LLM-based Dynamic Differential Testing for Database Connectors with Reinforcement Learning-Guided Prompt Selection

(Extended Abstracts)

Ce Lyu
East China Normal University
51275903097@stu.ecnu.edu.cn

Minghao Zhao
East China Normal University
mhzhao@dase.ecnu.edu.cn

Yanhao Wang
East China Normal University
yhwang@dase.ecnu.edu.cn

Jie Liang
Beihang University
liangjie.mailbox.cn@gmail.com

ABSTRACT

Database connectors are critical components enabling applications to interact with underlying database management systems (DBMS), yet their security vulnerabilities often remain overlooked. Unlike traditional software defects, connector vulnerabilities exhibit subtle behavioral patterns and are inherently challenging to detect. Besides, nonstandardized implementation of connectors leaves potential risks (a.k.a. unsafe implementations) but is more elusive. As a result, traditional fuzzing methods are incapable of finding such vulnerabilities. Even for LLM-enable test case generation, due to a lack of domain knowledge, they are also incapable of generating test cases that invoke all interface and internal logic of connectors.

In this paper, we propose reinforcement learning (RL)-guided LLM test-case generation for database connector testing. Specifically, to equip the LLM with sufficient and appropriate domain knowledge, a parameterized prompt template is composed which can be utilized to generate numerous prompts. Test cases are generated via LLM with a prompt, and are dynamically evaluated through differential testing across multiple connectors. The testing is iteratively conducted, with each round RL is adopted to select optimal prompt based on prior-round behavioral feedback, so as to maximize control flow coverage. We implement aforementioned methodology in a practical tool and evaluate it on two widely used JDBC connectors: MySQL Connector/J and OceanBase Connector/J. In total, we reported 16 bugs, among them 10 are officially confirmed and the rest are acknowledged as unsafe implementations.

VLDB Workshop Reference Format:
Ce Lyu, Minghao Zhao, Yanhao Wang, and Jie Liang. LLM-based Dynamic Differential Testing for Database Connectors with Reinforcement Learning-Guided Prompt Selection. VLDB 2025 Workshop: AIDB.

VLDB Workshop Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Manuel-Neuer1/Bug_Issue_Link.

Proceedings of the VLDB Endowment. ISSN 2150-8097.

1 INTRODUCTION

Database connectors, also known as database drivers, serve as crucial middleware that provides standardized interfaces for application-database interactions. These components translate application API calls into native database commands while transforming query results into application-processable formats. Although this abstraction layer significantly enhances development efficiency, any inherent defects in these connectors may propagate system-wide failures. Thus, it is essential to ensure their reliability and correctness.

Unfortunately, detecting vulnerabilities of database connectors is challenging. Unlike traditional software defects, connector vulnerabilities exhibit subtle behavioral patterns. As a result, traditional fuzzing techniques exhibit limited effectiveness in testing database connectors due to their inability to handle protocol-specific syntax and stateful interactions. Notably, while these fuzzing techniques have proven highly successful for DBMS testing, their effectiveness remains constrained for connectors [2–5, 8, 13, 14, 17]. This is because existing fuzzers primarily generate equivalent SQL queries, whereas the connectors bypass rather than execute them. Moreover, the *static* nature of conventional fuzzing renders it ineffective for comprehensively testing connector logic – the generated queries typically exercise only a limited subset of interfaces and achieve insufficient branch coverage.

What is even worse, certain connector vulnerabilities are scenario-specific or originate from flawed implementation strategies, making such bugs significantly harder to detect. For example, applications often migrate data from one DBMS to another, relying on compatible connectors to handle differences in protocols and SQL dialects. When migrating, differences in connector implementations (even for supposedly compatible ones) can trigger exceptions or exhibit inconsistent behavior on additional connectors, such as data loss, silent rollbacks, or duplicate inserts. These subtle deviations may not generate error messages directly, and thus are difficult to detect.

connectors are typically expected to comply with standardized interface specifications, such as ODBC (Open Database Connectivity) and JDBC (Java Database Connectivity). However, in practice, some database vendors deviate from – or even entirely disregard – these standards, *e.g.,* due to compatibility requirements with legacy DBMS. Such nonstandardized implementation of connectors leaves potential risks (a.k.a. unsafe implementations) but is more elusive.

Besides, the behavior of the connector is highly sensitive to connection property options. These properties can have subtle but critical effects on query execution, transaction processing, and batch processing behavior. In addition, connector behavior is highly dependent on contextual semantics and invocation methods, further increasing the difficulty of writing test cases manually.

