计 算 机 研 究 与 发 展
Journal of Computer Research and Development
https://crad.ict.ac.cn
63(4): 884−899, 2026

基于大语言模型的数据库管理系统模糊测试方法

张瑞宸　张亚东　兰文尉　岳　清　王正武　崔展齐
（北京信息科技大学计算机学院　北京　100192）
（zrcl.richard_zhang@bistu.edu.cn）

Large Language Model Based Fuzz Testing Approach for Database Management System

Zhang Ruichen, Zhang Yadong, Lan Wenwei, Yue Qing, Wang Zhengwu, and Cui Zhanqi
（*School of Computer Science*, *Beijing Information Science and Technology University*, *Beijing* 100192）

Abstract　Database management systems (DBMSs), as fundamental software for data management and storage, are critical to ensuring the security, reliability and stability of modern data-intensive applications. In recent years, fuzz testing has been increasingly adopted for DBMS validation owing to its low manual cost, high efficiency, and capability of automatically exercising diverse execution paths. However, existing DBMS fuzzing approaches remain constrained by insufficient test case coverage and limited adaptability across heterogeneous DBMS implementations, which substantially weakens their effectiveness and generality. CLCC (curated LLM case construct) is a novel test case generation approach for DBMS fuzzing based on a large language model (LLM). In CLCC, LLMs are employed to construct high-quality initial seeds prior to fuzzing, while edge coverage-guided seed selection is incorporated during fuzzing to steer LLM-driven test case generation. Extensive comparative experiments demonstrate that CLCC achieves 14.96%−49.31% higher edge coverage on SQLite, MySQL, MariaDB, DuckDB and PostgreSQL compared with SQUIRREL, delivers 6.09%−17.10% improvements on SQLite, MySQL and PostgreSQL relative to SQLRight, and provides 17.95%−41.20% gains on SQLite, MySQL and MariaDB over ParserFuzz.

Key words　database management system; fuzz testing; large language model; vulnerability detection; coverage analysis

摘　要　数据库管理系统（database management system, DBMS）作为数据管理与存储的关键软件，其可靠性直接影响数据密集型系统的安全稳定运行。近年来，模糊测试因其具备人工成本低、测试效率高等特点，成为DBMS测试方法之一。然而现有的DBMS模糊测试方法面临测试用例覆盖能力不足与多DBMS适配性差两大问题，限制了测试的效果和泛用性。为此，CLCC（curated LLM case construct）是一种基于大语言模型（large language model, LLM）的DBMS模糊测试方法。该方法在模糊测试前，利用LLM对初始种子进行构建，并在模糊测试过程中，根据边覆盖情况筛选种子，引导LLM生成测试用例。与SQUIRREL、SQLRight和ParserFuzz进行的对比实验表明，CLCC测试SQLite、MySQL、MariaDB、DuckDB和PostgreSQL的边覆盖数量比SQUIRREL增加了14.96%~49.31%；测试SQLite、MySQL和PostgreSQL的边覆盖数量比SQLRight增加了6.09%~17.10%；测试SQLite、MySQL和MariaDB的边覆盖数量比ParserFuzz增加了17.95%~41.20%。

关键词　数据库管理系统；模糊测试；大语言模型；漏洞检测；覆盖率分析

收稿日期：2025-09-21；**修回日期**：2025-12-24

基金项目：江苏省前沿引领技术基础研究专项（BK20202001）；北京信息科技大学“勤信人才”培育计划项目（QXTCP B202406）
This work was supported by the Leading-Edge Technology Program of Jiangsu Natural Science Foundation (BK20202001) and the Beijing Information Science and Technology University “Qin-Xin Talent” Cultivation Project (QXTCP B202406).

通信作者：崔展齐（czq@bistu.edu.cn）

中图法分类号 TP311.5

DOI: 10.7544/issn1000-1239.202550692 CSTR: 32373.14.issn1000-1239.202550692

数据库管理系统(database management system, DBMS)被广泛应用于金融、物联网、工业控制等领域[1], 因其中存储着大量重要数据, 一旦触发故障则可能造成重大损失。例如, 2025 年, 美国进出口银行的 FOIAXpress 系统因数据库崩溃导致其公众信息请求服务中断超 1 个月, 并造成数据丢失①; 2018 年, GitHub 因 MySQL 数据库集群崩溃导致其核心服务中断超过 24 小时, 对数百万开发者和企业造成了重大冲击②。为提高 DBMS 的可靠性, 研究人员从 DBMS 故障检测[2]、数据恢复[3-4]、形式化验证[5]、系统测试[6]等方面展开了相关研究。

DBMS 测试的相关研究中, 模糊测试技术因其具有人工成本低、测试效率高等优点, 受到了广泛关注。现有 DBMS 模糊测试方法如 SQUIRREL[7]、SQLsmith③、RATEL[8] 等, 通过随机生成包含多个结构化查询语言(structured query language, SQL)语句的测试用例对 DBMS 进行测试, 并将能够覆盖尚未被覆盖程序边的“有趣的”测试用例, 作为种子添加至种子池, 以引导后续测试用例生成。然而, 上述方法生成的测试用例中包含的 SQL 关键字较为单一, 导致难以全面覆盖 DBMS 的各类功能模块, 从而影响 DBMS 模糊测试的效果[9]。此外, 大多数方法难以兼容不同 DBMS, 限制了模糊测试的适用范围[10]。

如何确保生成的测试用例能够覆盖各类功能模块, 且能够通用于多种 DBMS, 仍面临着大量挑战。首先是由于状态空间庞大带来的组合爆炸问题。DBMS 通常需要支持大量 SQL 关键字。例如, SQLite 中有 147 种 SQL 关键字, 其中 SELECT、DELETE 等 6 种 SQL 关键字支持嵌套结构④。测试用例生成过程中, 考虑多种 SQL 关键字的组合和嵌套关系将导致计算成本大幅增加, 也使得有限的测试资源难以覆盖关键路径, 影响模糊测试的效果。其次是 SQL 方言带来的兼容性挑战[11]。不同 DBMS 在 SQL 语法细节、支持的 SQL 关键字和函数实现等方面存在差异, 形成了独特的 SQL 方言。DBMS 模糊测试工具通常需要针对 SQL 方言实现专门的适配逻辑, 以确保生成的测试用例能够被目标 DBMS 正确执行。以 SQUIRREL 为例, 为兼容 MariaDB 编写了超过 9 000 行代码[10]。这种人工适配 SQL 方言的方式不仅开发成本高、周期长, 还影响了测试工具的扩展性, 限制了其对多种 DBMS 进行测试的迁移能力。

为此, 本文提出一种了基于 LLM 的 DBMS 模糊测试方法 CLCC(curated LLM case construct), 以提升 DBMS 模糊测试的覆盖效果和兼容性。首先, 使用 LLM 构建适配目标 DBMS 的初始种子。其次, 分析种子池中种子的边覆盖情况以计算种子适应度。然后, 根据种子适应度挑选种子作为输入, 引导 LLM 生成测试用例。最后, 执行生成的测试用例, 并将有趣的测试用例加入种子池中。我们基于 SQUIRREL 和 Gemini-2.0-flash⑤实现了 CLCC 原型工具。实验结果表明, CLCC 测试 SQLite, MySQL, MariaDB, DuckDB 和 PostgreSQL 的边覆盖数量比 SQUIRREL 分别增加了 14.96%、26.87%、47.94%、49.31% 和 19.75%; 测试 SQLite、MySQL 和 PostgreSQL 的边覆盖数量比 SQLRight[12] 分别增加了 6.09%、17.10% 和 16.83%; 测试 SQLite、MySQL、MariaDB 的边覆盖数量比 ParserFuzz[13] 分别增加了 17.95%、24.85% 和 41.20%。

本文的主要贡献有 2 点:

1)提出了一种基于 LLM 的 DBMS 模糊测试方法 CLCC, 使用 LLM 构建初始种子, 并根据种子边覆盖信息引导 LLM 生成测试用例, 以提高模糊测试效果。

2)基于 CLCC 实现了原型工具, 在 SQLite、MySQL 等 5 个开源数据库上与 SQUIRREL、SQLRight 和 ParserFuzz 进行了对比实验, 以评估 CLCC 的有效性。

1 动机示例

当使用 SQUIRREL 工具对 SQLite 进行 2 小时的模糊测试后, 边覆盖率仅增加了 1.16%。经分析发现, 这一情况可能和模糊测试过程中生成的测试用例未

① Export-Import Bank (EXIM) Records, https://www.governmentattic.org/58docs/EXIMoutageFOIAsystemFebMar2025.pdf

② October 21 post-incident analysis, https://github.blog/news-insights/company-news/oct21-post-incident-analysis/

③ SQLsmith, https://github.com/anse1/sqlsmith

④ SQLite Keywords, https://www.sqlite.org/lang_keywords.html

⑤ Gemini-2.0-flash, https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-0-flash

能覆盖 DBMS 部分功能有关。

模糊测试过程中,覆盖 DBMS 特定功能不仅和测试用例中包含的 SQL 关键字相关,还和测试用例所包含的 SQL 语句的数量相关。单个测试用例包含的 SQL 语句数量越多,所覆盖的 SQL 关键字越丰富,通常越有可能覆盖 DBMS 的更多功能。例如,触发器(TRIGGER)是 DBMS 中用于在特定事件(如更新、插入)发生时自动执行预定义操作的功能。为执行该功能,测试用例需至少包含 5 条 SQL 语句。例如,图 1 为测试触发器功能的测试用例示例,其中,第 1~10 行为创建目标表和辅助表、第 11 行为插入初始数据、第 12~18 行为定义触发器、第 19 行为触发触发器。

```
1  CREATE TABLE employees (
2      id INTEGER PRIMARY KEY,
3      name TEXT,
4      salary INTEGER
5  );
6  CREATE TABLE salary_log (
7      emp_id INTEGER,
8      old_salary INTEGER,
9      new_salary INTEGER
10 );
11 INSERT INTO employees (id, name, salary) VALUES (1, 'Alice', 5000);
12 CREATE TRIGGER log_salary_update
13 AFTER UPDATE ON employees
14 FOR EACH ROW
15 BEGIN
16     INSERT INTO salary_log (emp_id, old_salary, new_salary)
17     VALUES (OLD.id, OLD.salary, NEW.salary);
18 END;
19 UPDATE employees SET salary = 6000 WHERE id = 1;
```

