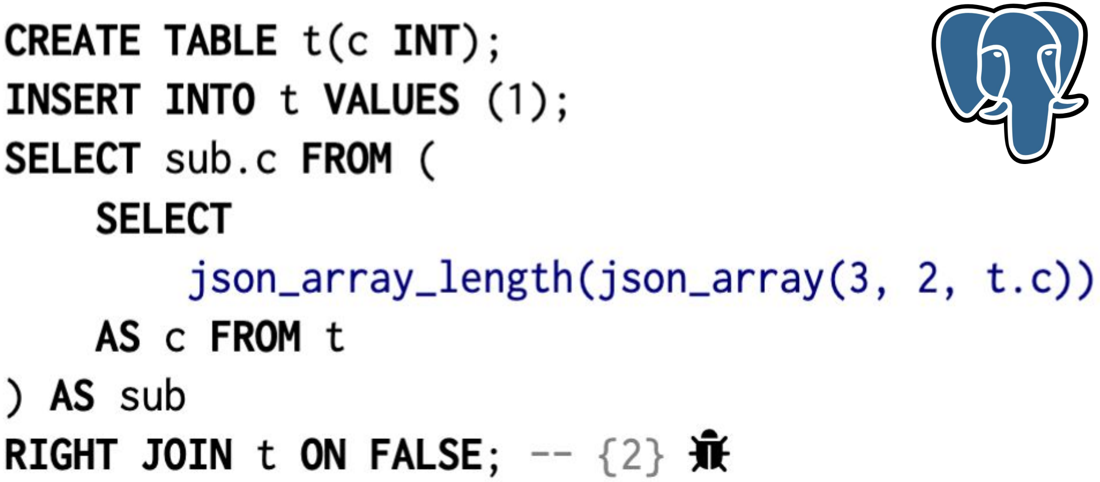
Argus: Automated Discovering Test Oracles for Database Management Systems with LLMs

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

sky

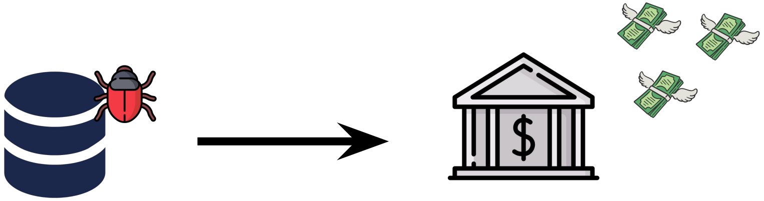
TLDR: **$10** of LLM usage generates **millions** of **reliable** DBMS test cases and uncovers **unknown** logic bugs.