Recent research has demonstrated the significant potential and initial successes of large language models (LLMs) in software testing applications [6, 7, 12, 16]. However, due to a lack of domain knowledge, it is difficult for the LLMs to generate test cases that invoke all interface and internal logic of connectors. Besides, static or single prompts often fail to detect deep vulnerabilities, *esp.,* for those defects appears in data transmission among multiple DBMSes.

In this paper, we propose reinforcement learning (RL)-guided LLM test-case generation for database connector testing. Specifically, to equip the LLM with sufficient and appropriate domain knowledge, a parameterized prompt template is composed which can be utilized to generate numerous prompts. Test cases are generated via LLM with a prompt, and are dynamically evaluated through differential testing across multiple connectors. The testing is iteratively conducted, with each round RL is adopted to select an optimal prompt based on prior-round behavioral feedback, so as to maximize control flow coverage.

By focusing on historically efficient prompts, our approach enables more efficient connector test case generation and was evaluated on MySQL Connector/J and OceanBase Connector/J, where 10 have been confirmed as bugs, and 6 unsafe implementations, as summarized in Table 1. Some of these bugs have existed for decades without being fixed. These unsafe implementations do not follow the current JDBC specification [10] due to compatibility with the erroneous behavior of older versions of MySQL Connector/J.

2 METHODOLOGY

2.1 Overview

We propose an architecture as illustrated in Figure 1. The core idea is to leverage LLMs to automatically generate a diverse suite of test cases. Beginning with the *Prompt Generator*, it utilizes a structured prompt template to create a set of prompt candidates, which are designed to cover a wide range of connector behaviors (①). Next, an optimal prompt is selected from the candidate set by a RL-Guided strategy, which is then passed to an LLM to generate the test case (②). The generated test case is tested on two compatible connectors for differential testing (③). The *Comparator* then compares the results. If any inconsistencies are detected, a reward signal is sent back to the *RL Guidance* to reinforce the next iteration of prompt selection (④). Finally, we analyze the cases that have inconsistencies during differential testing, make logic simplifications, and report to the respective development teams (⑤).

Our framework introduces a structured prompt template to instruct the LLM to behave as an expert of DBMS testing. As detailed in Figure 2, the template covers four major aspects, namely role definition, dynamic context specification, task decomposition, and output requirements. This design can more effectively guide the LLM to generate complex and targeted test cases, thereby detecting bugs and unsafe implementations in the connector.

2.2 Database Connection Property

The behavior of database connectors can be significantly influenced by their connection-level property parameters, such as `allowMultiQueries` and `rewriteBatchedStatements` in JDBC. These options affect the internal optimization paths of connectors, query rewriting logic, and exception handling mechanisms. However, many existing test generators treat the JDBC URL as static, failing to explore the rich behavioral variations induced by different connection property settings. Thus, they miss a critical axis of behavioral variability.

To address this limitation, guided by domain knowledge, we design a systematic connection property module that explores a diverse set of JDBC parameters during test generation. Our goal is to maximize the behavioral surface exposed to the downstream connectors under test.

We first identify a set of predefined m JDBC parameters. Then, we define a property schema $C = \{c_1, c_2, \ldots, c_m\}$, where each c_j corresponds to a boolean or enumerated JDBC parameter. Each parameter has a well-defined domain.

Rather than exhaustively searching the space V of all possible parameter combinations, i.e.,

$$\mathcal{V} = \mathrm{Dom}(c_1) \times \mathrm{Dom}(c_2) \times \cdots \times \mathrm{Dom}(c_m), \tag{1}$$

we alternatively construct a curated set $\mathbb{S}$ of k representative property subsets, i.e.,

$$\mathbb{S} = \{S_1, S_2, \ldots, S_k\}, \quad S_i \subseteq \mathcal{V}, \tag{2}$$

where each S_i is a selected subset of connection properties, which is designed to capture different behaviors. By running tests on a subset of these connection properties, we can reveal different execution behaviors without exhaustively covering the entire property space. This approach allows for extensive and non-exhaustive testing of connector interfaces affected by properties, helping to reveal issues that are missed because of connection properties.

2.3 RL-Guided Prompt Scheduling

To maximize the effectiveness of LLM-generated JDBC test cases, we introduce an RL-guided prompt scheduling mechanism, which adaptively selects prompts based on their historical bug-finding performance. Inspired by the multi-armed bandit (MAB) formulation [15], we model each prompt template as a distinct arm and apply the Upper Confidence Bound (UCB1) algorithm [1] to guide prompt selection over multiple testing rounds.