Fig. 1 An example of test case

图 1 测试用例示例

我们进一步分析了测试期间 SQUIRREL 生成的种子(共计 5 794 个)中所包含的 SQL 语句数量和 SQL 关键字覆盖情况,结果如图 2 所示。图 2 为包含不同 SQL 语句数量的种子占全部种子的比例,其中仅有 3 523 个(约占 60.8%)种子包含了 5 条及以上的 SQL 语句。SQL 关键字覆盖方面,在 SQLite 的 147 个 SQL 关键字中,生成的种子仅覆盖了 67 个(约占 45.4%)SQL 关键字,未包含其余 80 个(约占 54.6%)SQL 关键字。其中,TRIGGER、ALTER 等常见 SQL 关键字均未被生成。

为探索 SQL 关键字缺失对边覆盖数量的影响,我们对 TRIGGER 关键字进行了分析。分析发现,SQLite 程序中有 1 135 条程序边和 TRIGGER 关键字相关,占 SQLite 总边数的 2.5%。以图 3 中 SQLite 的语法解析文件代码片段为例,只有当解析到结构正确且包含 TRIGGER 关键字的 SQL 语句时才会调用 sqlite3BeginTrigger 方法。这表明,当测试用例覆盖更多类型的 SQL 关键字时,更有助于增加模糊测试的边覆盖数量。此外,DBMS 模糊测试方法通常还存在兼容性问题。例如,SQUIRREL 仅适用于对 SQLite、MySQL、MariaDB 和 PostgreSQL 进行测试,对于其他 DBMS 缺乏支持[7]。导致这一问题的原因在于不同的 DBMS 虽遵循通用的 SQL 语言规范,但通常还会引入独特的语法扩展和方言特性,模糊测试工具需要针对不同 DBMS 进行调整和适配。

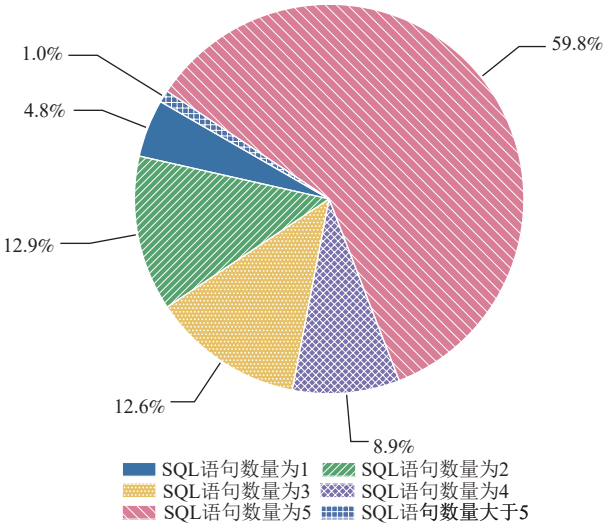


Fig. 2 SQL statement count in seeds generated by SQUIRREL

图 2 SQUIRREL 生成的种子中 SQL 语句数量

```
trigger_decl(A) ::= temp(T) TRIGGER ifnotexists(NOERR) nm(B) dbnm(Z)
                    trigger_time(C) trigger_event(D)
                    ON fullname(E) foreach_clause when_clause(G). {
  sqlite3BeginTrigger(pParse, &B, &Z, C, D.a, D.b, E, G, T, NOERR);
  ...
}
```

Fig. 3 Code snippet of the syntax parsing file parse. y in SQLite

图 3 SQLite 中语法解析文件 parse. y 的代码片段

如果要有效提升 DBMS 模糊测试的边覆盖效果,则须生成能够覆盖丰富 DBMS 功能的测试用例。然而,现有的模糊测试工具难以生成 SQL 语句数量多、SQL 关键字种类丰富的测试用例,且缺乏对不同 DBMS 进行针对性适配的方法。如能结合 LLM 对 SQL 语句的理解能力和生成能力来生成 SQL 语句数量充足、SQL 关键字种类丰富的测试用例,并针对不同 DBMS 的方言特性进行动态适配,将能有效提升 DBMS 模糊测试的边覆盖效果和适配能力。

2 基于 LLM 的数据库管理系统模糊测试(CLCC)

图 4 为 CLCC 的框架图,包括初始种子构建、适应度计算和测试用例生成 3 个阶段。在初始种子构

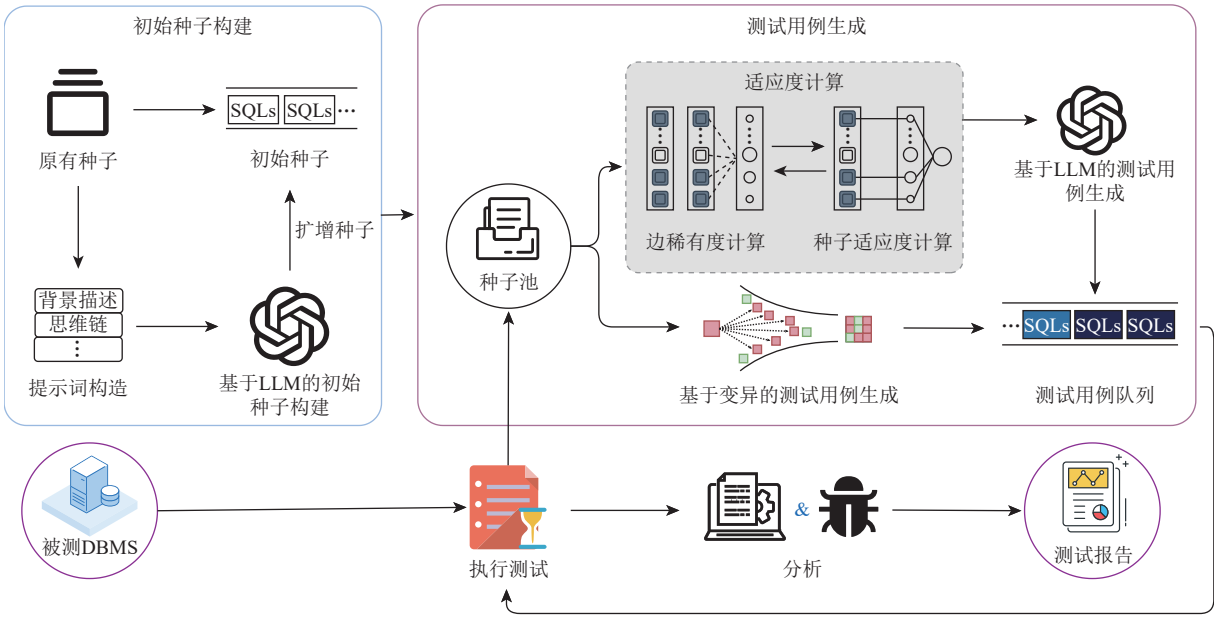


Fig. 4 LLM based DBMS fuzz testing framework

图 4 基于 LLM 的 DBMS 模糊测试框架

建阶段，CLCC 使用 LLM 对初始种子进行构建。在适应度计算阶段，CLCC 分析种子池中每个种子的边覆盖情况，据此计算种子的适应度。在测试用例生成阶段，根据适应度挑选种子作为提示引导 LLM 生成测试用例。

2.1 初始种子构建

基于变异的 DBMS 模糊测试方法通常对初始种子进行持续变异以生成新的测试用例。初始种子中包含的 SQL 关键字种类越丰富，越有可能覆盖更多的 DBMS 功能模块。因此，CLCC 引入 LLM 对初始种子进行构建，生成包含丰富 SQL 关键字的测试用例，以提升覆盖效果。

初始种子构建时，首先，对于缺少原有种子的情况，CLCC 直接通过零样本方式构建 LLM 提示；对于存在原有种子的情况，CLCC 在构建 LLM 提示时将其作为组成部分之一。然后，CLCC 将构建的提示作为 LLM 的输入以扩增初始种子。最后，将原有种子和扩增种子加入到种子池中，作为后续模糊测试的初始输入。

CLCC 构建初始种子的提示模板如表 1 所示。提示模板中包括 5 个组成部分：背景描述、思维链（chain of thought, CoT）、目标描述、输出格式和样本参考。其中，背景描述部分用于确定 LLM 的任务目标，即生成用于 DBMS 模糊测试的初始种子。思维链部分，包括保证语法和语义正确性、明确 SQL 语句条数和包含的 SQL 关键字、适配目标 DBMS 特征、参考原有种子和正确性检查 5 个步骤，以提升生成初始种子的逻辑性和有效性。目标描述部分，指定被测 DBMS 的名称及版本信息，用于引导 LLM 生成符合被测 DBMS 语法语义的初始种子。输出格式部分，规定 LLM 输出扩增种子的格式，为后续自动化处理提供基础。样本参考部分，对于存在原有种子的情况，该部分为 LLM 提供初始种子样本；对于没有原有种子的情况，该部分可被忽略。

2.2 适应度计算

将种子作为 LLM 提示的组成部分能够提升其生成测试用例的有效性，若直接将全部种子作为提示输入，将产生大量 LLM 调用成本。为控制成本，CLCC 从种子池中筛选部分种子作为提示。筛选种子的依据是基于种子覆盖信息所计算的适应度。

为计算种子适应度，需要先对程序边的稀有度进行分析。边的稀有度表示其被种子覆盖的频繁程度，被覆盖次数越少则稀有度越高。对于由m条程序边组成的 DBMS，其中第i条边被n个种子覆盖次数的统计方法如式(1)所示。

$$B_i^n = \sum_{j=1}^{n} b_i^j \text{。} \tag{1}$$

边覆盖情况可通过 afl-showmap[①]工具获取，式(1)

① afl-showmap, https://github.com/AFLplusplus/AFLplusplus/blob/stable/src/afl-showmap.c

