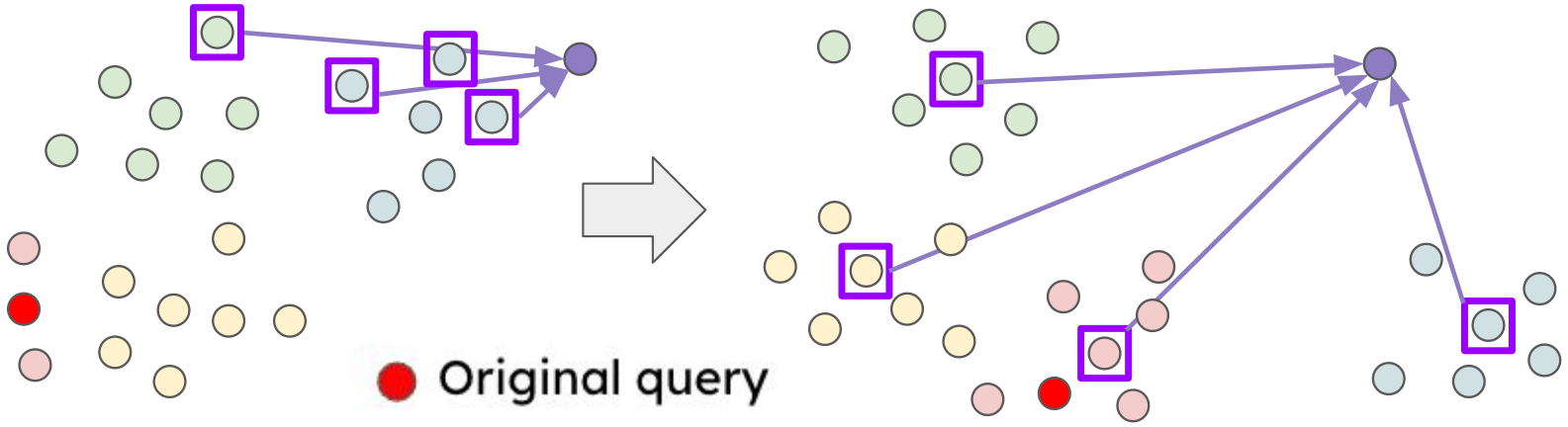


SQL Equivalence Decider can conservatively prove the equivalence between a pair of SQL queries.

Diversity **O**riented Test Oracle **G**eneration

Goal: Guide LLMs to generate diverse test cases.

Method: Evolve from Top-k centroids with highest **diversity scores**.

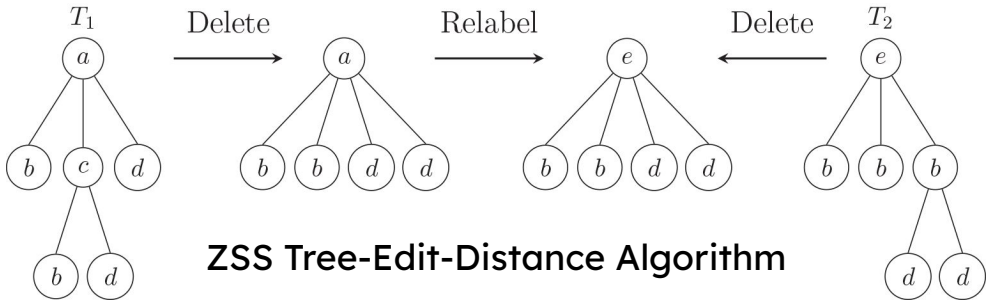


Initial: Beam search

Improved: DOG

Measuring Diversity in Equivalent CAQs

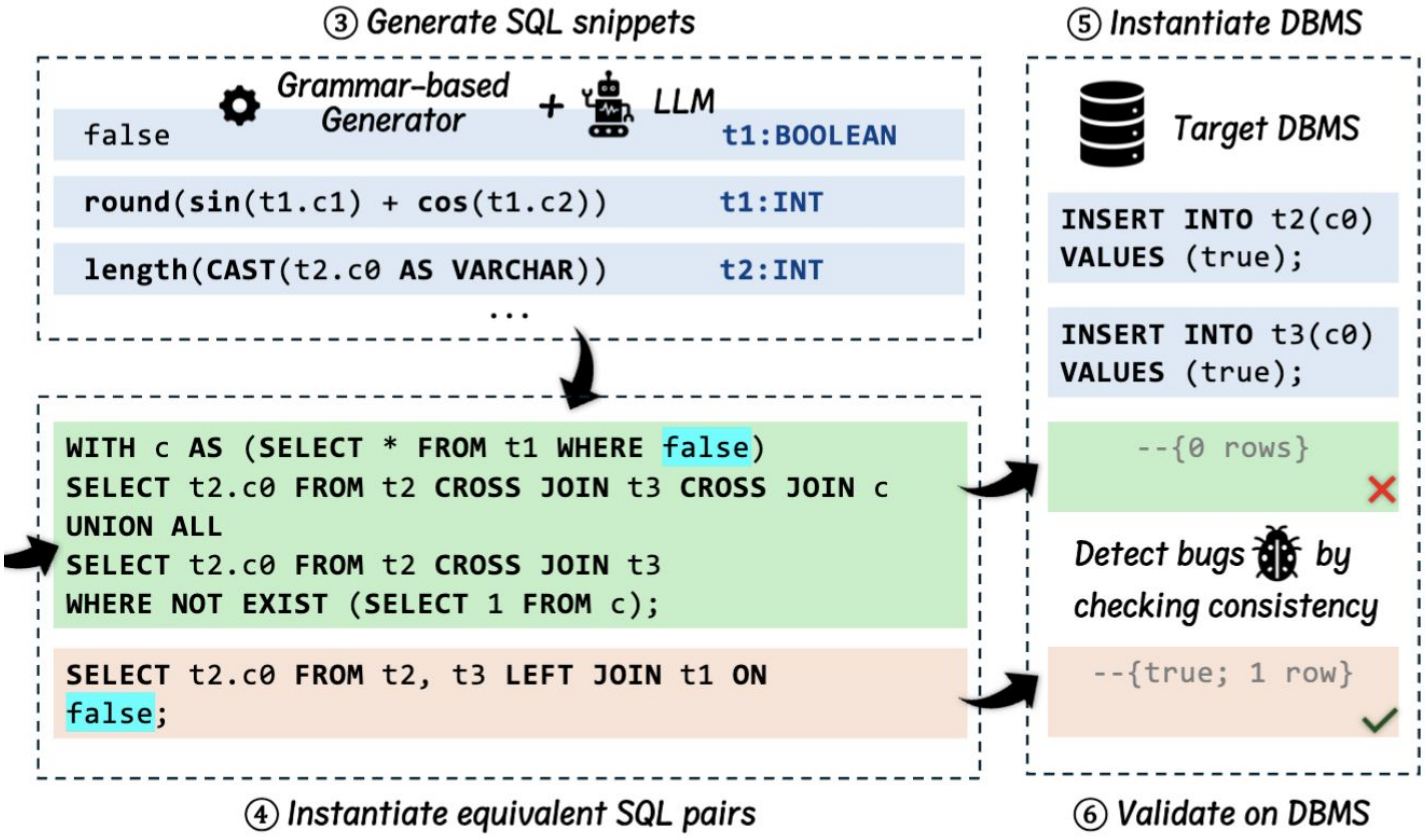
The greater the **difference in execution paths** between equivalent queries, the query is more likely to detect logic bugs in the DBMS