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}
```

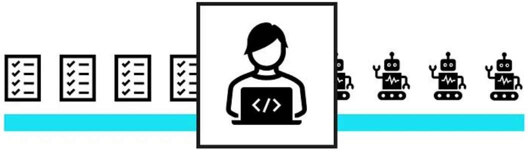
Detect such **logic bugs** in DBMS



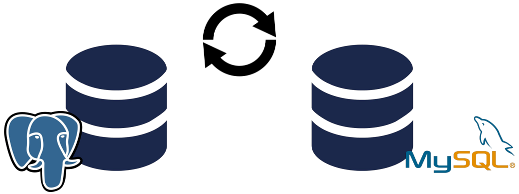
Buggy DBMS

Downstream Application

Existing DBMS Testing Methodologies



Manual Crafted Test Cases



Reference Engine

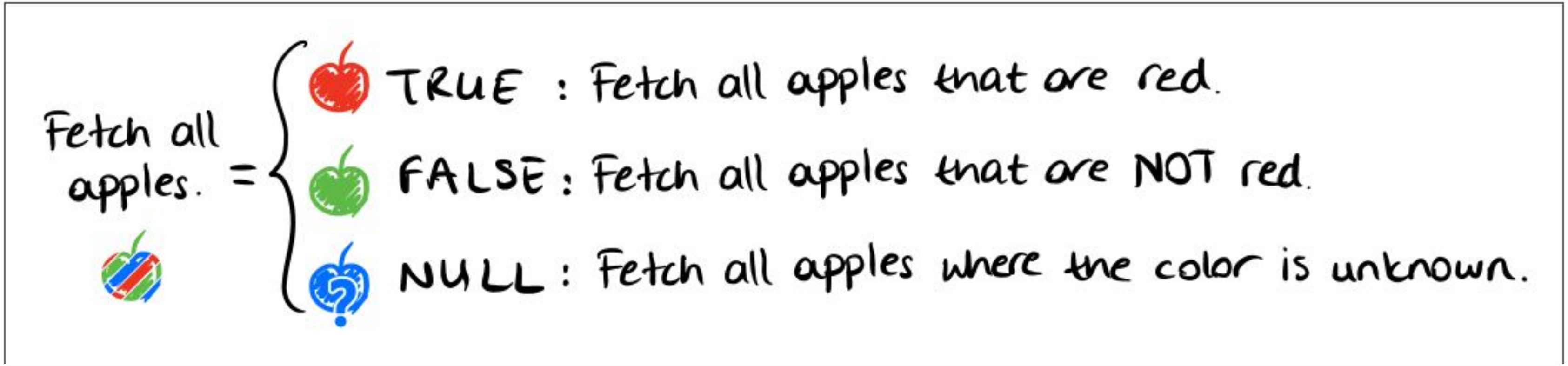
Test Oracles

⇒ *Pairs of equivalent queries*

Before 2020

2020

Example: Ternary Logic Partitioning

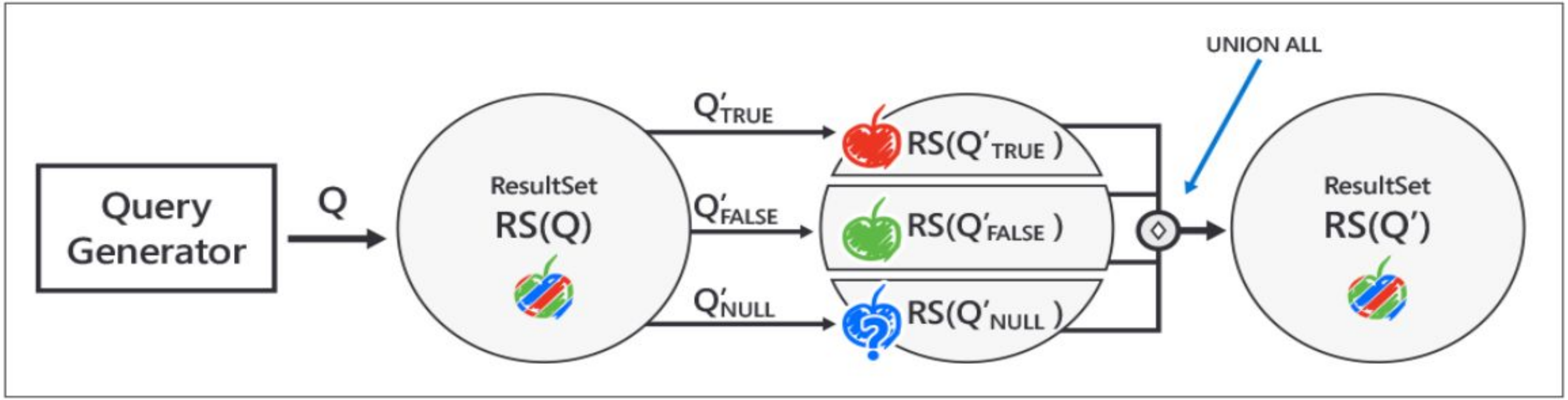


Fetch all apples. =

TRUE : Fetch all apples that are red.

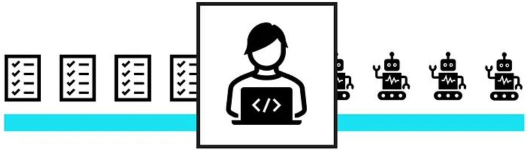
FALSE : Fetch all apples that are NOT red.

NULL : Fetch all apples where the color is unknown.

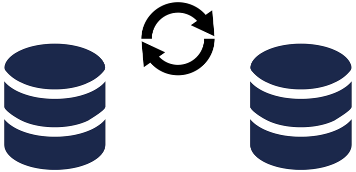


Existing DBMS Testing Solutions