Given a set of prompt templates, we define $\mathcal{P} = \{P_1, P_2, \ldots, P_N\}$ where $N \in \mathbb{N}^+$ denotes the total number of available prompt templates. Our objective is to adaptively identify and prioritize the prompt template (P_i) that maximizes the discovery of differential behaviors (indicative of potential bugs) in database connectors. Each prompt P_i acts as an arm in the MAB setting, with an unknown reward distribution corresponding to the likelihood that test cases generated from P_i expose connector inconsistencies. We employ the classic UCB1 algorithm to balance exploration and exploitation in prompt selection. In each round, the algorithm selects the prompt P_i that maximizes

$$\mu_i + \sqrt{\frac{2 \log R}{s_i}}, \quad i \in N. \tag{3}$$

For each prompt P_i, we maintain the following:

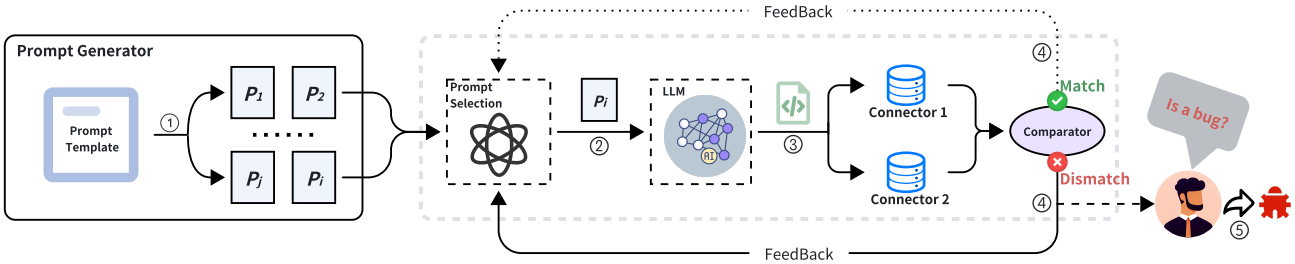

Figure 1: Overall Workflow of LLM-Enhanced DBMS Connector Testing with RL-Guided Prompt Scheduling.

1. Role Definition (Instruction to LLM)
"You are an expert in Java, JDBC, and database connector testing, tasked with generating code to expose subtle behavioral differences and non-standard implementations."

2. Context Specification (Dynamically Provided Inputs)
- *RL-Guided Test Focus*
  - |__Database Connection URL (including specific properties)
  - |__Database Schema: Relevant table structures
  - |__Prioritized Advanced JDBC Operations
  - |__Targeted Exception/Boundary Scenarios

3. Task Decomposition & Design Directives (Instructions for Test Case Generation)
- *Overarching Generation Philosophy*
  "Generate a "chaotic yet complete" Java test case simulating complex, interwoven, and sometimes unconventional real-world JDBC usage patterns."
- *Code Style & Structural Requirements*
- *Semantic & Behavioral Structure*
- *JDBC API Usage*
- *Test Scenario*

4. Output Format & Execution Requirements (Instructions for final generated case)
- *Code Characteristics*
- *Observability for Differential Testing*
- *Clarity and Conciseness*

Figure 2: Prompt template for DBMS connector testing.

- s_i: the number of times P_i has been selected;
- μ_i: the empirical mean number of output inconsistencies detected by test cases generated from P_i;
- R: the total number of round iterations so far.

After generating a test case using the selected prompt, the test is automatically rewritten, compiled, executed, and compared across different JDBC database connectors. The number of observed behavioral discrepancies serves as a reward signal to update s_i and μ_i. This closes the feedback loop between prompt selection and the identification of potentially problematic test scenarios, allowing learning-based scheduling to focus on high-impact prompts.

3 PRELIMINARY EXPERIMENTS

To evaluate the effectiveness of our method in discovering bugs and unsafe implementations on database connectors, we answer the following question: *How does our method perform on real-world database connectors?*

Tested Database Connectors. We tested two widely used JDBC connectors, namely MySQL Connector/J [11] and OceanBase Connector/J [9]. For MySQL Connector/J, we used version 9.2.0 with MySQL 8.0.36. For OceanBase Connector/J, we used OceanBase Client 4.2.0 with 5.7.25-OceanBase_CE-v4.2.1.10. We use Qwen-plus as the LLM for database connector test case generation.

Table 1: Number of Bugs and Unsafe Implementations.