Table 1 Prompt Template

表 1 提示词模板

提示词类型	提示词	I	S	T
背景描述	You need to generate high-quality [initial seeds, test case] for DBMS fuzz testing. [An initial seed, A testcase] consists of multiple SQL statements and only contain SQL statements.	√		√
思维链	Follow the steps below to reason step by step and generate the [initial seeds, test case]:	√		√
	Step 1: Grammatical and Semantic Correctness Ensure that all SQL statements in the [initial seeds, test case] conform to the syntax rules and semantic logic of the target database.	√		√
	Step 2: SQL Quality Assurance The [initial seeds, test case] should contain sufficient number of SQL statements, with rich usage of various SQL keywords.	√		√
	Step 3: Target-Specific SQL Features Incorporate SQL statements that use database-specific keywords or features based on the target DBMS to make the [initial seeds, test case] tailored and effective.	√		√
	Step 4: Use Sample as Reference Refer to the provided sample initial seed in the sample section to understand the required format.	√	√	
	Step 4: Use Sample as Reference The sample section provides a curated test case that covers a broader range of program edge. You are encouraged to mutate the existing SQL statements in the sample and incrementally add new SQL statements, aiming to cover additional program edges in the target DBMS.			√
	Step 5: Completeness Check After generation, perform a thorough validation of the seed: If any SQL syntax is incorrect for the target DBMS, it must be fixed. If there are semantic issues (e. g. a statement uses a table that hasn't been created), insert necessary preorder statements in the seed in a logically consistent order.	√		√
目标描述	Generate one high-quality and well-structured [initial seed, test case] for the {target DBMS name} version in {target DBMS version}.	√		√
输出格式	Wrap generated [initial seeds, test case] using the following format: '\n"'sql\n (generated [initial seeds, test case]) \n"'\n'	√		√
样本参考	A sample of [initial seeds, test case]	√	√	√

注：I：初始种子构建时必须使用的部分；S：可选择使用的部分；T：测试用例生成时必须使用的部分；[]：选择括号内的提示词进行填充；{}：根据被测 DBMS 的信息进行填充。

中，b_i^j为第j个种子对第i条边的覆盖情况(若未覆盖则b_i^j=0，反之b_i^j=1)。第j个种子对 DBMS 中所有程序边的覆盖情况可表示为如式(2)所示的向量。

$$\boldsymbol{c}^j=(b_1^j,b_2^j,\cdots,b_m^j)\text{。} \tag{2}$$

在完成对所有边的覆盖总次数的统计后，可进一步计算边的稀有度，其中第i条边在第n个种子执行结束时的稀有度计算方法如式(3)所示。

$$r_i^n=\frac{1}{\sqrt{m}}\lg\left(\frac{u^n}{1+B_i^n}\right)\text{。} \tag{3}$$

式(3)中，u^n为被当前已执行的n个种子所覆盖的边数量。在执行第n个种子后，被测 DBMS 中所有程序边的稀有度可表示为如式(4)所示的向量。

$$\boldsymbol{rv}^n=(r_1^n,r_2^n,\cdots,r_m^n)\text{。} \tag{4}$$

在测试开始前，CLCC 对$\boldsymbol{rv}^0$进行初始化，所有边的稀有度均设置为 0。

在得到第n个种子的边覆盖向量和被测 DBMS 中边的稀有度向量后，可用式(5)计算该种子的适应度。

$$fit^n=\boldsymbol{c}^n\cdot\boldsymbol{rv}^{n-1}\text{。} \tag{5}$$

此外，为消除测试不同 DBMS 时适应度取值范围的差异，我们对种子适应度进行归一化，归一化后的适应度记为$\widetilde{fit}^n$。

2.3 测试用例生成

相比基于变异的测试用例生成方式，调用 LLM 生成测试用例速度较慢，将影响模糊测试的效率。因此，CLCC 同时使用基于变异的测试用例生成方式和基于 LLM 的测试用例生成方式。

其中，基于变异的测试用例生成方式(如 SQUIRREL)能够有效结合边覆盖信息对种子进行变异，快速生成测试用例，可弥补基于 LLM 的测试用例生成方式生成速度的不足；而基于 LLM 的测试用例生成方法能够生成 SQL 语句数量更多、SQL 关键字更丰富的测试用例，可弥补基于变异的测试用例生成方式所生成的测试用例边覆盖低的缺点。

CLCC 将上述 2 种方式生成的测试用例存入统一的待执行测试用例队列中，并进行优先级调度。其中，由于 LLM 生成测试用例的速度相对较慢但生成的测试用例覆盖潜力较高，因此其放入待执行测试用例队首，以确保被优先执行。基于变异的方式生成的测试用例则依次加入队尾。

基于 LLM 的测试用例生成方式中，CLCC 同时使用了选择性 LLM 测试用例生成和持续性 LLM 测试用例生成 2 种策略。下面将详细介绍这 2 种策略以及它们的优势。

1)选择性 LLM 测试用例生成。为使高适应度的种子能够及时引导 LLM 生成测试用例，CLCC 将归一化的种子适应度和预设的阈值θ相比较，并据此判

断是否使用该模块，计算方法如式(6)所示。

$$act=\begin{cases}1, & \widetilde{fit}^{n}\geqslant\theta,\\ 0, & \widetilde{fit}^{n}<\theta。\end{cases}\tag{6}$$

式(6)中θ为预设的阈值。当act=1时，该策略将第n个种子作为提示的组成部分来生成测试用例。此外，CLCC采用多线程的方式向LLM发送提示，以提升使用LLM生成测试用例的效率。

2)持续性LLM测试用例生成。随着模糊测试的进行，超过阈值θ的归一化的种子适应度逐渐降低，因而能够被选择性LLM测试用例生成策略选中的种子随之减少。为持续执行由LLM生成的测试用例，CLCC还使用了持续性LLM测试用例生成策略。首先，从已执行的n个种子中选择第$n-w+1$到第n个种子(w为窗口大小)中适应度最高的作为提示词组成部分；然后，将构建的提示词作为LLM的输入生成测试用例，并加入待执行测试用例队列。

3 实验设计

为评估CLCC的有效性，提出以下4个研究问题。

RQ1：与其他数据库模糊测试工具相比，CLCC的表现如何？

动机：为分析CLCC与其他基准方法相比对DBMS进行测试的效果是否取得了有效提升，我们选择SQUIRREL、SQLRIGHT和ParserFuzz作为对比，以被测DBMS的边覆盖数量作为评价指标。

RQ2：扩增种子是否可以提升测试DBMS的效果？

动机：在初始种子构建阶段，CLCC基于原有种子构建扩增种子，为评估扩增种子在模糊测试中的影响，我们将原有种子和扩增种子进行不同比例的混合作为初始种子，对模糊测试的效果进行对比。

RQ3：选择性LLM测试用例生成策略中使用不同阈值筛选种子对CLCC的影响如何？

动机：选择性LLM测试用例生成策略中CLCC将种子适应度和阈值θ相比较以选择测试用例。为分析不同阈值对CLCC的影响如何，我们设置了不同的阈值进行实验，并对边覆盖数量差异进行对比。

RQ4：不同的LLM对CLCC的影响如何？

动机：CLCC使用LLM生成测试用例，为评估不同LLM对CLCC的性能影响，我们选择gpt-3.5-turbo、claude-2.0-haiku等5个不同的LLM分别进行实验，并分析了LLM对边覆盖数量的影响。

3.1 基 线

我们选择SQUIRREL[7]、SQLRight[12]和ParserFuzz[13]作为基线方法进行对比。其中，SQUIRREL是基于覆盖引导的DBMS模糊测试工具，它通过中间表示(intermediate representation, IR)结构保存和管理SQL语句，并基于语义分析生成测试用例；SQLRight通过构造功能等价的SQL语句并比较其执行结果，以发现DBMS中存在的逻辑错误，并利用覆盖率信息引导测试用例生成，从而提升模糊测试的覆盖效果；ParserFuzz通过DBMS的解析器中提取语法语义规则，并基于此生成语法语义正确的SQL语句，提升其生成测试用例的有效性。

SEDAR[14]、GRIFFIN[10]、QPG[15]、LEGO[9]、Wing-Fuzz[11]等方法没有开源其原型工具，BUZZBEE[16]由于其中Bazel模块更新导致无法部署，因此上述工具无法作为基线进行比较。

3.2 被测DBMS

为验证CLCC测试不同DBMS的性能，我们选取了SQUIRREL[7]和SEDAR[14]使用的8个被测DBMS中的SQLite、MariaDB、MySQL、DuckDB和PostgreSQL作为被测DBMS。实验中被测DBMS的版本为：SQLite v3.49.1、MySQL v8.0.41、MariaDB v10.11.1、DuckDB v1.2.2和PostgreSQL v17.6。

选择SQLite、MariaDB、MySQL以及PostgreSQL的原因在于这些数据库均能被基线工具SQUIRREL所适配。而SEDAR使用了DuckDB, MonetDB, Virtuoso-opensource以及ClickHouse作为被测DBMS。我们仅选择了其中的DuckDB作为被测DBMS的原因在于其他DBMS均采用客户端/服务器架构，即数据库引擎作为独立服务运行。相比于嵌入式DBMS(如SQLite和DuckDB)可编译后直接运行，SQUIRREL在适配对客户端/服务器架构的DBMS时需要为其实现语义规则和客户端，因此难以将SQUIRREL适配于这些数据库。

选择的5个被测DBMS中，由于SQLRight不适配MariaDB和DuckDB以及ParserFuzz不适配DuckDB和PostgreSQL，且将上述工具进行修改以适配其他DBMS较为困难，故我们的实验中未使用SQLRight对MariaDB和DuckDB进行测试且未使用ParserFuzz对DuckDB和PostgreSQL进行测试。

3.3 初始种子

在初始种子构建阶段，对于SQLite、MySQL、MariaDB和PostgreSQL,CLCC以SQUIRREL、SQLRight和ParserFuzz的原有种子为基础，生成同样数量的扩增种子。尽管SQUIRREL、SQLRight和ParserFuzz未为DuckDB提供原有种子，但其SQL方言是基于

PostgreSQL 设计的[①], 因此在实验中我们直接使用 SQUIRREL 和 SQLRight 为 PostgreSQL 提供的初始种子作为 DuckDB 的原有种子, 并为其生成数量相同的扩增种子。表 2 为实验中使用的原有种子和扩增种子的基本信息。

Table 2 Basic Information of Initial Seeds

表 2 初始种子的基本信息

DBMS	初始种子信息	原有种子	扩增种子
SQLite	种子个数	146	146
	SQL 语句数	1 009	2 416
	总文件大小/B	49 411	305 598
MySQL	种子个数	110	110
	SQL 语句数	794	2 281
	总文件大小/B	59 695	313 296
MariaDB	种子个数	110	110
	SQL 语句数	794	2 226
	总文件大小/B	59 695	258 473
DuckDB	种子个数	320	320
	SQL 语句数	28 461	10 507
	总文件大小/B	2 673 817	1 145 479
PostgreSQL	种子个数	320	320
	SQL 语句数	28 461	13 258
	总文件大小/B	2 673 817	1 199 139