$$\text{Score} = \begin{cases} \dfrac{|T_1|+|T_2|}{\text{TreeEditDistance}(T_1,T_2)}, & \text{provable} \\ 0, & \text{otherwise} \end{cases}$$

Measure CAQ Diversity by the difference of two **query plans**

From Test Oracles to Scalable Test Cases



LLM-generated Test Cases Uncover Unknown Bugs

DBMS	Reported	Bug status				Bug type	
		Fixed	Conf.	Dup.	Pend.	Logic	Other
Dolt	19	18	1	0	0	18	1
DuckDB	8	6	0	1	1	4	4
MySQL	8	0	5	1	2	8	0
PostgreSQL	1	1	0	0	0	1	0
TiDB	5	2	3	0	0	5	0
Total	41	27	9	2	3	36	5

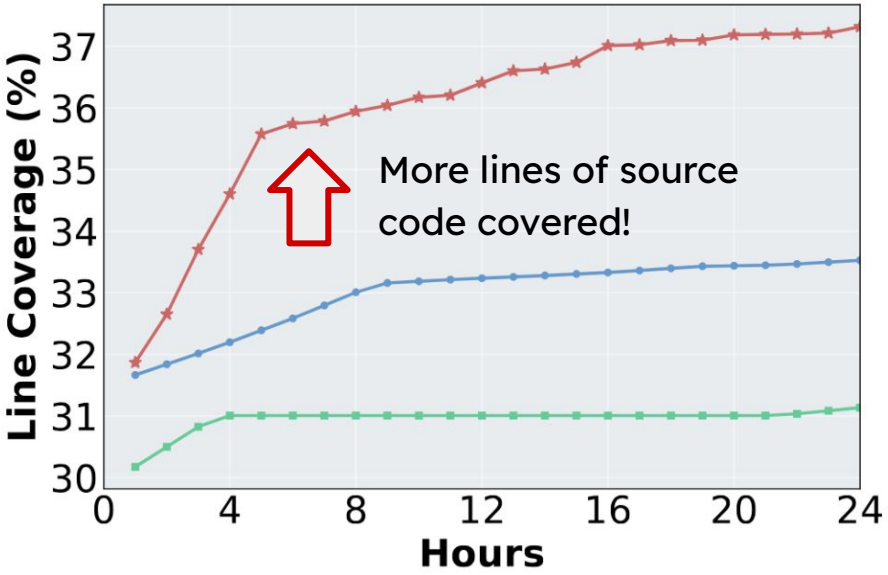
We implement our approach as **Argus**, an LLM-powered DBMS testing tool, and uncover **41** previously unknown bugs across five mature DBMSs using **GPT o4-mini**.

LLM-generated Test Cases Uncover Unknown Bugs

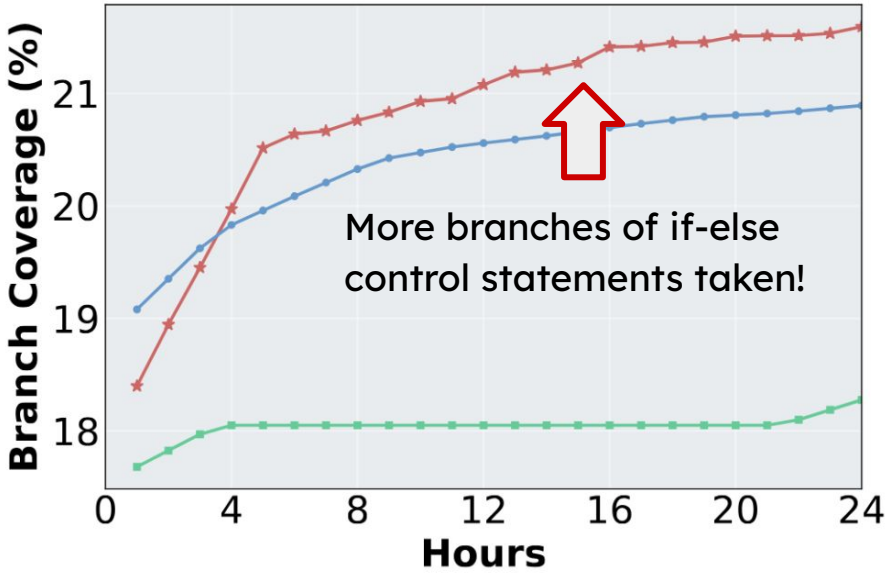

```
CREATE TABLE t(c0 INT);
INSERT INTO t VALUES (1);
SELECT * FROM t LEFT JOIN (
    SELECT MOD(5, 2) AS c0 FROM t
) AS t2 ON FALSE
WHERE t2.c0 IS NOT NULL; -- {1} 🐞 {} ✔
```


```
CREATE TABLE t1(c INT);
INSERT INTO t1 VALUES (1);
SELECT c / 3 FROM t1 WHERE false; -- {} ✔
SELECT c / 3 FROM t1 EXCEPT SELECT c / 3 FROM t1;
-- {0.3333} 🐞
```

LLM-generated Test Cases Extend Test Coverages



(a) DuckDB line



(b) DuckDB branch

Argus SQLancer SQLancer++

Takeaway #1: (Database) research is shifting from solving problems to shaping them into proper open-ended problems

Topic #2: How can we benchmark LLMs' capabilities in solving open-ended problems?