Type	Database Connector	Quantity
Bugs	MySQL / OceanBase	7 / 3
Unsafe Implementations	OceanBase	6
Total (Bugs + Unsafe Implementations)		16

Confirmed Bugs & Unsafe Implementations. Table 1 shows the statistics of our results: seven bugs in MySQL and there bugs and six unsafe implementations in OceanBase. We briefly describe what triggers each issue and the description in Table 2. OceanBase development team provided valuable insight into the emergence of unsafe implementations. According to an official email response from the OceanBase development team, *"objdbc does not report errors because it is compatible with the erroneous behavior of MySQL-jdbc 5.x. It will be compatible with 8.x in the future"*. Surprisingly, Issue 3 remains unfixed in MySQL Connector/J for 17 years!

Case Study. To concisely demonstrate the detected results, we present two representative case studies: an unsafe implementation in OceanBase Connector/J that deviates from the JDBC specification and a bug in MySQL Connector/J caused by the connection property. We simplified the logic of test cases to highlight the core issues that trigger errors.

Case 1: Listing 1 illustrates an inconsistency between the MySQL and OceanBase connectors. When invoking `beforeFirst()` on a `ResultSet` created with `TYPE_FORWARD_ONLY`, MySQL correctly adheres to the JDBC specification [10] by triggering a `SQLException`, while OceanBase executes the same call without exception. This non-standard implementation may introduce security risks to systems designed to maintain portability between databases.

Listing 1: MySQL vs. OceanBase: Inconsistent `beforeFirst()`

```
con = DriverManager.getConnection(url);
stmt = con.createStatement(ResultSet.TYPE_FORWARD_ONLY);
stmt.executeUpdate("CREATE TABLE t0 (Id INT);");
rs = stmt.executeQuery("SELECT Id FROM t0 WHERE Id > 0");
rs.beforeFirst(); 💣
// MySQL triggers SQLException.
// OceanBase successfully execute.
```

Case 2: As shown in Listing 2, which attempts to insert duplicate primary keys (1), (1), (2) into table $t0$, we find an inconsistency bug. When the connection property `allowMultiQueries` is enabled, the

Table 2: Summary of bugs and unsafe implementations in MySQL and OceanBase Connector/J.

ID	Type	Database	Key Aspect / Trigger	Description
Issue 1	🐞	MySQL	Exception Message Error	Output message error when using `setMaxRows()`
Issue 2	🐞	MySQL	Spec. Violation	Using `executeBatch()` to execute a non-DML statement returns an illegal value rather than throws an exception.
Issue 3	🐞	MySQL	API Behavior	Calling `getHoldability()` is expected to get 1, but actually throws an exception.
Issue 4	🐞	MySQL	Config. Interaction	The `rewriteBatchedStatements` connection property unexpectedly affects query results following batch inserts.
Issue 5	🐞	OceanBase	Resource Management	`ResultSet` should be closed, unexpectedly not closed in OceanBase.
Issue 6	🐞	MySQL	Config. Interaction	The `allowMultiQueries` connection property unexpectedly affects the result of `getUpdateCounts()` after batch execution.
Issue 7	🐞	MySQL	Config. Breaks Atomicity	Atomicity of batch operation is compromised by `allowMultiQueries`.
Issue 8	🐞	OceanBase	Config. Interaction	The `rewriteBatchedStatements` connection property unexpectedly affects query results following batch inserts.
Issue 9	🐞	OceanBase	Spec. Violation	Using `executeBatch()` to execute a non-DML statement returns an illegal value rather than throws an exception.
Issue 10	🐞	MySQL	API Behavior	When sets `resultSetHoldability` to 2, `getResultSetHoldability()` unexpectedly returns 1.
Issue 11	💣	OceanBase	non-standard	
Issue 12	💣	OceanBase	non standard	OceanBase compatibility with erroneous behaviors of MySQL JDBC 5.x: `previous()`, `first()`, `afterLast()`, `absolute()`, `last()`, and `beforeFirst()` do not conform to JDBC documentation [10], which requires throwing `SQLException` when called on a `TYPE_FORWARD_ONLY ResultSet`.
Issue 13	💣	OceanBase	non-standard	
Issue 14	💣	OceanBase	non-standard	
Issue 15	💣	OceanBase	non-standard	
Issue 16	💣	OceanBase	non-standard	

[a] 🐞: Confirmed Bug. [b] 💣: Acknowledged Unsafe Implementation. [c] Config.: Configuration. [d] Spec.: specification.

atomicity of the batch is compromised. Ideally, the results of the batch operations for primary key conflicts should be consistent, independent of `allowMultiQueries`. The MySQL development team has also replied: *"Regardless what the documentation says about the connection property allowMultiQueries, it does affect batched statements"*.