Low Bug Coverage



Manual Crafted Test Cases



Reference Engine

Test Oracles

⇒ *Pairs of equivalent queries*

Researcher

New Oracles

New Papers

New Bugs

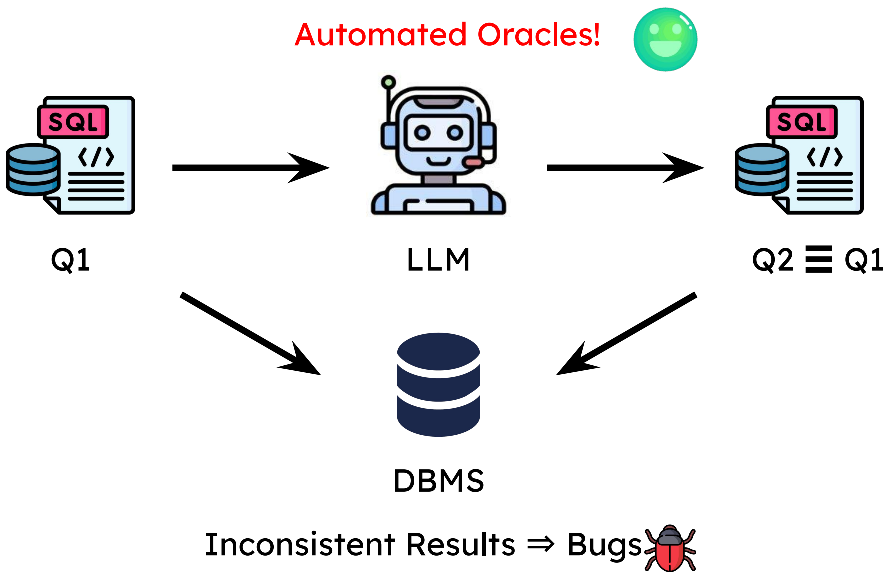
Before 2020

2020

2020 – 2025

Limited Test Oracles Hand-made by Human Experts

Using LLM to Break the Endless Cycle



Automated Oracles!

Q1

LLM

Q2 ≡ Q1

DBMS

Inconsistent Results ⇒ Bugs

C1: LLMs are slow and expensive

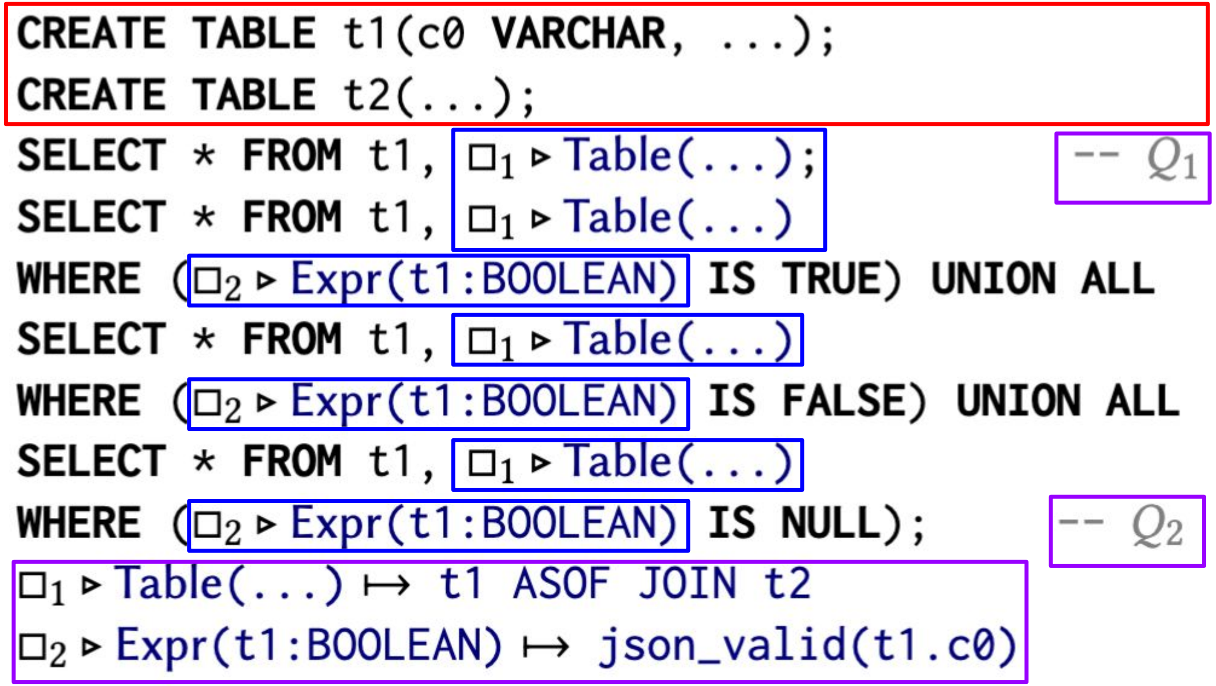
We needs about **100K** test cases to detect one unique bugs in mature DBMS :0

C2: Hallucination ⇒ False Alarms

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

M1: Generating Test Oracles, Not Test Cases

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