3.4 实验设置

实验在拥有 24 个内核的 CPU(INTEL i9-14900K @2.4 GHz)和 96 GB 的内存、运行 64 位 Windows 11 的计算机上通过 Docker 进行。被测 DBMS 由 AFL++[17] 插桩编译以获得边覆盖反馈。为定量比较, 我们为每个 Docker 容器分配 2 个 CPU 内核和 24 GB 内存。LLM 默认使用 Gemini-2.0-flash 来构建初始种子和生成测试用例, 阈值θ默认设置为 0.1, 初始种子默认同时使用全部的原有种子和扩增种子, LLM 调用时温度(temperature)设置为 0.7、max tokens 为各 LLM 的默认值。实验中实验时间的选择参考了 P-FUZZ[18] 的设置, 每个实验运行 2 小时并重复 3 次。

4 实验结果及评价

1)对于 RQ1 的回答。我们将 CLCC 与 SQUIRREL、SQLRight 和 ParserFuzz 在 SQLite、MySQL、MariaDB、DuckDB 和 PostgreSQL 上进行了对比实验, 并同时使用原有种子和扩增种子作为初始种子, 实验结果如图 5 所示。

图 5 可以看到, 经过 2 小时的测试, CLCC 比其他测试工具覆盖了更多程序边。与 SQUIRREL 相比, CLCC 测试 SQLite、MySQL、MariaDB、DuckDB 和 PostgreSQL 的边覆盖数量分别增加了 14.96%、26.87%、47.94%、49.31% 和 19.75%。与 SQLRight 相比, CLCC 测试 SQLite、MySQL 和 PostgreSQL 的边覆盖数量分别增加了 6.09%、17.10% 和 16.83%。与 ParserFuzz 相比, CLCC 对 SQLite、MySQL 和 MariaDB 进行测试的边覆盖数量分别增加了 17.95%、24.85%、41.20%。

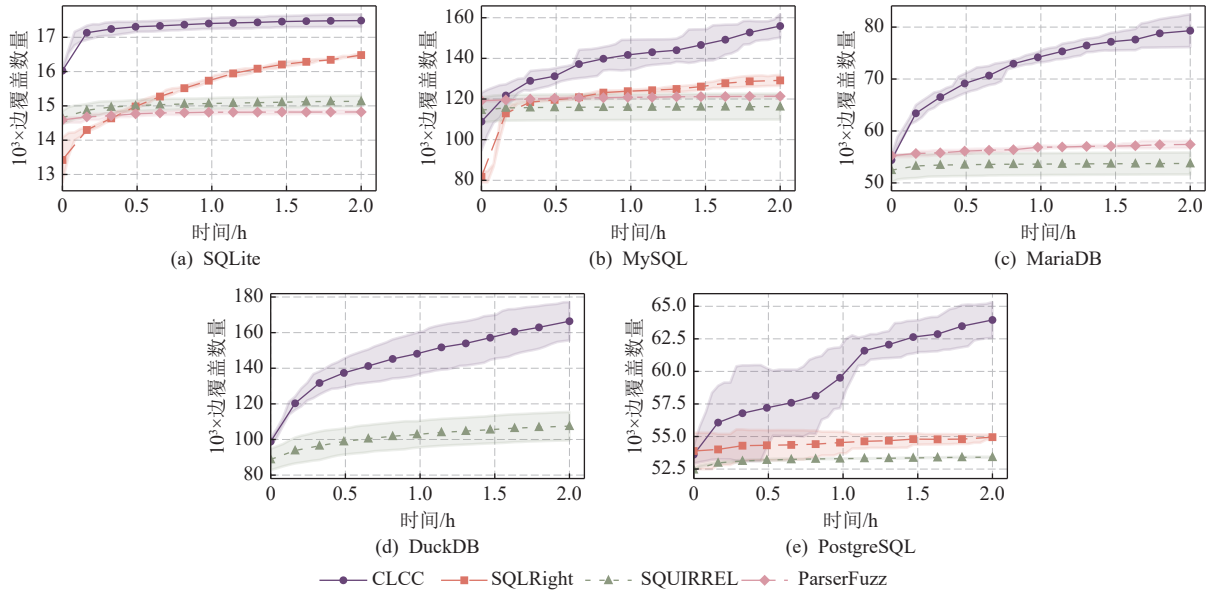
对于崩溃触发能力, 我们对 CLCC、SQUIRREL、SQLRight 和 ParserFuzz 所发现的崩溃进行了人工分析。结果显示, 以上工具均未触发任何真实的崩溃错误。这一现象可能源于 2 方面原因: 首先, 实验中选择的被测 DBMS 均为最新版本, 其稳定性较高; 其次, 由于实验成本限制, 测试持续时间相对较短, 因此未能充分覆盖可能导致崩溃的边界场景。

CLCC 对 DBMS 进行测试时的边覆盖数量优于其他基线工具的主要原因是 CLCC 引入 LLM, 生成了 SQL 关键词更丰富、SQL 语句数量更多的测试用例。相较而言, SQUIRREL、SQLRight 和 ParserFuzz 为基于变异的 DBMS 模糊测试方法, 在变异策略上没有考虑测试用例的 SQL 关键词丰富度和 SQL 语句数量对模糊测试的影响, 导致边覆盖数量较低。与 SQUIRREL 相比, CLCC 测试 DuckDB 的边覆盖数量提升较为明显。经分析发现, SQUIRREL 未针对 DuckDB 进行适应性调整, 生成的测试用例对其功能覆盖有限。而 CLCC 在对 DuckDB 进行测试时, 其生成的测试用例符合 DuckDB 的 SQL 语法要求, 且能够覆盖更多的 SQL 关键字, 因此覆盖了更多功能模块, 有效提升了边覆盖效果。

图 5 中, SQLRight 对 SQLite 进行测试的边覆盖数量逐渐上升, 为明确在长时间运行时 SQLRight 边覆盖数量是否优于 CLCC, 以及进一步评估 CLCC 在更长时间运行场景中的表现, 我们将 CLCC、SQUIRREL、SQLRight 和 ParserFuzz 分别运行 24 小时, 并对比其边覆盖情况。由于长时间调用 LLM 将产生较大的成本开销, 24 小时实验仅进行 1 次, 因此无法进行显著性检验与置信区间分析, 结果如图 6 所示。

从图 6 中可以看出, 经过 24 小时测试, CLCC 对 5 个被测 DBMS 的边覆盖数量整体同样优于 SQU-

① DuckDB's SQL Dialect, https://duckdb.org/docs/stable/sql/dialect/overview.html



注：阴影区域为标准差。

Fig. 5 Edge coverage results of different DBMS fuzz testing methods (2 hours)

图 5 不同 DBMS 模糊测试方法的边覆盖结果（2 小时）

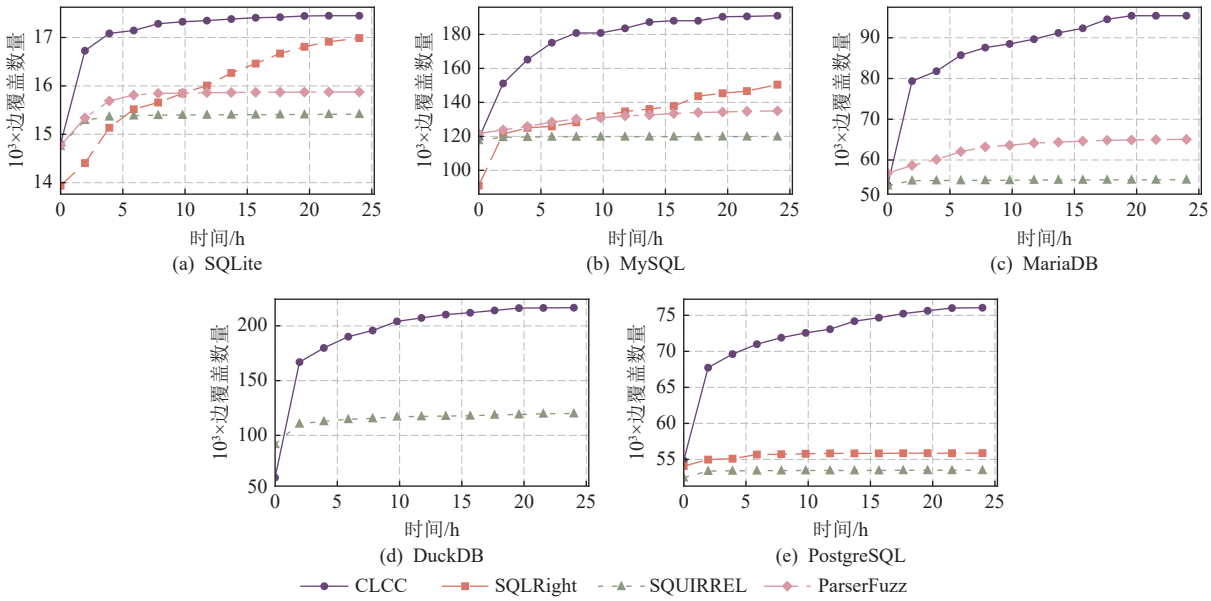


Fig. 6 Edge coverage results of different DBMS fuzz testing methods (24 hours)

图 6 不同 DBMS 模糊测试方法的边覆盖结果（24 小时）

IRREL、SQLRight 和 ParserFuzz。相比于 SQUIRREL，CLCC 对 SQLite、MySQL、MariaDB、DuckDB 和 PostgreSQL 进行测试的边覆盖数量分别增加了 13.19%、59.37%、73.04%、80.53% 和 42.17%。相比于 SQLRight，CLCC 对 SQLite、MySQL 和 PostgreSQL 进行测试的边覆盖数量分别增加了 2.72%、26.98% 和 36.12%。相比于 ParserFuzz，CLCC 对 SQLite、MySQL 和 MariaDB 进行测试的边覆盖数量分别增加了 9.93%、41.48% 和 46.45%。如图 5 所示，通过 2 小时的测试结果进行对比可以发现，CLCC 在长时间运行情况下的边覆盖数量相比于所有基线工具有了进一步提升，最高达到 80.53%，说明了 CLCC 在长时间测试场景下在边覆盖方面仍保持优势。