FrontierCS 2025-12-18

FrontierCS: Evolving Challenges for Evolving Intelligence

Contributors (*equal contribution)

Qiuyang Mang[1,*], Wenhao Chai[2,*], Zhifei Li[1,*], Huanzhi Mao[1,*], Shang Zhou[3,*], Alexander Du[1,4,*], Hanchen Li[1,*], Shu Liu[1,*], Edwin Chen[5], Yichuan Wang[1], Xieting Chu[6], Zerui Cheng[2], Yuan Xu[4], Tian Xia[1], Zirui Wang[1], Tianneng Shi[1], Jianzhu Yao[2], Yilong Zhao[1], Qizheng Zhang[7], Charlie Ruan[1], Zeyu Shen[2], Kaiyuan Liu[8], Runyuan He[1], Dong Xing[4], Zerui Li[4], Zirong Zeng[1], Yige Jiang[9], Lufeng Cheng[10], Ziyi Zhao[11], Youran Sun[1], Wesley Zheng[1], Meiyuwang Zhang[5], Ruyi Ji[12], Xuechang Tu[6], Zihan Zheng[13], Zexing Chen[3], Kangyang Zhou[14], Zhaozi Wang[13], Jingbang Chen[5]

Advisors (*equal advising)

Aleksandra Korolova[2], Peter Henderson[2], Pramod Viswanath[2], Vijay Ganesh[6], Saining Xie[13], Zhuang Liu[2], Dawn Song[1], Sewon Min[1], Ion Stoica[1], Joseph E. Gonzalez[1,*], Jingbo Shang[3,*], Alvin Cheung[1,*]

Affiliations

[1]UC Berkeley [2]Princeton University [3]UCSD [4]X-camp Academy [5]Independent [6]Georgia Tech [7]Stanford University [8]University of Washington [9]Nanyang Technological University [10]University of Toronto [11]UIUC [12]University of Michigan [13]New York University [14]MIT

Abstract: We introduce FrontierCS, a benchmark of 156 open-ended problems across diverse areas of computer science, designed and reviewed by experts, including CS PhDs and top-tier competitive programming participants and problem setters. Unlike existing benchmarks that focus on tasks with known optimal solutions, FrontierCS targets problems where *the optimal solution is unknown, but the quality of a solution can be objectively evaluated.* Models solve these tasks by implementing executable programs rather than outputting a direct answer. FrontierCS includes algorithmic problems, which are often NP-hard variants of competitive programming problems with objective partial scoring, and research problems with the same property. For each problem we provide an expert reference solution and an automatic evaluator. Combining open-ended design, measurable progress, and expert curation, FrontierCS provides a benchmark at the frontier of computer-science difficulty. Empirically, we find that frontier reasoning models still lag far behind human experts on both the algorithmic and research tracks, that increasing reasoning budgets alone does not close this gap, and that models often over-optimize for generating merely workable code instead of discovering high-quality algorithms and system designs.

Project page: www.frontier-cs.org

Code and Data: Frontier-CS GitHub

FrontierCS: Evolving Challenges for Evolving Intelligence

To be appear in ICML 2026

Qiuyang Mang, Wenhao Chai*, Zhifei Li*, Huanzhi Mao*, Shang Zhou*, Alexander Du*, Hanchen Li*, Shu Liu*, ..., and Alvin Cheung*

We Don’t Know the **Best**, but We Know How **Good**

Objectively Verifiable Evaluation

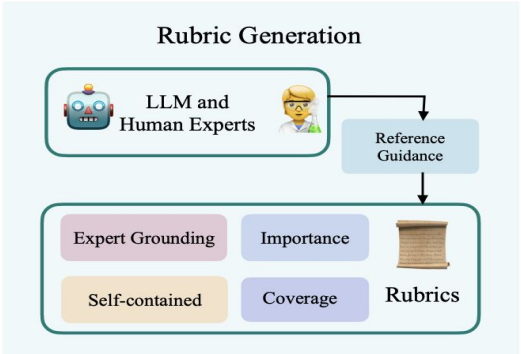
Open, deterministic evaluation

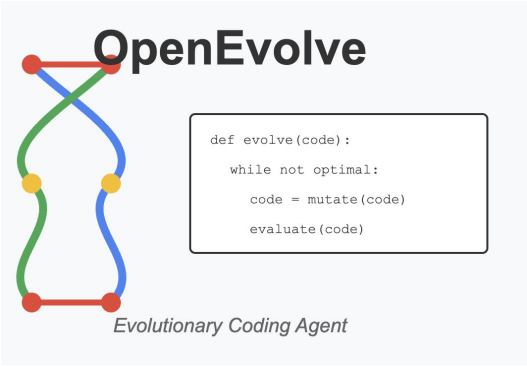
No LLM-as-a-judge

Anybody can rerun it

Runs it independently

Gets the same score





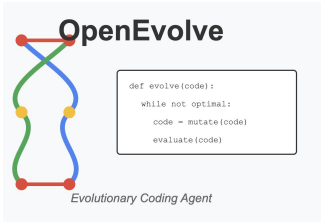
⇒ *Continuous scores guide agentic evolution and support post-training*

LLMs for Open-Ended Problems: Where We Are Limited

KernelBench: ~200 GPU programming problems (single-domain)

AI-Driven
Research Systems

ADRS: ~10 system research problems

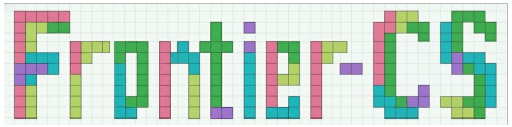


OpenEvolve: 14 example problems

ALE-bench: 40 Atcoder Heuristic Contest problems (single-source)

We need a **large-scale benchmark** for open-ended problems, spanning multiple domains, sources, and problem types.

Introducing Frontier-CS



= **200+** open-ended problems adapted from competitive programming and real CS research

- **Algorithmic Track**: 10+ experts converse exam-style problems from multiple sources into an open-ended style by *changing objectives*, *adding constraints* . . . 

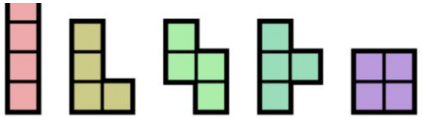
- **Research Track**: CS PhDs extract the core algorithmic problems from their research interests, including OS, HPC, AI, DB, PL, . . .