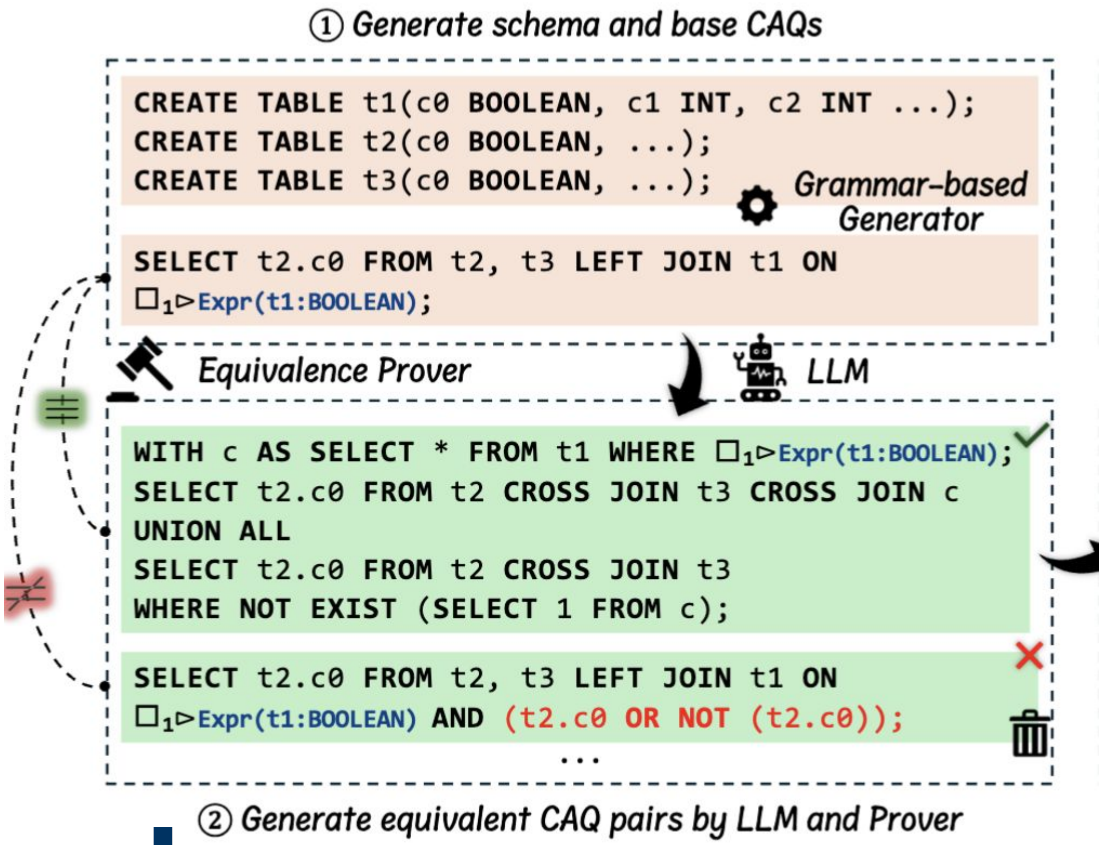
```
CREATE TABLE t1(c0 VARCHAR, ...);
CREATE TABLE t2(...);
SELECT * FROM t1, □1 ▹ Table(...);                    -- Q1
SELECT * FROM t1, □1 ▹ Table(...)
WHERE (□2 ▹ Expr(t1:BOOLEAN)) IS TRUE) UNION ALL
SELECT * FROM t1, □1 ▹ Table(...)
WHERE (□2 ▹ Expr(t1:BOOLEAN)) IS FALSE) UNION ALL
SELECT * FROM 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)
```

1. Query Schema

2. Placeholders

3. Test Oracle: Q1 and Q2 are **semantically equivalent** under all possible instantiations of their placeholders.

M2: Using Verification to Avoid Inequivalent CAQs


① Generate schema and base CAQs
Grammar-based Generator
Equivalence Prover
LLM
≡
≢
✓
✗
② Generate equivalent CAQ pairs by LLM and Prover