为分析边覆盖数量提升的主要原因，我们统计了 CLCC 和基线方法生成种子的 SQL 语句数量和 SQL 关键字覆盖情况，如表 3 所示。

从表 3 中可以看出，CLCC 对除 MySQL 和 PostgreSQL 以外的其他 DBMS 进行测试时生成种子中包含的

Table 3 Average SQL Statement Count and Coverage Ratio of Keyword

表 3 平均 SQL 语句数量和关键字覆盖比

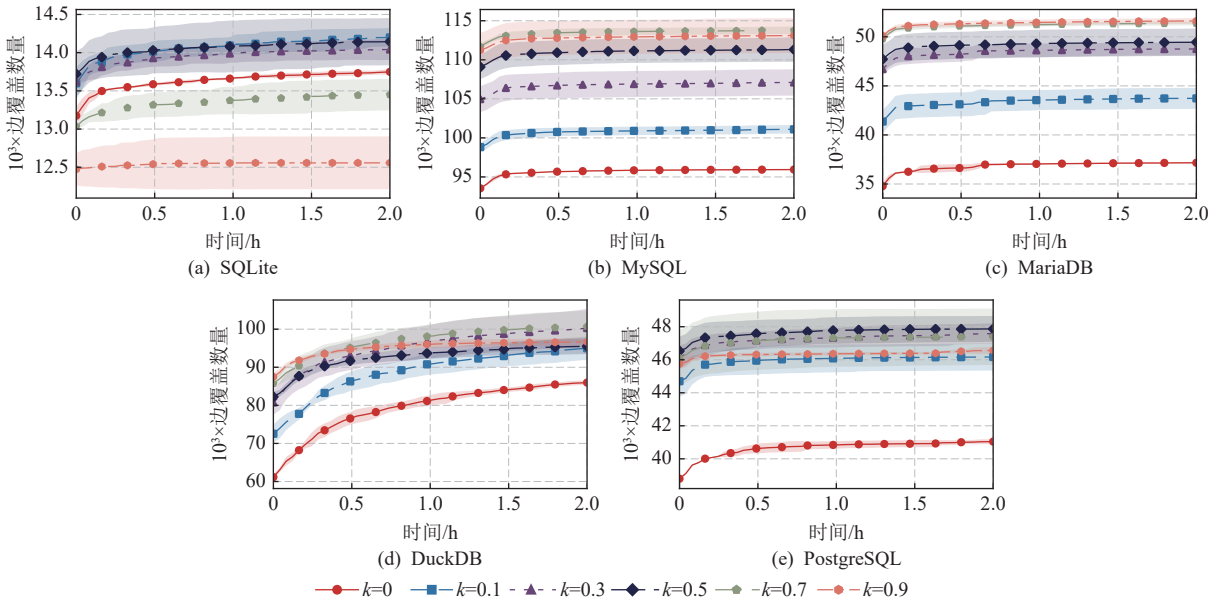
方法	SQLite		MySQL		MariaDB		DuckDB		PostgreSQL	
	ASS	KCR/%	ASS	KCR/%	ASS	KCR/%	ASS	KCR/%	ASS	KCR/%
CLCC	**20.02**	**100**	23.51	**78.60**	**12.61**	**88.00**	**18.34**	**83.30**	24.52	**96.10**
SQUIRREL	4.09	45.40	1.51	58.00	1.46	72.50	4.68	46.30	19.85	91.50
SQLRight	15.51	92.20	**26.43**	77.20					**28.21**	91.80
ParserFuzz	17.82	**100**	10.69	76.49	11.66	87.65				

注：ASS 表示平均 SQL 语句数量；KCR 表示 SQL 关键字覆盖比。黑体数值表示指标中的最优值。

SQL 语句数量和 SQL 关键字覆盖比优于 SQUIRREL、SQLRight 和 ParserFuzz。测试 MySQL 和 PostgreSQL 时，虽然 CLCC 的平均 SQL 语句数量略少于 SQLRight，但 CLCC 具有更高的 SQL 关键字覆盖比，因此能覆盖 DBMS 中的更多功能模块，从而提升边覆盖效果。

对于模糊测试边覆盖数量指标，CLCC 优于 SQUIRREL、SQLRight 和 ParserFuzz。相比于 SQUIRREL，CLCC 的边覆盖数量增加了 14.96%~49.31%。相比于 SQLRight，CLCC 的边覆盖数量增加了 6.09%~17.10%。相比于 ParserFuzz，CLCC 的边覆盖数量增加了 9.93%~46.45%。在长时间测试中，CLCC 对所有被测 DBMS 进行测试的边覆盖数量同样优于 SQUIRREL、SQLRight 和 ParserFuzz。CLCC 取得最优性能的原因在于其生成的测试用例中包含更多的 SQL 语句和更丰富的 SQL 关键字种类。

2）对于 RQ2 的回答。CLCC 在初始种子构建阶段针对不同 DBMS 构建扩增种子。为评价扩增种子的有效性，并排除 LLM 在测试用例生成阶段对实验的影响，我们对 SQUIRREL 使用不同初始种子进行对比实验。为实现公平对比，我们构建数量一致但混合比例不同的初始种子，其中混合比例k表示扩增种子占初始种子的比例，$1-k$为原有种子所占比例。例如，当$k=0.3$时表示使用 30% 的扩增种子和 70% 原有种子构建初始种子。实验中$k\in\{0.1,0.3,0.5,0.7,0.9\}$，结果如图 7 所示。



注：阴影区域为标准差。

Fig. 7 Edge coverage results of SQUIRREL using initial seeds with different mixing ratios (2 hours)

图 7 SQUIRREL 使用不同混合比例的初始种子的边覆盖结果（2 小时）

图 7 中的实验结果表明，相较于仅使用原有种子（$k=0$），当k=0.1, 0.3, 0.5 时，SQUIRREL 测试 SQLite、MySQL、MariaDB、DuckDB 和 PostgreSQL 的边覆盖数量均实现了一定程度的增加。其中，当k=0.5 时，在大多数被测 DBMS 的边覆盖数量增加幅度较大，分别增加了 2.87%、16.02%、33.07%、11.04% 和 16.63%。

实验结果表明，CLCC 使用 LLM 直接构建被测 DBMS 的初始种子可以有效提升 DBMS 模糊测试的边覆盖效果。

然而，当k值过大时，CLCC 测试部分 DBMS（如 SQLite, MySQL）的边覆盖数量提升较少，甚至低于仅使用原有种子的情况。例如在测试 SQLite 过程中，当k = 0.9时 SQUIRREL 比仅使用原有种子的边覆盖数量下降了 8.66%。造成这一情况的原因是由于 SQUIRREL 的 SQL 解析器无法解析部分扩增种子，导致其被视为无效种子，从而降低了边覆盖数量。

为具体地说明扩增种子的有效性，我们进行了案例研究，如图 8 所示。

```
1  CREATE TABLE products (
2     id INT PRIMARY KEY AUTO_INCREMENT,
3     name VARCHAR(255) NOT NULL,
4     price DECIMAL(10, 2) NOT NULL,
5     category VARCHAR(255)
6  );
7  INSERT INTO products (name, price, category) VALUES
8  ('Laptop', 1200.00, 'Electronics'),
9  ('Mouse', 25.00, 'Electronics'),
10 ('Book', 15.50, 'Books'),
11 ('Keyboard', 75.00, 'Electronics');
12 CREATE TABLE orders (
13    id INT PRIMARY KEY AUTO_INCREMENT,
14    product_id INT,
15    quantity INT NOT NULL,
16    order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
17    FOREIGN KEY (product_id) REFERENCES products(id)
18 );
```

```
19 INSERT INTO orders (product_id, quantity) VALUES
20 (1, 1),
21 (2, 5),
22 (3, 2),
23 (4, 1);
24 SELECT
25    p.name,
26    SUM(o.quantity) OVER (
27    ORDER BY
28    p.price RANGE BETWEEN UNBOUNDED PRECEDING
29    AND
30    UNBOUNDED FOLLOWING
31    ) AS total_quantity
32 FROM products p
33 JOIN orders o ON p.id = o.product_id;
34 SELECT JSON_ARRAYAGG(name) FROM products
35    WHERE category = 'Electronics';
36 SELECT seq FROM seq_1_to_5;
37 ANALYZE TABLE products;
38 SELECT * FROM products WHERE name LIKE '%a%';
```

Fig. 8　Case on augmented seeds

图 8　扩增种子案例

图 8 为 CLCC 测试 MariaDB 中的扩增种子示例，该扩增种子在建表语句（行 1~6 和行 12~18）、数据插入语句（行 8~11 和行 19~23）中使用了丰富的关键字，涵盖主键（PRIMARY KEY）、自增字段（AUTO_INCREMENT）、外键约束（FOREIGN KEY）以及不同数据类型，并且在查询语句（行 24~31 和行 34~35）中使用了窗口函数（*SUM*( )*OVER*）、窗口边界关键字（RANGE、UNBOUNDED 等）以及 JSON 函数（*JSON_ARRAYAGG*）。进一步地，该种子还包含 MariaDB 特有的语法特性。例如，在查询语句（行 36）中使用 MariaDB 中 Sequence 存储引擎[①]的内置虚拟表（seq_1_to_5）。

实验结果表明，混合原有种子和扩增种子作为初始种子可以增加 SQUIRREL 对 5 个被测 DBMS 进行测试的边覆盖数量，但当扩增种子占比过高时，可能会降低模糊测试的边覆盖效果。其中，当混合比例为 0.5 时，SQUIRREL 测试 DBMS 的边覆盖数量最高，相比于只使用原有种子时，边覆盖数量提升幅度最高达到 33.07%。