An **Open-Ended**, **Verifiable** Example with **Continuous** Score

FrontierCS

open-ended

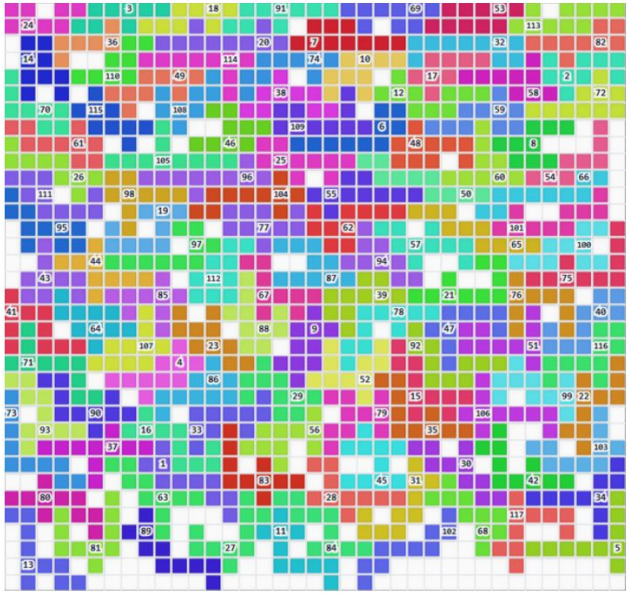
verifiable



Example: Polyomino Packing
Pack all polyominoes as tightly as possible into the grid.

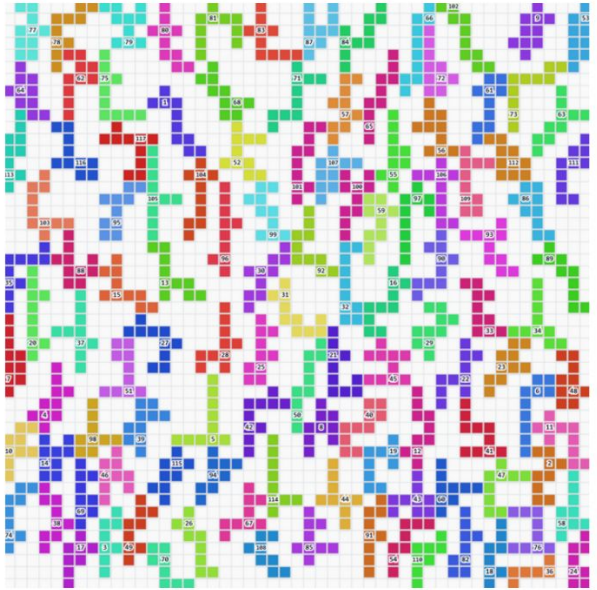
Human Expert

87%



GPT-5 Thinking

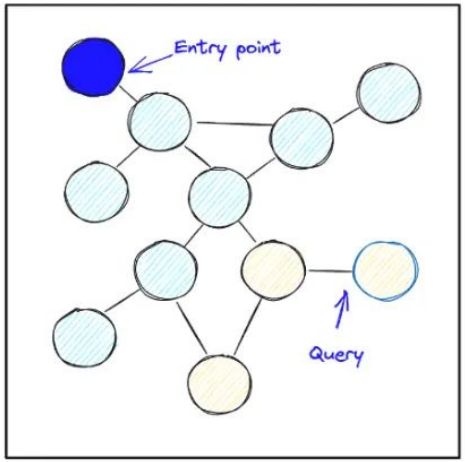
47%



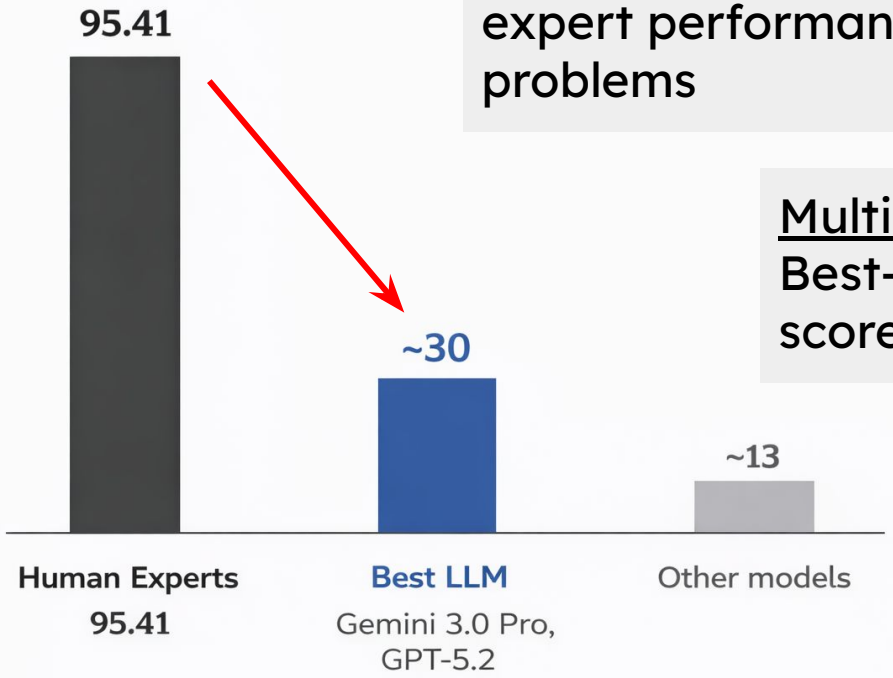
$$\text{score} = \begin{cases} 0 & \text{invalid arrangement} \\ 100 \cdot \dfrac{\text{total area covered}}{\text{grid area}} & \text{otherwise} \end{cases}$$

Research Problem Example: VectorDB Design

- Given: Millions of vectors and a set of query vectors.
- Task: For each query, return the nearest stored vector as fast as possible, while being correct at least 80% of the time.
- Scoring: Recall@1 ≥ 80% is required; passing solutions receive a 0–100 score normalized by average query latency relative to a human SOTA.