```
CREATE TABLE t1(c0 BOOLEAN, c1 INT, c2 INT ...);
CREATE TABLE t2(c0 BOOLEAN, ...);
CREATE TABLE t3(c0 BOOLEAN, ...);
```

```
SELECT t2.c0 FROM t2, t3 LEFT JOIN t1 ON
□1▷Expr(t1:BOOLEAN);
```

```
WITH c AS SELECT * FROM t1 WHERE □1▷Expr(t1:BOOLEAN);
SELECT t2.c0 FROM t2 CROSS JOIN t3 CROSS JOIN c
UNION ALL
SELECT t2.c0 FROM t2 CROSS JOIN t3
WHERE NOT EXIST (SELECT 1 FROM c);
```

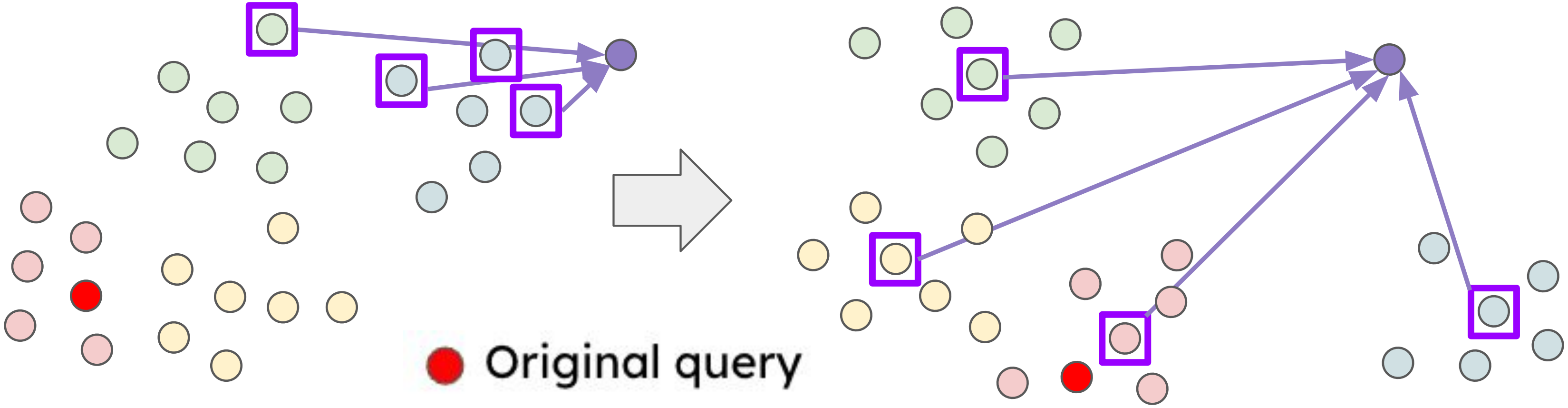
```
SELECT t2.c0 FROM t2, t3 LEFT JOIN t1 ON
□1▷Expr(t1:BOOLEAN) AND (t2.c0 OR NOT (t2.c0));
...
```

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

M3: **D**iversity **O**riented Test Oracle **G**eneration

Goal: Guide LLMs to generate diverse test cases.

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



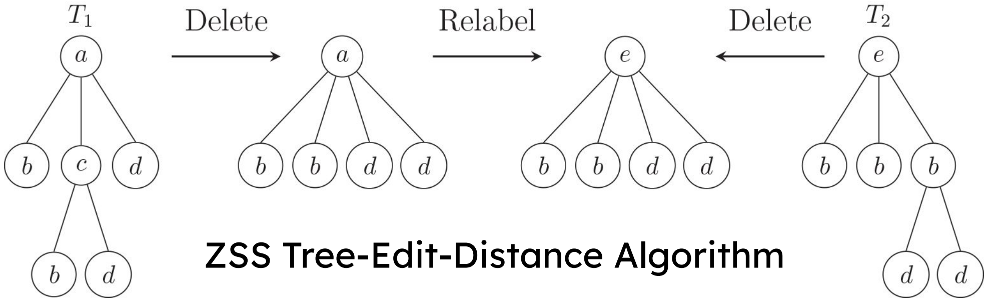
Initial: Beam search

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 (Ba et al. 2025).

LLMs are guided to generate semantically equivalent queries with **highly different** paths

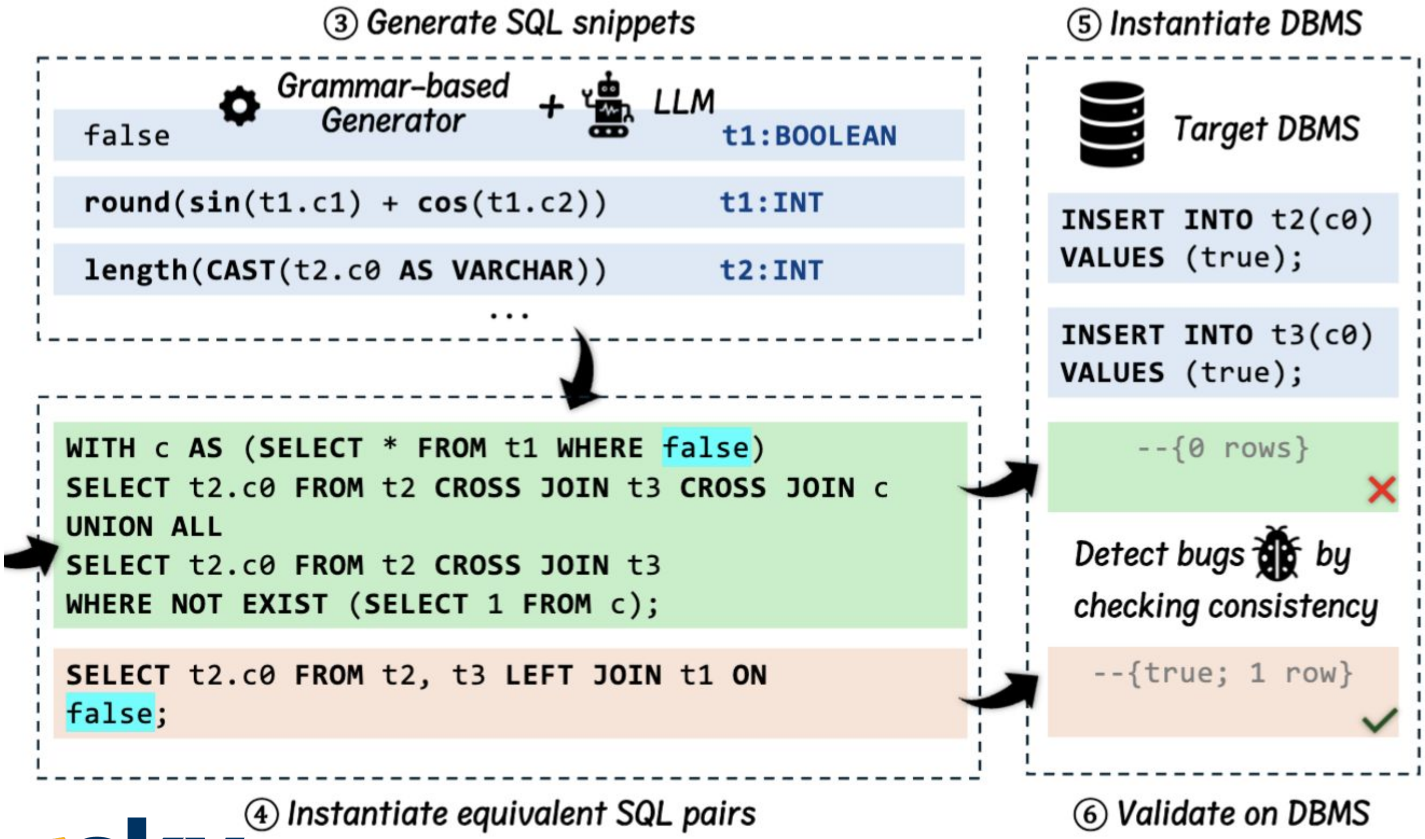


ZSS Tree-Edit-Distance Algorithm

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

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

M4: From Test Oracles to Scalable Test Cases



```
false                                        t1:BOOLEAN
round(sin(t1.c1) + cos(t1.c2))               t1:INT
length(CAST(t2.c0 AS VARCHAR))               t2:INT
...
```

```
WITH c AS (SELECT * FROM t1 WHERE false)
SELECT t2.c0 FROM t2 CROSS JOIN t3 CROSS JOIN c
UNION ALL
SELECT t2.c0 FROM t2 CROSS JOIN t3
WHERE NOT EXIST (SELECT 1 FROM c);
```