3）对于 RQ3 的回答。选择性 LLM 测试用例生成策略中，CLCC 将适应度超过阈值θ的种子作为部分提示，引导 LLM 生成测试用例。为确定阈值θ的不同取值对 CLCC 的影响，我们分别将θ设置为 0.1、0.5 和 1 来进行实验。未将θ设置为 0 进行实验是因为在此情况下所有种子均会作为 LLM 的提示输入，带来过高的 LLM 调用成本。当θ=1 时，所有归一化的种子适应度均不满足筛选标准，不会被用作 LLM 生成测试用例的提示词，即不使用选择性 LLM 测试用例生成策略。图 9 展示了设定不同阈值的 CLCC 2 小时测试下对 5 个 DBMS 进行测试的边覆盖情况。实验结果表明，当θ=0.1 时，CLCC 对 SQLite、MySQL、MariaDB、DuckDB 和 PostgreSQL 进行测试的边覆盖数量相比于θ=0.5 时分别提升了 2.57%、3.30%、19.24%、2.52% 和 2.82%；相比于θ=1 时在被测的 5 个 DBMS 中分别提升了 11.88%、7.34%、17.81%、19.13% 和 8.55%。实验结果表明，选择性 LLM 测试用例生成策略有效增加了 CLCC 的边覆盖数量。除 MariaDB 外，CLCC 对其他 DBMS 进行测试的边覆盖数量随θ值增加而下降，表明较低的θ值可以提高模糊测试的边覆盖数量。因此在实验设置中θ默认为 0.1。

边覆盖数量随θ值的增加而下降原因在于，当θ

① MariaDB sequence storage engines, https://mariadb.com/docs/server/server-usage/storage-engines/sequence-storage-engine

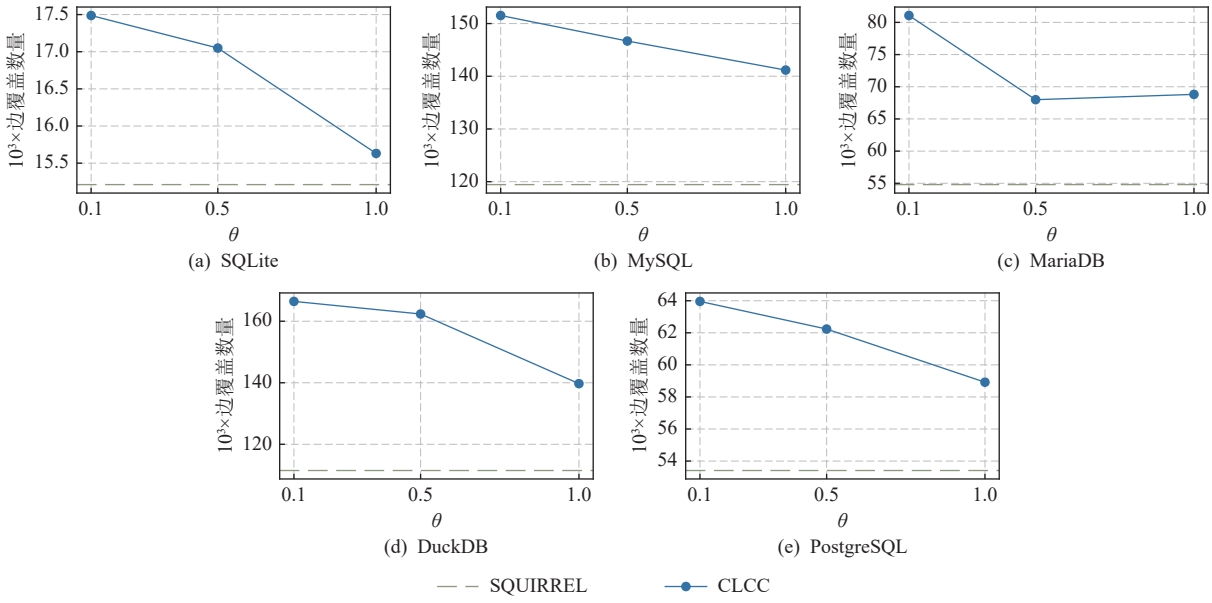


Fig. 9 Edge coverage results of the tested DBMS under different CLCC thresholds (2 hours)

图 9 不同 CLCC 阈值下被测 DBMS 的边覆盖结果(2 小时)

Table 4 Normalized Fitness Statistics of Generated Seeds Tested by CLCC on MariaDB

表 4 CLCC 测试 MariaDB 生成种子的归一化适应度统计

阈值θ	0.1	0.5	1
高于阈值的种子数	3 760	228	0
低于阈值的种子数	2 215	5 841	6 091
种子总数	5 975	6 069	6 091

值较高时, 满足条件的种子数量减少, 导致用于构建 LLM 提示的种子数量减少, 降低了选择性 LLM 测试用例生成策略所生成的测试用例数量。

需要注意的是, 当θ=1 时, CLCC 对 MariaDB 进行测试的边覆盖数量相比于θ=0.5 时提高了 1.21%, 未随θ值的增加而下降。为分析该异常结果, 我们统计了设定不同θ值时, CLCC 对 MariaDB 进行测试所生成种子的归一化适应度, 结果如表 4 所示。

表 4 中, CLCC 为 MariaDB 生成高于θ=0.5 的种子数仅比θ=1 时多 228 个(占总种子数的 3.76%)。可以看出, 由 LLM 生成的测试用例数量差异较小, 导致θ取值不同时, CLCC 对 MariaDB 进行测试时边覆盖数量接近, 从而未随θ值的增加下降。

选择性 LLM 测试用例生成策略的阈值影响 CLCC 对被测 DBMS 进行测试的边覆盖数量, 阈值越低则选择性 LLM 测试用例生成模块生成的测试用例越多, 从而能够覆盖更多边。相比于θ=0.1 时 CLCC 的边覆盖数量提升了 2.52%~19.24%。

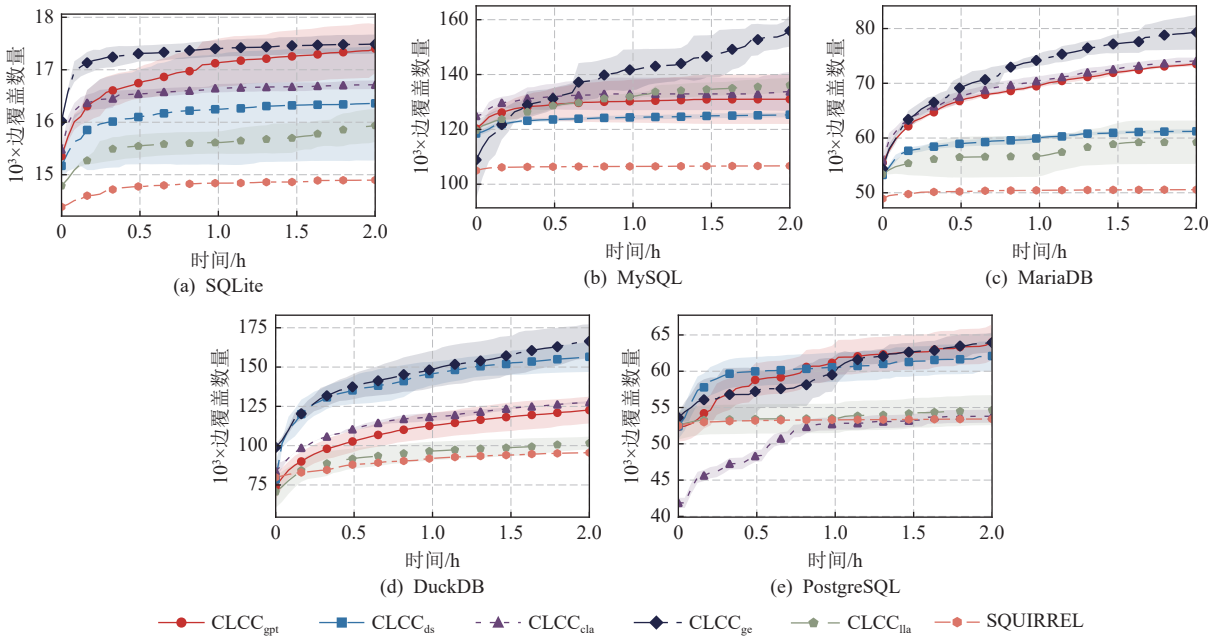
4)对于 RQ4 的回答。我们对比了在使用不同 LLM 时, CLCC 测试 5 个 DBMS 的边覆盖情况。我们使用 API 调用的 LLM 包括 gpt-3.5-turbo①、DeepSeek-V3[19]、claude-3-haiku-20240307②、Gemini-2.0-flash, 以及本地部署的 LLaMA-3.0:8B[20]。上述使用不同 LLM 的 CLCC 变体分别表示为 $CLCC_{gpt}$, $CLCC_{ds}$, $CLCC_{cla}$, $CLCC_{ge}$, $CLCC_{lla}$。实验结果如图 10 所示。

从图 10 中可以看出, 使用不同 LLM 的 CLCC 在 2 小时测试下测试 5 个 DBMS 时的边覆盖情况存在差异。其中, $CLCC_{ge}$ 对所有被测 DBMS 进行测试的边覆盖效果最优。此外, 相比于其他变体, $CLCC_{lla}$ 对除 MySQL 以外的其他 DBMS 进行测试的边覆盖数量最低, 可能是由于选择的本地 LLM 参数为 8×10^9, 较小的模型参数会使其生成的测试用例所包含的 SQL 语句数量和 SQL 关键字种类较少, 导致生成的测试用例难以覆盖更多 DBMS 功能模块。由于计算资源限制, 本文在实验中未对比更大参数的 LLaMA-3.0 模型。

使用不同的 LLM 对 CLCC 的性能有较大的影响。实验结果表明, 相比使用 gpt-3.5-turbo、DeepSeek-V3 等其他 4 个 LLM, 使用 Gemini-2.0-flash 的 CLCC 对 5 个 DBMS 进行测试的边覆盖表现最优, 提升了 0.47%~67.43%。

① gpt-3.5-turbo, https://platform.openai.com/docs/models/gpt-3.5-turbo

② claude-3-haiku, https://cloud.google.com/vertex-ai/generative-ai/docs/partner-models/claude/haiku-3



注：阴影区域为标准差。

Fig. 10 Edge coverage results of the tested DBMS with CLCC using different LLMs (2 hours)

图 10 使用不同 LLM 的 CLCC 对被测 DBMS 的边覆盖结果（2 小时）

5 讨 论

5.1 LLM 生成

由于 CLCC 依赖于 LLM 生成扩增种子与测试用例，因此 LLM 生成的有效性和成本将影响 CLCC 的测试效果及其在大规模测试中的可行性。

为分析 LLM 生成的有效性，我们对 LLM 生成的测试用例的语法语义正确率（syntax and semantic correctness rate, SSCR）、平均生成时间（average generation time, AGT）、平均生成成本（average generation cost, AGC）进行分析，结果如表 5 所示。