Open-ended Problems Remain Challenging for Frontier LLMs

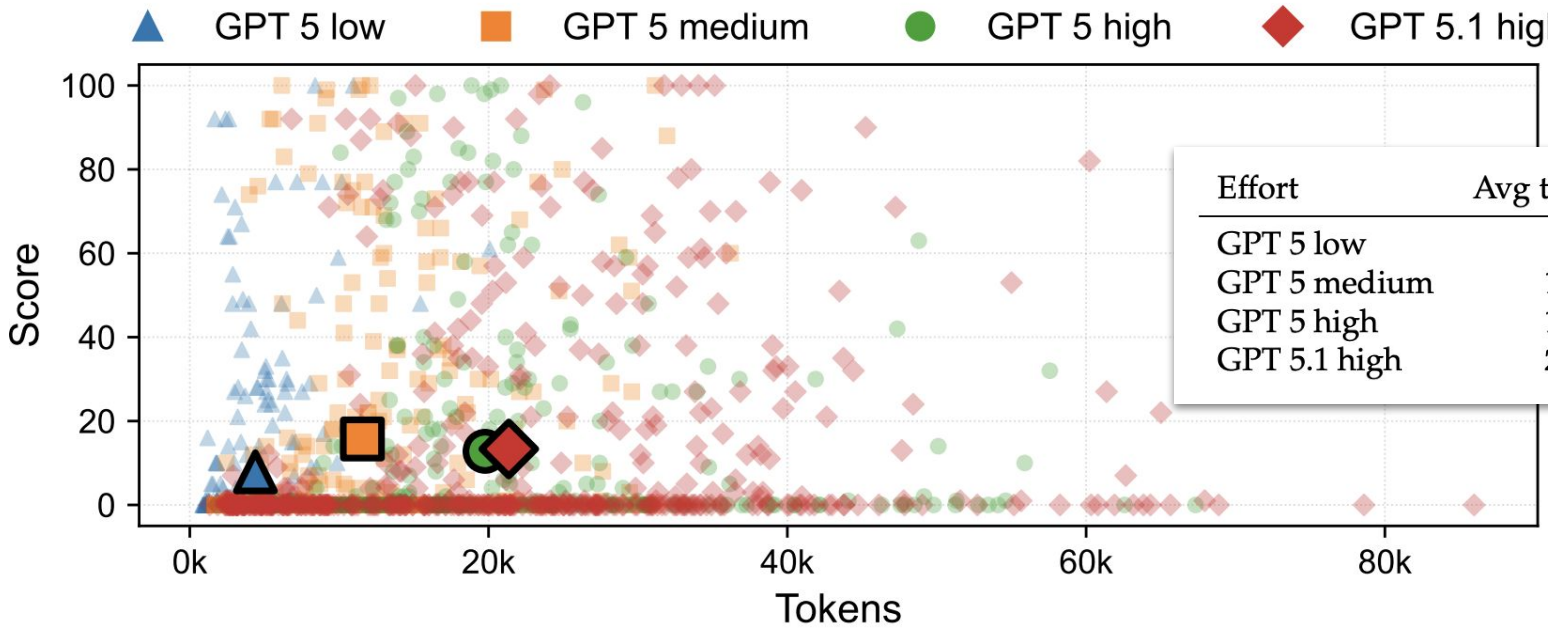


Algorithmic Problems (one-shot, until 12/17)

LLMs remains a major gap to human expert performance on open-ended problems

Multiple attempts matter
Best-of-5 improves average score by **+6.5 – +22.7**

Improving Reasoning Effort Does Not Yield Further Gains



Effort	Avg tokens	Avg score
GPT 5 low	4,389	7.903
GPT 5 medium	11,554	**15.336**
GPT 5 high	19,763	12.626
GPT 5.1 high	**20,402**	12.508

Longer chain-of-thought not always help open-ended problem solving.

Open-ended Problems are Long-horizon for Agents

Metric	Claude Opus 4-7	Kimi K2.6	Δ (Difference)
Partial Score	35.1	34.0	**Tied**
Avg. Steps	83.0 (up to 459)	56.5 (up to 405)	**Opus +47%**
Avg. Turns	80.8 (up to 456)	55.7	**Opus +45%**
Avg. Output Tokens	200K (up to 531K)	149K	**Opus +34%**
Avg. Thinking Calls	—	56.5 (up to 404)	**Every step**
Avg. Tool Calls	45.0 (up to 291)	55.7 (up to 405)	**Kimi +24%**
Zero-Score Tasks	52 / 105	50 / 105	**~Half**
Token Limit Hit	17	8	**Opus +113%**

Takeaway #2: Open-ended problems remain underexplored by LLMs and pose new challenges for long-horizon coding agents.

Topic #3: Can we improve models' open-ended problem-solving capabilities by automatically scaling the training data?

Preprint FrontierCS Team

FrontierSmith: Synthesizing Open-Ended Coding Problems at Scale

Runyuan He[1,*], Qiuyang Mang[1,*,§], Shang Zhou[2], Kaiyuan Liu[3], Hanchen Li[1], Huanzhi Mao[1], Qizheng Zhang[4], Zerui Li, Bo Peng[5], Lufeng Cheng, Tianfu Fu[6], Yichuan Wang[1], Wenhao Chai[5], Jingbo Shang[2], Alex Dimakis[1], Joseph E. Gonzalez[1], Alvin Cheung[1]

[1]UC Berkeley, [2]UC San Diego, [3]University of Washington, [4]Stanford University, [5]Princeton University, [6]Massachusetts Institute of Technology

Correspondence: qmang@berkeley.edu, wenhao.chai@princeton.edu

Model: huggingface.co/runyuanhe/qwen35-9b-frontiersmith

Sample Problems & Training Scripts: github.com/FrontierCS/FrontierSmith

Abstract

Many real-world coding challenges are open-ended and admit no known optimal solution. Yet, recent progress in LLM coding has focused on well-defined tasks such as feature implementation, bug fixing, and competitive programming. Open-ended coding remains a weak spot for LLMs, largely because open-ended training problems are scarce and expensive to construct. Our goal is to synthesize open-ended coding problems at scale to train stronger LLM coders. We introduce *FrontierSmith*, an automated system for iteratively evolving open-ended problems from existing closed-ended coding tasks. Starting from competitive programming problems, FrontierSmith generates candidate open-ended variants by changing the problems' goals, restricting outputs, and generalizing inputs. It then uses a quantitative idea divergence metric to select problems that elicit genuinely diverse approaches from different solvers. Agents then generate test cases and verifiers for the surviving candidates. On two open-ended coding benchmarks, training on our synthesized data yields substantial gains over the base models: Qwen3.5-9B improves by +8.82 score on FrontierCS and +306.36 (Elo-rating-based performance) on ALE-bench; Qwen3.5-27B improves by +12.12 and +309.12, respectively. The synthesized problems also make agents take more turns and use more tokens, similar to human-curated ones, suggesting that closed-ended seeds can be a practical starting point for long-horizon coding data.

1 Introduction

LLMs now excel at well-defined coding tasks, reaching gold-medal performance in competitive programming (ICPC Foundation, 2026) and solving over 80% of SWE-bench verified issues (Jimenez et al., 2024). Yet these settings are mostly closed-ended: correctness is discrete and efficiently verifiable. Open-ended coding tasks, in contrast, lack tractable certificates of optimality at the target scale and score submissions by continuous quality. For example, in cloud cluster scheduling, many job-to-machine assignments are feasible, but they differ continuously in makespan, tail latency, energy use, and utilization; checking feasibility is easy, whereas certifying global optimality at production scale is intractable (Li et al., 2026). This difficulty is reflected in current open-ended benchmarks. On FrontierCS (Mang et al., 2025), human experts score 95.41 on algorithmic tasks, compared with 29.37 for Gemini 3.0 Pro (Google DeepMind, 2025); on ALE-bench (Imajuku et al., 2025), frontier LLMs still trail average human participants in heuristic contests. This gap reflects a data asymmetry: verified closed-ended tasks are abundant, while open-ended tasks remain scarce. We aim to automatically synthesize open-ended coding problems at scale to train stronger LLM coders.