```
SELECT t2.c0 FROM t2, t3 LEFT JOIN t1 ON
false;
```

```
INSERT INTO t2(c0)
VALUES (true);
```

```
INSERT INTO t3(c0)
VALUES (true);
```

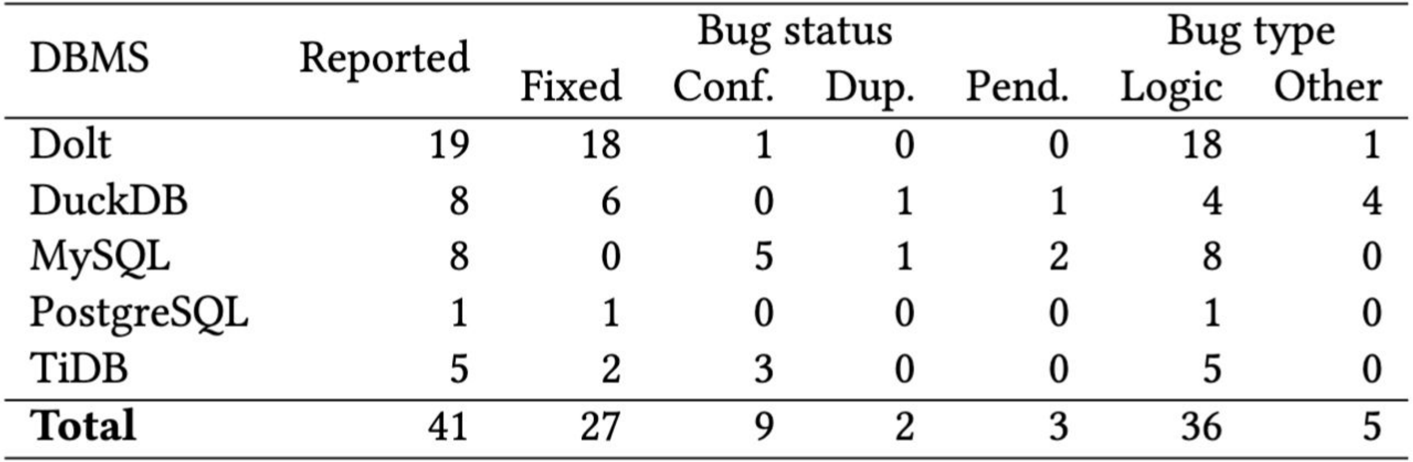
```
--{0 rows}
```

✗

```
--{true; 1 row}
```

✓

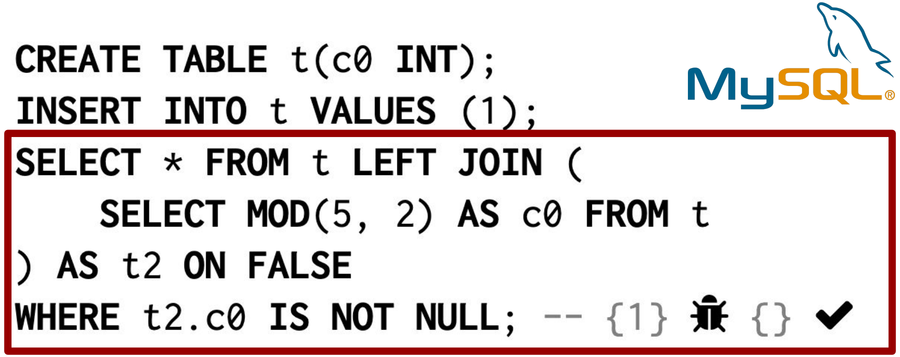
E1: LLM-generated Test Cases Uncover Unknown Bugs



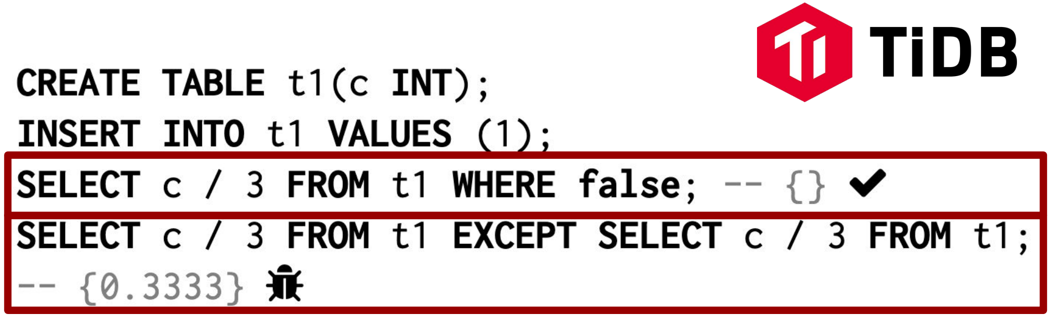
DBMS	Reported	Bug status				Bug type	
		Fixed	Conf.	Dup.	Pend.	Logic	Other
Dolt	19	18	1	0	0	18	1
DuckDB	8	6	0	1	1	4	4
MySQL	8	0	5	1	2	8	0
PostgreSQL	1	1	0	0	0	1	0
TiDB	5	2	3	0	0	5	0
Total	41	27	9	2	3	36	5

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**.

E1: LLM-generated Test Cases Uncover Unknown Bugs

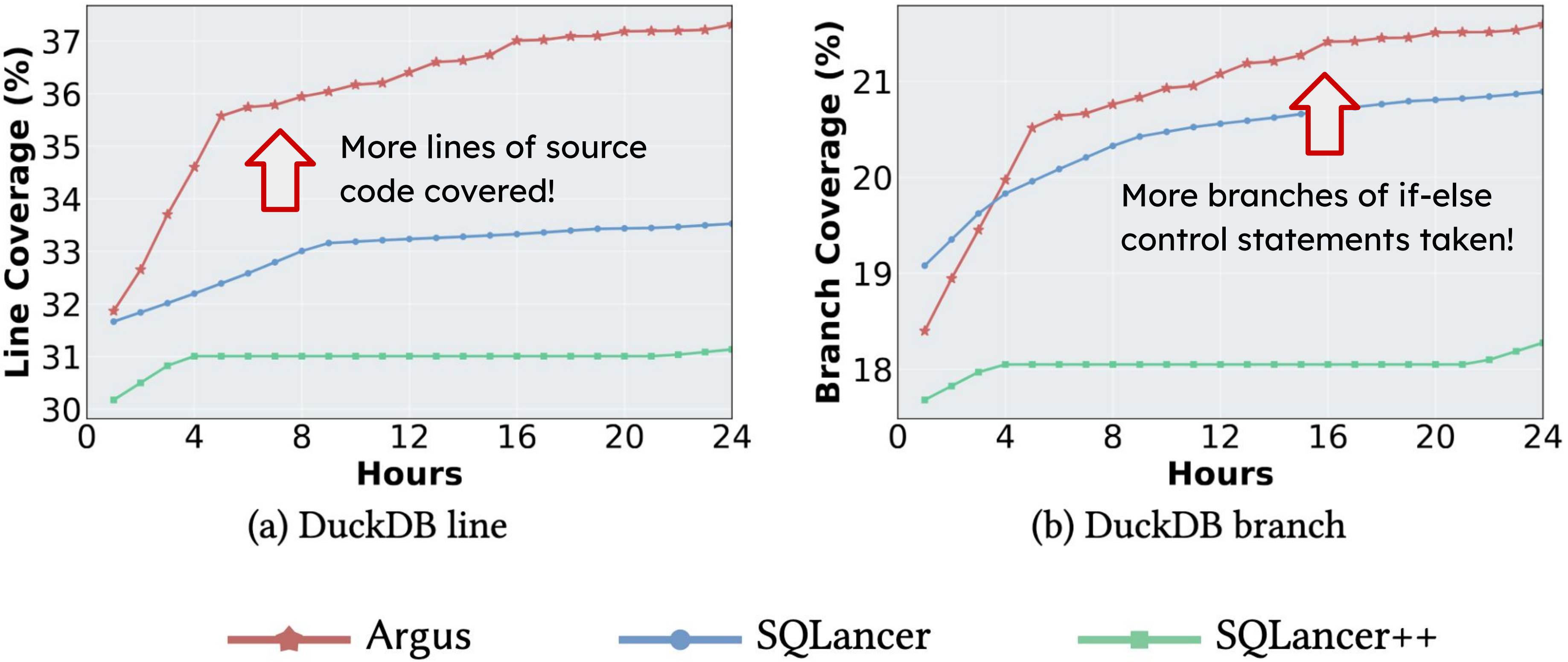