Table 5 Statistical of LLM Test Case Generation

表 5 LLM 测试用例生成统计

模型	SSCR/%	AGT/s	AGC/$
Gemini-2.0-flash	93.81	10.24	0.000 64
gpt-3.5-turbo	93.78	**2.11**	**0.000 35**
DeepSeek-V3	85.12	31.74	0.000 92
claude-3-haiku-20240307	**98.29**	7.89	0.000 87
LLaMA-3.0:8B	73.57	4.95	-

注：黑体数值表示该指标中的最优值。

表 5 中，claude-3-haiku-20240307 模型生成的测试用例在 SSCR 方面表现最优（98.29%）。然而结合 RQ4 的实验结论可以发现，模型所生成的测试用例的 SSCR 与使用该模型的 CLCC 进行测试的边覆盖效果并不严格正相关。造成这一情况的原因在于，CLCC 的测试效果不仅取决于 LLM 生成的测试用例是否语法语义正确，还依赖于生成的测试用例能否覆盖更多的 DBMS 功能模块和 SQL 关键字。因此，Gemini-2.0-flash 模型生成的测试用例虽然在 SSCR 方面低于 claude-3-haiku-20240307，但其在边覆盖效果方面优于它。

为进一步分析 LLM 在测试用例生成过程中出现的语法错误和语义错误及其占比，我们对 LLM 生成的测试用例中出现的错误进行了更细粒度的分析和统计，结果如表 6 所示。表 6 中，“表不存在”错误占比最高，约占总错误的 31.20%；其次是“语法错误”，约占总错误的 20.90%。在所有错误类型中，“表不存在”和“列不存在”等错误与 DBMS 对象引用相关的错误合计占 37.43%。这类错误产生的主要原因是 LLM 在生成测试用例时易对 DBMS 对象名称产生“幻觉”。此外，从表 6 中可以看出，与参数规模较大的模型相比，参数量较小的模型（如 LLaMA-3.0:8B）生成的测试用例更难保证基本 SQL 语法结构的正确性，更容易出现括号缺失、符号错误等基础语法错误。

另一方面，模型的 AGT 也将对 CLCC 的测试边覆盖效果产生影响。AGT 越低，单位时间内被动 LLM 测试用例生成策略生成的测试用例数量越多。然而，结合表 5 和 RQ4 的结果中可以得出，AGT 并非影响

Table 6 Error Types and Their Proportions of LLM-Generated Test Cases

表 6 LLM 生成的测试用例错误类型及其占比 %

模型	错误类型及其占比				
Gemini-2.0-flash	TNE（25.94）	SE（11.79）	FKV（4.25）	CNE（4.25）	其他类型（53.77）
gpt-3.5-turbo	TNE（42.70）	SE（20.97）	CNE（11.24）	CCV（8.61）	其他类型（23.95）
DeepSeek-V3	CCV（35.79）	FKV（25.26）	TNE（16.84）	SE（11.58）	其他类型（10.53）
claude-3-haiku-20240307	TNE（36.62）	FKV（16.90）	SE（14.08）	NNV（8.45）	其他类型（23.95）
LLaMA-3.0:8B	SE（46.09）	TNE（33.91）	CNE（15.65）	FE（1.74）	其他类型（2.61）

注：TNE：表不存在错误；SE：语法错误；FKV：违反外键约束；CNE：列不存在错误；NNV：违反非空约束；CCV：违反检查约束；FE：函数调用错误。

CLCC 边覆盖效果的决定性因素。例如，gpt-3.5-turbo 的 AGT 较低，但使用该模型的 CLCC 对 DBMS 进行测试的边覆盖效果低于 Gemini-2.0-flash。这说明，在 DBMS 模糊测试中，LLM 的选择需要综合考虑其生成的测试用例在 DBMS 功能与 SQL 关键字的覆盖能力、SSCR 和 AGT，而不能仅依赖单一指标。

LLM 的成本开销直接影响 CLCC 在大规模测试中的可行性。本文累计投入约 1 000$，用于探究 LLM 在 DBMS 模糊测试中的应用。表 7 展示了 Gemini-2.0-flash，在阈值 θ=0.1 时对不同 DBMS 进行 24 小时测试的成本开销及 Token 消耗情况。

Table 7 LLM Cost Overhead and Token Usage (24 h)

表 7 LLM 成本开销和 Token 消耗（24 h）

模型	SQLite	MySQL	MariaDB	DuckDB	PostgreSQL
成本开销/$	19.96	13.59	14.37	33.85	19.01
提示 Token 数*	62.98	50.83	53.17	109.69	60.19
补全 Token 数*	25.17	23.76	25.14	57.20	24.74

注：“*”表示百万单位。

由于不同 DBMS 的测试用例执行速度存在差异，因此 CLCC 的成本开销在测试不同 DBMS 时存在不同，其中 DuckDB 的成本开销最高（33.85$），MySQL 的成本开销最低（13.59$）。整体而言，CLCC 的平均每小时成本开销约为 0.84$，进行 24 小时测试的平均开销为 20.16$。表明通过对测试用例生成策略的设计以及种子适应度筛选，CLCC 可以通过阈值有效控制成本，在长时间、大规模测试中具有一定可行性。在成本开销组成方面，提示 Token 和补全 Token 的平均占比为 2.16，即提示词占成本开销的主要部分。这表明优化提示词设计、缩短提示长度是降低成本开销的潜在方向。在预算受限的情况下，可通过在测试中提高阈值的选择、对提示词进行优化精简或选择本地 LLM 等方式进一步减少 CLCC 的成本开销。结合前文不同阈值与 LLM 的实验结果，我们建议在阈值设定为 0.1、LLM 选择为 Gemini-2.0-flash 的配置下运行 CLCC。该参数选择能够在保证与本文一致的覆盖率表现的同时，将成本控制在可接受范围内，有效平衡了 CLCC 的测试覆盖率和 LLM 调用成本。

5.2 有效性分析

本节从内部有效性、外部有效性和构造有效性 3 个方面对所提方法进行有效性分析。

本方法的内部有效性威胁主要源于 CLCC 原型工具实现的正确性。为了降低内部有效性威胁，CLCC 基于现有 DBMS 模糊测试研究中广泛使用的 SQUIRREL 实现，并在 CLCC 中使用了 NumPy[①]等开源库。最后，我们对编写的代码进行了多次的检查，以确保 CLCC 的正确性。

本方法的外部有效性威胁主要源于被测 DBMS 的选择。为降低外部有效性威胁，我们选择了 5 个在现有 DBMS 模糊测试研究中被广泛使用的 DBMS[7,13]，涵盖客户端/服务器架构和嵌入式架构 2 种类型，并具有不同的 SQL 方言特性。对这些 DBMS 进行实验可在一定程度上验证 CLCC 的通用性，但是不能保证 CLCC 测试其他 DBMS 时可表现出相同的性能。

本方法的构造有效性威胁主要在于评估指标的有效性。为了降低构造有效性威胁，我们采用了边覆盖数量作为评估指标。这一指标已被广泛使用于现有的模糊测试研究中[8,11,17-18]。此外，为确保 CLCC 和基线工具对比的公平性，所有实验均通过 AFL++[17] 内置的工具收集边覆盖数量。

6 相关工作

CLCC 使用 LLM 进行初始种子构建和测试用例生成。其中，在测试用例生成部分结合了基于变异和基于 LLM 的测试用例生成策略。本节从初始种子生

① https://numpy.org/

成、基于变异的 DBMS 模糊测试和基于生成的 DBMS 模糊测试 3 个方面对相关工作进行介绍。

6.1 初始种子生成

初始种子是影响模糊测试性能的关键因素之一，其质量直接影响模糊测试的覆盖率和漏洞发现能力[21]。因此，一些现有工作对初始种子生成进行了改进，以提升模糊测试的效果。

NEUZZ[22]，BaSFuzz[23] 等方法使用原有种子进行模糊测试，将其中生成的有趣的测试用例作为后续模糊测试的初始种子，以提升初始种子的质量和覆盖效果。然而这种方法依赖原有初始种子的质量和多样性，导致其在不存在原有种子时获得的初始种子质量不佳。为此，SAFL[24] 通过符号执行来探索程序的不同执行路径，以生成代码覆盖效果更强的初始种子，减少了对于原有初始种子的依赖性。戴嘉润等人[25] 以真实世界中的数据集作为参考，通过自动化转换生成和真实世界场景等价的初始种子。近年来随着 LLM 的快速发展，已有部分工作使用 LLM 生成初始种子。例如 SEDAR[14] 利用 LLM 对其他 DBMS 的初始种子进行转换，得到 SQL 语义相同但符合目标 DBMS 语法的初始种子，从而提高 DBMS 模糊测试的边覆盖效果。

不同于上述初始种子生成方法，CLCC 使用 LLM 直接生成语法语义符合目标 DBMS 要求的扩增种子，避免了对于原有种子的依赖。尽管 SEDAR 方法同样引入 LLM 生成初始种子，但其方法主要关注于等义 SQL 语句转换，以适配目标 DBMS 的语法。然而，该方法存在局限性，若目标 DBMS 功能包含其他 DBMS 中不具备的功能模块，则等义 SQL 转换无法生成能够覆盖该功能的有效初始种子。而 CLCC 能够基于 LLM 对目标 DBMS 直接生成测试用例，在功能模块覆盖方面具备更强的适应性。此外，CLCC 生成的扩增种子在 SQL 语句数量和 SQL 关键字丰富度方面优于原有种子，进一步提升了测试边覆盖效果。

6.2 基于变异的 DBMS 模糊测试

基于变异的模糊测试利用覆盖率和测试用例的执行信息，引导新的测试用例生成，从而提升模糊测试的效率和覆盖率[26]。由于基于变异的模糊测试能够深度探索程序[27]，该方法已被广泛应用到 DBMS 测试中[1]。