For closed-ended tasks, the data problem is largely solved. Codeforces (Codeforces, 2026) and LeetCode (LeetCode, 2026) provide hundreds of thousands of closed-ended problems with verified solutions, enabling reinforcement learning from verifiable rewards to drive substantial gains. Open-ended coding has no equivalent.

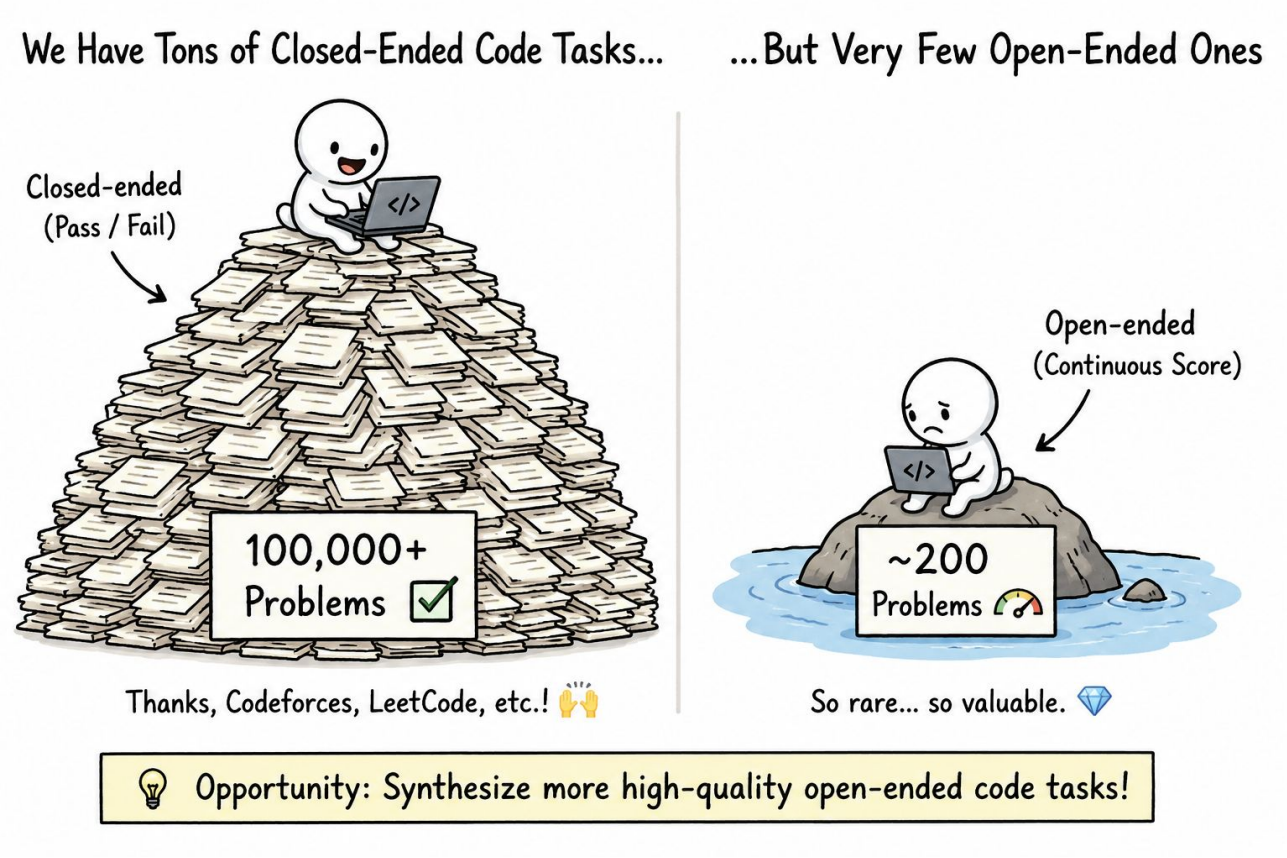
*Equal contribution, §Project lead

FrontierSmith: Synthesizing Open-ended Coding Problems at Scale

In submission to Neurips 2026

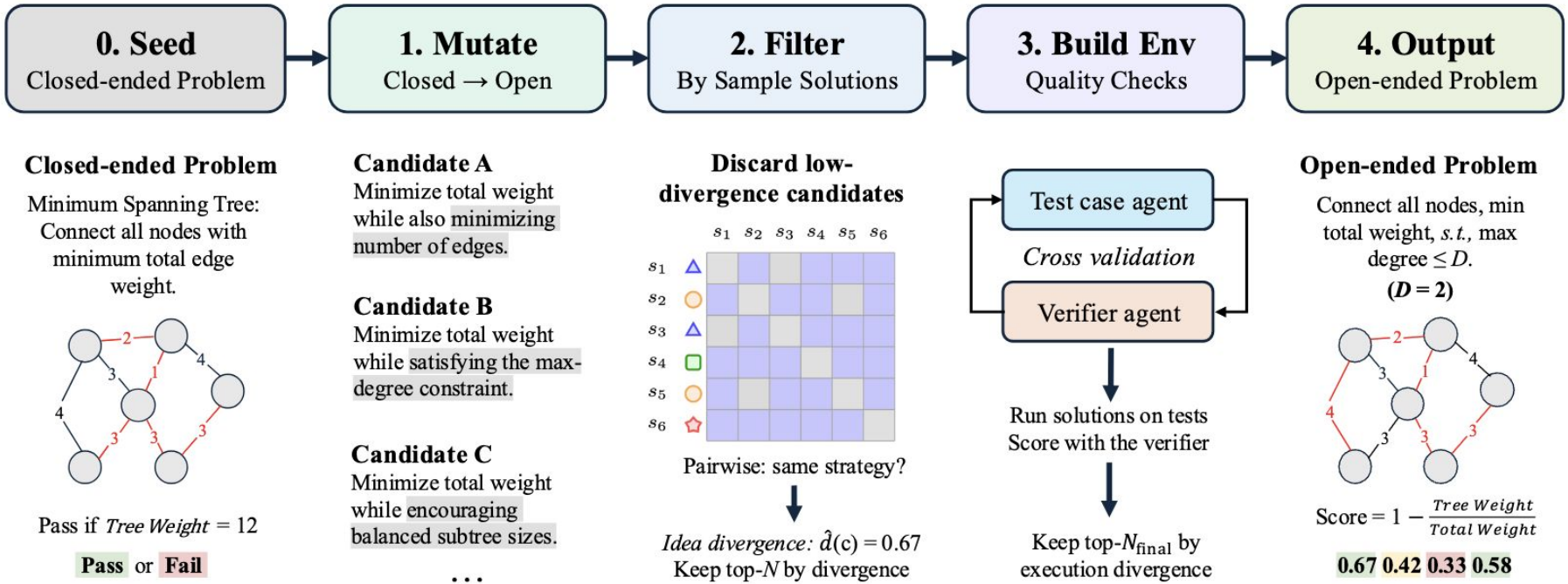
Runyuan He, **Qiuyang Mang***, Shang Zhou, Kaiyuan Liu, Hanchen Li, Huanzhi Mao, Qizheng Zhang, Zerui Li, Bo Peng, Lufeng Cheng, Tianfu Fu, Yichuan Wang, Wenhao Chai, Jingbo Shang, Alex Dimakis, Joseph E. Gonzalez, and Alvin Cheung*