```
CREATE TABLE t(c0 INT);
INSERT INTO t VALUES (1);
SELECT * FROM t LEFT JOIN (
    SELECT MOD(5, 2) AS c0 FROM t
) AS t2 ON FALSE
WHERE t2.c0 IS NOT NULL; -- {1} 🐞 {} ✔
```



```
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} 🐞
```

E2: LLM-generated Test Cases Extend Test Coverages



More lines of source code covered!

More branches of if-else control statements taken!

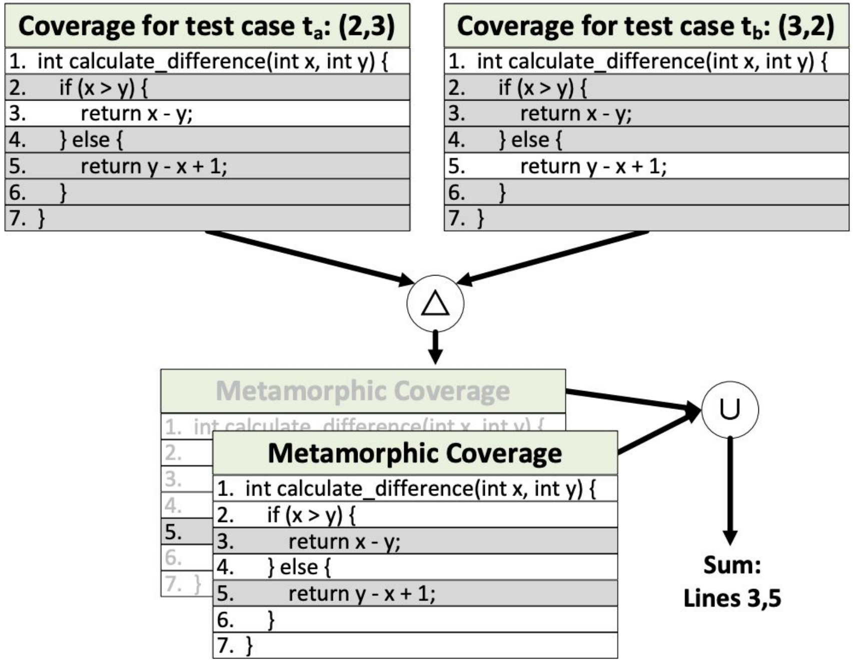
(a) DuckDB line

(b) DuckDB branch

E2: LLM-generated Test Cases Extend Test Coverages

Approach	Lines	Functions	Branches
SQLancer	3.256%	1.230%	1.313%
Argus	**17.820%**	**7.910%**	**7.315%**
	5.473×	6.431×	5.571×

Higher **metamorphic coverage** means we exercise more DBMS code paths under the same query semantics.



Coverage for test case t_a: (2,3)