Listing 2: MySQL: Batch Bug with allowMultiQueries Setting

```
con = DriverManager.getConnection(url);
stmt = con.createStatement();
stmt.execute("CREATE TABLE t0 (Id INT PRIMARY KEY);");
stmt.addBatch("INSERT INTO t0 VALUES(1);");
stmt.addBatch("INSERT INTO t0 VALUES(1);");
stmt.addBatch("INSERT INTO t0 VALUES(2);");
stmt.executeBatch(); 🐞
print(); // Assuming prints content of t0
// When allowMultiQueries=true -> print: 1
// When allowMultiQueries=false -> print: 1 2
```

In summary, both cases confirm that our method can detect bugs and unsafe implementations in database connectors.

4 CONCLUSION AND FUTURE WORK

In this paper, we studied the problem of testing database connectors. We proposed a novel framework for this problem based on LLMs with an RL-guided prompt scheduling strategy and identified 10 bugs and 6 unsafe implementations in Oceanbase and MySQL connectors. As an early-stage study, we aim to answer a central question: *Can our framework find meaningful vulnerabilities in real-world scenarios?* The results presented a strong affirmative. In the future, a more comprehensive performance review, baseline comparison, and a deeper analysis of the discovered vulnerabilities will be developed as a more in-depth study.

REFERENCES

[1] Djallel Bouneffouf. 2016. Finite-time analysis of the multi-armed bandit problem with known trend. In *CEC*. 2543–2549.

[2] Ziyu Cui, Wensheng Dou, Yu Gao, Rui Yang, Yingying Zheng, Jiansen Song, Yuan Feng, and Jun Wei. 2025. Simple Testing Can Expose Most Critical Transaction Bugs: Understanding and Detecting Write-Specific Serializability Violations in Database Systems. *Proc. VLDB Endow.* 18, 8 (2025).

[3] Wenqian Deng, Jie Liang, Zhiyong Wu, Jingzhou Fu, Mingzhe Wang, and Yu Jiang. 2024. Coni: Detecting Database Connector Bugs via State-Aware Test Case Generation. In *ICSE*. 26–37.

[4] Jingzhou Fu, Jie Liang, Zhiyong Wu, Mingzhe Wang, and Yu Jiang. 2022. Griffin: Grammar-free DBMS fuzzing. In *ASE*. 1–12.

[5] Jingzhou Fu, Jie Liang, Zhiyong Wu, Yanyang Zhao, Shanshan Li, and Yu Jiang. 2025. Understanding and Detecting SQL Function Bugs: Using Simple Boundary Arguments to Trigger Hundreds of DBMS Bugs. In *EuroSys*. 1061–1076.

[6] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large language models for software engineering: A systematic literature review. *ACM Trans. Softw. Eng. Methodol.* 33, 8 (2024), 1–79.

[7] Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A survey on large language models for code generation. *arXiv:2406.00515* (2024).

[8] Zu-Ming Jiang, Jia-Ju Bai, and Zhendong Su. 2023. DynSQL: Stateful Fuzzing for Database Management Systems with Complex and Valid SQL Query Generation. In *USENIX Security*. 4949–4965.

[9] OceanBase Team. 2025. OceanBase Connector/J. https://github.com/oceanbase/obconnector-j.

[10] Oracle. 2014. ResultSet.beforeFirst() Method. https://docs.oracle.com/javase/8/docs/api/java/sql/ResultSet.html#beforeFirst--.

[11] Oracle. 2025. MySQL Connector/J. https://github.com/mysql/mysql-connector-j.

[12] Fei Qi, Yingnan Hou, Ning Lin, Shanshan Bao, and Nuo Xu. 2024. A Survey of Testing Techniques Based on Large Language Models. In *ICCMT*. 280–284.

[13] Manuel Rigger and Zhendong Su. 2020. Testing database engines via pivoted query synthesis. In *OSDI*. 667–682.

[14] Jiansen Song, Wensheng Dou, Yingying Zheng, Yu Gao, Ziyu Cui, Wei Wang, and Jun Wei. 2025. Detecting Schema-Related Logic Bugs in Relational DBMSs via Equivalent Database Construction. *Proc. VLDB Endow.* 18, 7 (2025), 2281–2294.

[15] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement learning - an introduction, 2nd Edition*. MIT Press.

[16] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. 2024. Software Testing With Large Language Models: Survey, Landscape, and Vision. *IEEE Trans. Softw. Eng.* 50, 04 (2024), 911–936.

[17] Rui Zhong, Yongheng Chen, Hong Hu, Hangfan Zhang, Wenke Lee, and Dinghao Wu. 2020. Squirrel: Testing database management systems with language validity and coverage feedback. In *CCS*. 955–970.