Generating Open-ended Tasks from Closed-ended Ones



Can LLMs generate the data needed to train themselves?

Generating Open-ended Tasks from Closed-ended Ones



How Can We Model the Transformation in Principle

Problem Formulation = (O, CI, CO)

O: Objective | CI: Input Constraints | CO: Output Constraints

Changing Goals ($O \rightarrow O'$)

Replace exact/binary goals with optimization.

Shifts tasks from simple decision to graded performance.

Example:

2-SAT (Decision) → Min-True 2-SAT (Optimization)

Restricting Outputs ($CO \rightarrow CO'$)

Add output constraints.

Makes exact solutions infeasible at scale, favoring approximations.

Example:

MST → Degree-constrained Spanning Tree (NP-Hard)

Generalizing Inputs ($CI \rightarrow CI'$)

Relax constraints on input.

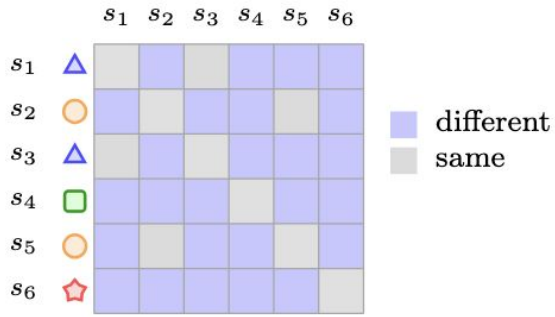
Transforms polynomial problems into NP-complete ones.

Example:

Max Independent Set: Bipartite (P) → Arbitrary Graphs (NP-C)

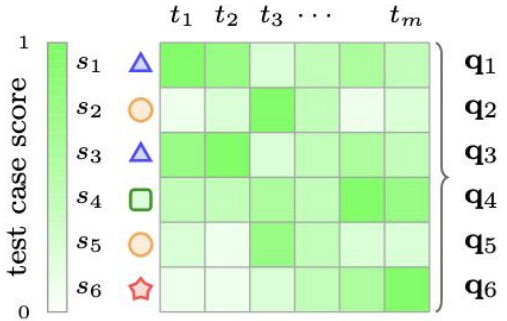
How Can We Ensure the Transformation Quality

LLM-as-a-judge-based estimation



$$\hat{d}(c) = \text{avg } \texttt{LLM-as-a-Judge}(s_i, s_j)$$

Execution-based estimation

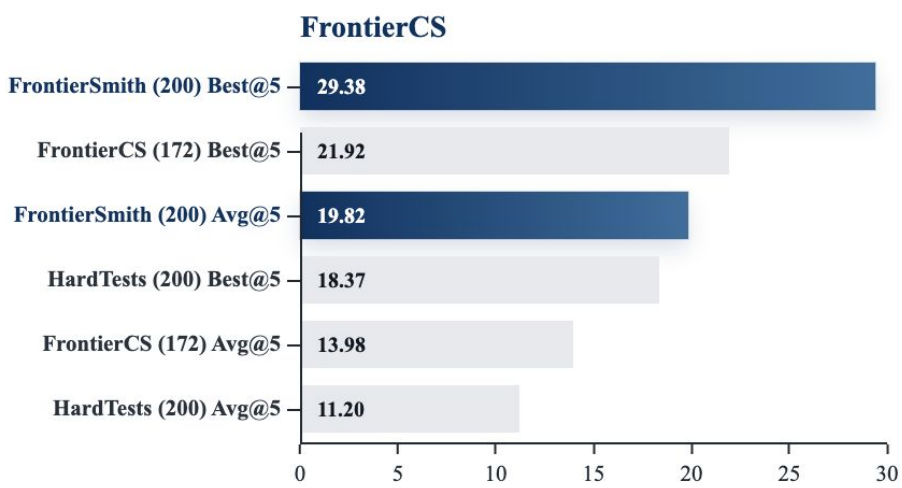


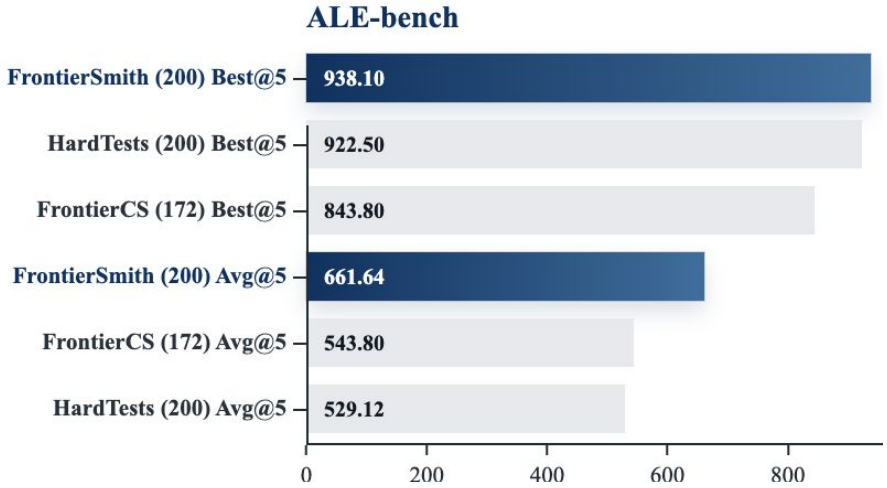
$$\hat{d}(c) = \text{avg } \frac{1}{\sqrt{m}} \|\mathbf{q}_i - \mathbf{q}_j\|_2$$

LLM-based estimation. An LLM judge labels each solution pair as same- or different-strategy. $\hat{d}(c)$ is the fraction of pairs judged different. Blue cells: different-strategy; gray cells: same-strategy.