```
1. int calculate_difference(int x, int y) {
2.     if (x > y) {
3.         return x - y;
4.     } else {
5.         return y - x + 1;
6.     }
7. }
```

Coverage for test case t_b: (3,2)

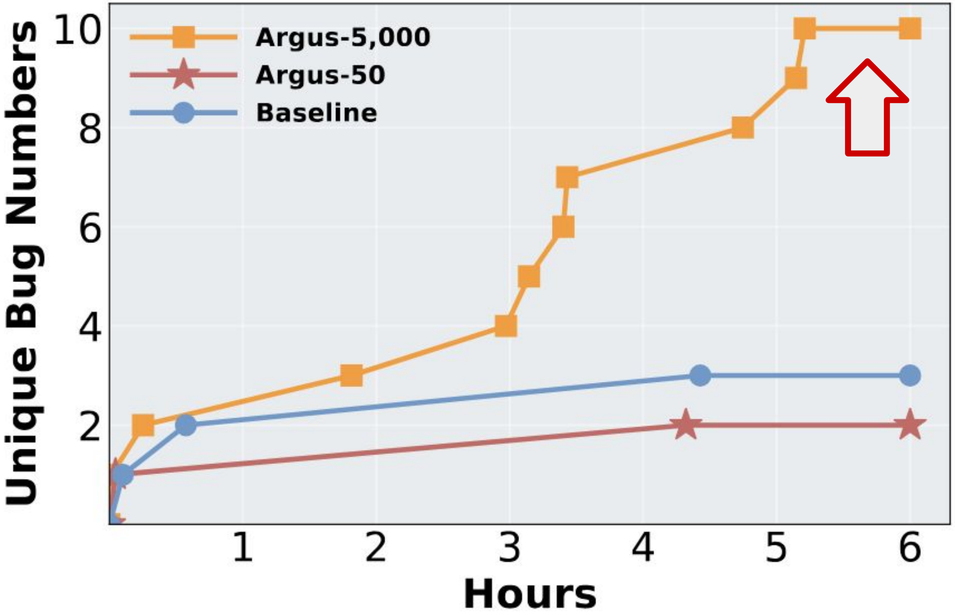
```
1. int calculate_difference(int x, int y) {
2.     if (x > y) {
3.         return x - y;
4.     } else {
5.         return y - x + 1;
6.     }
7. }
```

Metamorphic Coverage

```
1. int calculate_difference(int x, int y) {
2.     if (x > y) {
3.         return x - y;
4.     } else {
5.         return y - x + 1;
6.     }
7. }
```

Sum: Lines 3,5

E3: The Quantity of Test Oracles Matters



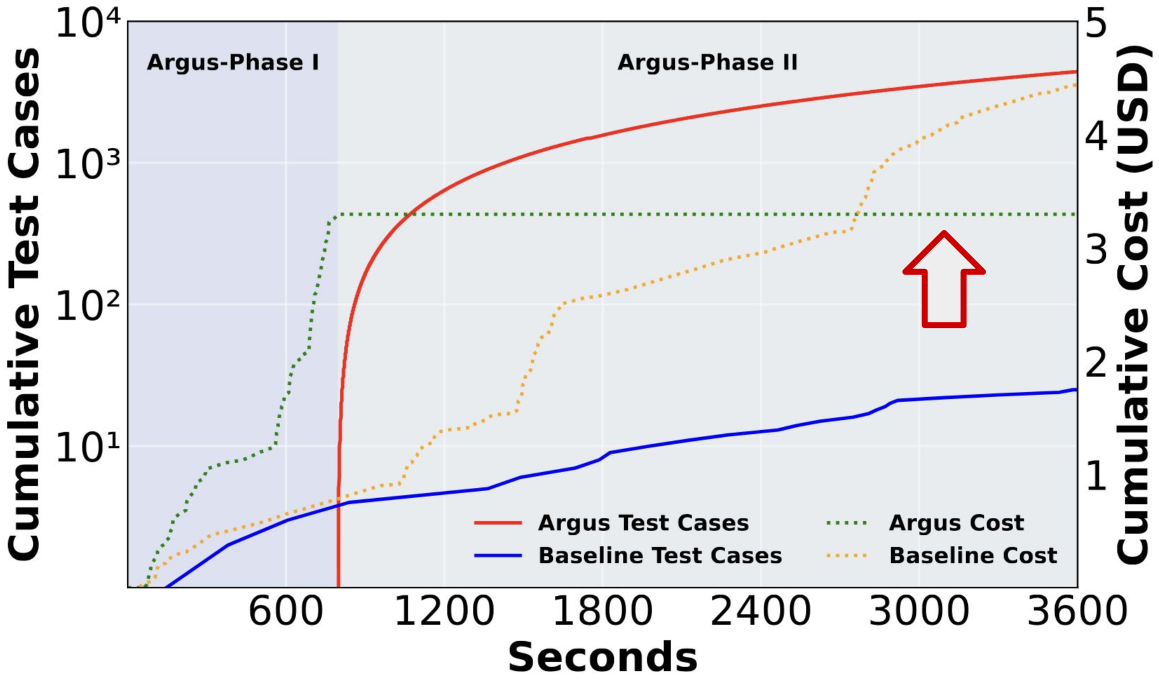
More test oracles used, more unique bugs detected within the same time window.

Baseline: 4 hand-made test oracles
TLP [OOPSLA 20], NoREC [FSE 20],
EET [OSDI 24], DQP [SIGMOD 24]

```
git bisect <subcommand> <options>
```

Using git-bisect to deduplicate unique bugs.

E4: LLM-powered DBMS Testing Is Both Efficient and Economical



After generating test oracles, we can instantiate test cases at scale with **no** additional LLM cost.

LLMs don’t just write code; they can also serve as testers that uncover deep bugs in real-world systems.

Thanks!