Execution-grounded estimation. Each solution is run on the test cases $t_1, \ldots, t_m$ and scored by the verifier, yielding a score vector $\mathbf{q}_i$. $\hat{d}(c)$ is the average pairwise distance between score vectors.

Synthetic data leads Qwen-3.5-27B in open-ended coding.

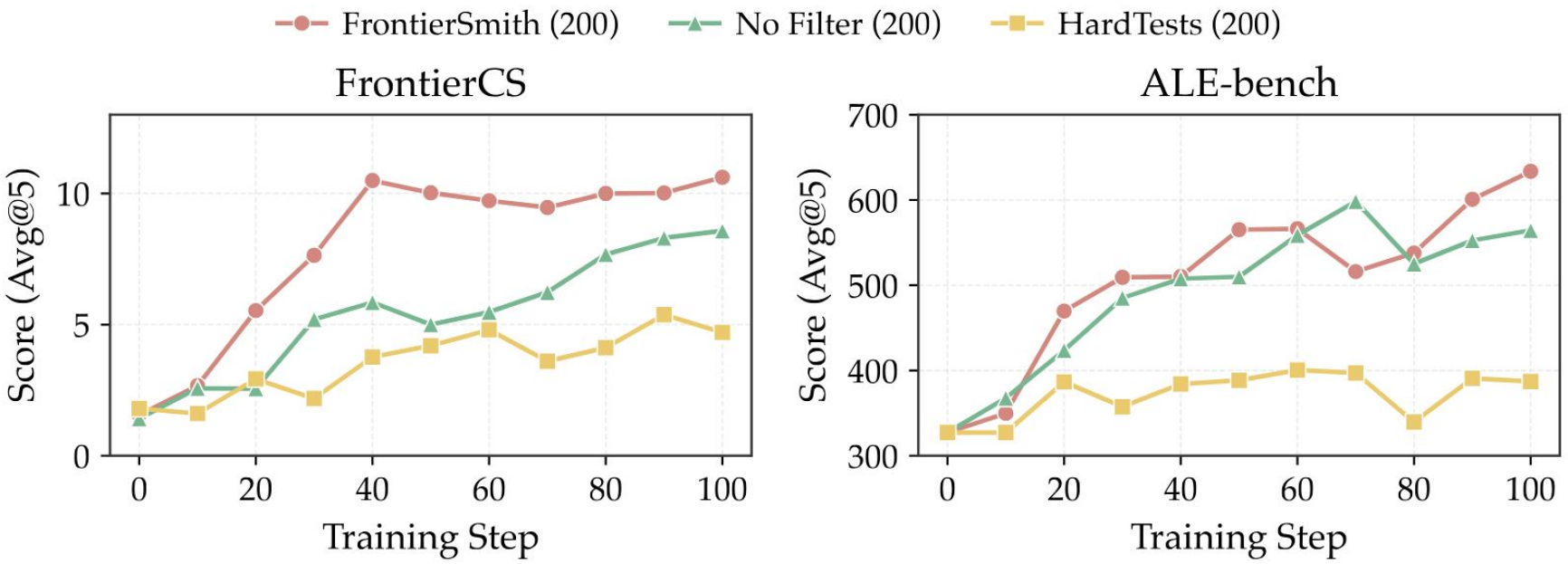




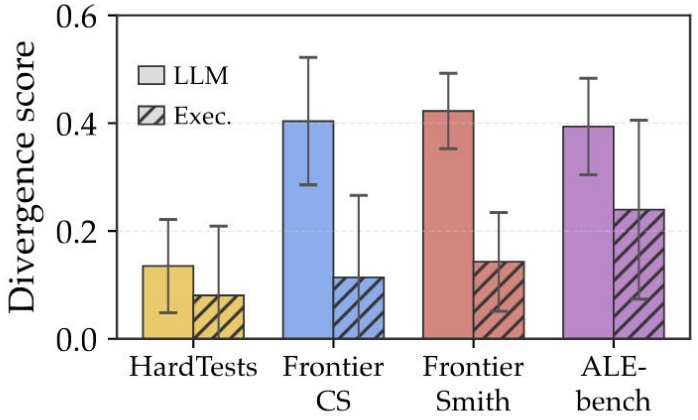
+12.12
FrontierCS Avg@5 over Base

+309.12
ALE-bench Avg@5 over Base

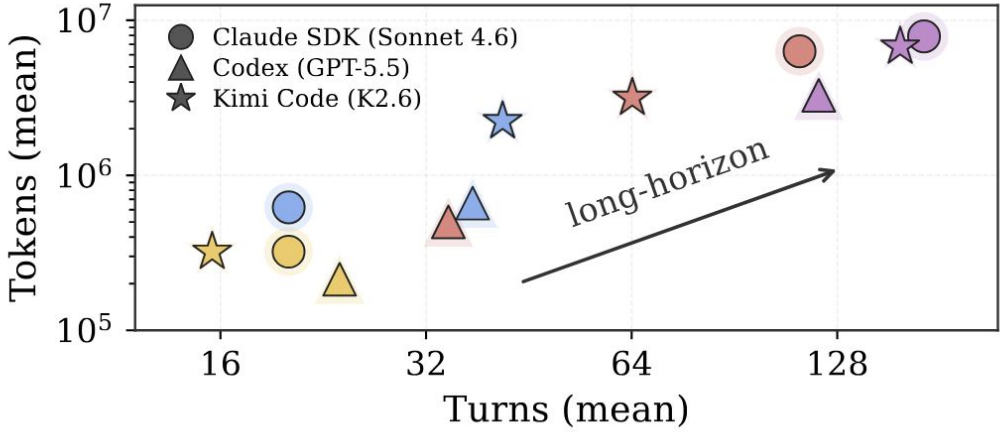
Data Quality Matters



Synthetic Tasks Match Human-curated Ones



Solution Space Diversity



Agentic Behavior

Takeaway #3: LLMs can synthesize open-ended tasks at scale to improve their capabilities in a self-play manner

- LLM-driven research is the future. The key is modeling problems as open-ended tasks.
- LLMs can well solve open-ended tasks via test-time scaling. Key requirement: the problem is well-specified.
- LLMs can synthesize open-ended data to self-improve. Next: How to also synthesize envs?
- Can LLMs also turn real-world challenges into well-specified open-ended tasks?