SQUIRREL[7] 首先将 SQL 语句转换为 IR，然后对 IR 进行变异，最后通过语义感知，将变异后的 IR 转换为语法语义正确的 SQL 语句，以对 DBMS 进行测试。然而 SQUIRREL 生成的 SQL 语句在 SQL 关键字种类和组合方面较为有限，限制了测试用例的多样性，从而影响了模糊测试的覆盖效果。为此，LEGO[9] 探索了测试用例中 SQL 语句间的关系。变异过程中，LEGO 分析 SQL 关键字以及 SQL 语句类型之间的亲和性，即语句间的承接关系，并基于此生成新的测试用例，提升了 DBMS 模糊测试的覆盖率。尽管 LEGO 有效提升了基于变异的 DBMS 模糊测试的覆盖效果，但其方法均依赖于对 SQL 语句的正确解析，这在面向不同方言特性的 DBMS 时降低了测试方法的通用性。为此，GRIFFIN[10] 使用了一种无语法依赖的测试用例变异方法，在测试过程中记录 SQL 语句对数据库对象（例如表、列）的依赖关系，并以此构建元数据图。通过对元数据图结构进行重新组合以生成新的测试用例，从而在有效提升覆盖率的同时增强了 DBMS 模糊测试的通用性。除使用元数据图的方式增强 DBMS 模糊测试的通用性以外，BUZZBEE[16] 方法通过构建上下文查询语言（context query language，CQL）定义 SQL 语句中的语义语法约束，通过高度抽象的 CQL 实现对于关系型数据库和非关系型数据库通用的 DBMS 模糊测试工具。上述方法主要关注 DBMS 中存在的崩溃漏洞，难以有效发现 DBMS 中存在的逻辑错误。为此，SQLRight[12] 将测试预言机和覆盖率引导相结合，通过构造功能等价的 SQL 语句并对比其结果以提升 DBSM 模糊测试发现逻辑错误的能力（构造功能等效的 SQL 语句以判断二者结果是否一致）。除上述的覆盖率引导的基于变异的 DBMS 模糊测试以外，部分工作提出了其他引导方式。例如，QPG[15] 提出了一种基于 SQL 语句查询计划的引导方式，触发了 53 个未知的 DBMS 漏洞。

和上述方法不同，CLCC 结合了基于变异的测试用例生成策略和基于 LLM 的测试用例生成策略。一方面，基于变异的测试用例生成策略能够高效地生成大量测试用例；另一方面，基于 LLM 的测试用例生成策略能够高效兼容不同 DBMS 的方言特征，有针对性地生成语法语义正确的测试用例。通过结合二者的优势，CLCC 能够有效提升 DBMS 模糊测试的覆盖效果。

6.3 基于生成的 DBMS 模糊测试

基于生成的数据库模糊测试通过预设的语法语义规则直接生成测试用例。相较于基于变异的模糊测试，基于生成的模糊测试可以生成结构完整、语义合理的复杂测试用例。因此，目前已有部分工作将基于生成的模糊测试运用于 DBMS 测试中。

例如，SQLsmith 通过预定义的抽象语法树（ab-

stract syntax tree, AST)生成测试用例, 以触发 DBMS 的崩溃漏洞。AMOEBA[28] 通过生成具有相同查询结果但结构不同的 SQL 语句来对比执行时间的差异, 以检测 DBMS 的性能错误。DQE[29] 通过生成具有相同谓词的 SQL 语句, 并通过比较它们访问的数据库行是否一致, 以检测由 UPDATE 和 DELETE 类型的 SQL 语句引发的 DBMS 逻辑错误。上述方法可以有效地生成大量 SQL 语句对 DBMS 进行测试, 但无法检测 DBMS 中内置函数所导致的崩溃。因此 SOFT[30] 方法详细分析了现有 DBMS 的 CVE 中内置函数出现漏洞的模式, 通过生成预定义的 DBMS 内置函数调用方法对 DBMS 进行测试, 发现了大量先前工作中未发现的漏洞。

和上述基于生成的数据库模糊测试方法不同, CLCC 分析种子池中种子的边覆盖信息以筛选适应度高于阈值的种子作为 LLM 提示的一部分, 为 LLM 测试用例生成提供了引导。并且 CLCC 在生成的测试用例中同时包含了 SQL 语句和 DBMS 内置函数, 进一步提升 DBMS 模糊测试的边覆盖效果。

7 总 结

本文提出了一种利用 LLM 进行初始种子构建与测试用例生成的 DBMS 模糊测试方法 CLCC。通过引入 LLM, CLCC 生成的测试用例在 SQL 语句数量和 SQL 关键字丰富度方面优于现有方法, 从而能够覆盖更多的 DBMS 功能模块, 提升了测试的边覆盖效果。实验结果表明, CLCC 在边覆盖指标上优于现有模糊测试工具 SQUIRREL、SQLRight 和 ParserFuzz。

然而, 由于成本限制, 本文仅在所选的 5 个 DBMS 上进行了相对有限时间的测试, 因此未能发现真实的崩溃错误。在未来的工作中, 我们计划进行更长时间的实验, 以评估 CLCC 的崩溃触发能力。

未来的研究计划从 3 个方向展开。1)降低 CLCC 在大规模测试中的使用成本。当前 LLM 调用费用是限制 CLCC 扩展性的重要因素, 未来将尝试对本地开源 LLM 进行微调, 以更低成本生成高质量测试用例。2)提升测试的并行性与吞吐量。未来将构建并行化的多代理模糊测试架构, 使多个微调后的本地 LLM 代理能够协作完成测试任务, 从而进一步提升 CLCC 在长时间、跨 DBMS 的大规模测试场景中的可扩展性。3)提升不同 DBMS 的适配能力。探索构建通用 SQL 解析工具, 使 DBMS 模糊测试方法能够同时适用于关系型和非关系型 DBMS。

作者贡献声明: 张瑞宸负责相关工作调研, 完成实验、论文撰写及修订等工作; 张亚东、兰文尉参加相关工作调研并修改论文; 岳清提出指导意见并修改论文; 王正武协助完成实验; 崔展齐指导论文选题、确定论文整体框架设计并修改论文。

参 考 文 献

[1] Liang Jie, Wu Zhiyong, Fu Jingzhou, et al. Survey on database management system fuzzing techniques[J]. Journal of Software, 2025, 36(1): 399−423 (in Chinese)
(梁杰, 吴志镛, 符景洲, 等. 数据库管理系统模糊测试技术研究综述[J]. 软件学报, 2025, 36(1): 399−423)

[2] Yamada H, Nemoto J. Scalar D L: Scalable and practical Byzantine fault detection for transactional database systems[J]. Proceedings of the VLDB Endowment, 2022, 15(7): 1324−1336

[3] Ahmad M S, Panda B N. ClusteredLog: Optimizing log structures for efficient data recovery and integrity management in database systems[J]. Electronics, 2024, 13(23): 4723

[4] Clark J, Donaldson A F, Wickerson J, et al. Validating database system isolation level implementations with version certificate recovery[C]//Proc of the 19th European Conf on Computer Systems. New York: ACM, 2024: 754−768

[5] Alam M I, Halder R. Formal verification of database applications using predicate abstraction[J]. SN Computer Science, 2021, 2(3): 135

[6] Liu Shuo, Tian Chunlei, Sun Jun, et al. Semantic conformance testing of relational DBMS[J]. Proceedings of the VLDB Endowment, 2024, 18(3): 850−862

[7] Zhong Rui, Chen Yihui, Hu Hao, et al. SQUIRREL: Testing database management systems with language validity and coverage feedback[C]//Proc of the 2020 ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2020: 955−970

[8] Wang Mingzhe, Wu Zhiyong, Xu Xinyu, et al. Industry practice of coverage-guided enterprise-level DBMS fuzzing[C]//Proc of the 43rd Int Conf on Software Engineering: Software Engineering in Practice. Piscataway, NJ: IEEE, 2021: 328−337

[9] Liang Jie, Chen Yuanguang, Wu Zhiyong, et al. Sequence-oriented DBMS fuzzing[C]//Proc of the 39th IEEE Int Conf on Data Engineering. Piscataway, NJ: IEEE, 2023: 668−681

[10] Fu Jingzhou, Liang Jie, Wu Zhiyong, et al. GRIFFIN: Grammar-free DBMS fuzzing[C]//Proc of the 37th IEEE/ACM Int Conf on Automated Software Engineering. New York: ACM, 2023: 1−12

[11] Liang Jie, Wu Zhiyong, Fu Jingzhou, et al. WingFuzz: Implementing continuous fuzzing for DBMSs[C]//Proc of the 2024 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2024: 479−492

[12] Liang Yu, Liu Shuo, Hu Hao. Detecting logical bugs of DBMS with coverage-based guidance[C]//Proc of the 31st USENIX Security Symp. Berkeley, CA: USENIX Association, 2022: 4309−4326

[13] Liang Yu, Hu Hong. Parser knows best: Testing DBMS with coverage-guided grammar-rule traversal[EB/OL]. 2025[2025-12-22]. https://arxiv.org/abs/2503.03893

[14] Fu Jingzhou, Liang Jie, Wu Zhiyong, et al. SEDAR: Obtaining high-quality seeds for DBMS fuzzing via cross-DBMS SQL transfer[C]// Proc of the 46th IEEE/ACM Int Conf on Software Engineering. New York: ACM, 2024: 1−12

[15] Ba J, Rigger M. Testing database engines via query plan guidance[C]// Proc of the 45th IEEE/ACM Int Conf on Software Engineering. Piscataway, NJ: IEEE, 2023: 2060−2071

[16] Yang Yupeng, Chen Yihui, Zhong Rui, et al. Towards generic database management system fuzzing[C]//Proc of the 2024 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2024: 901−918

[17] Fioraldi A, Maier D, Eißfeldt H, et al. AFL++: Combining incremental steps of fuzzing research[C]//Proc of the 14th USENIX Workshop on Offensive Technologies. Berkeley, CA: USENIX Association, 2020: 1−12

[18] Song Congxi, Xu Zhou, Yin Qidi, et al. P-FUZZ: A parallel grey-box fuzzing framework[J]. Applied Sciences, 2019, 9(23): 5100

[19] DeepSeek-AI, Liu Aixin, Feng Bei, et al. DeepSeek-V3 technical report[EB/OL]. 2024[2025-12-22]. https://arxiv.org/abs/2412.19437

[20] Dubey A, Jauhri A, Pandey A, et al. The LLaMA 3 herd of models[EB/OL]. 2024[2025-12-22]. https://arxiv.org/abs/2407.21783

[21] Ren Zezhong, Zheng Han, Zhang Jiayuan, et al. A review of fuzzing techniques[J]. Journal of Computer Research and Development, 2021, 58(5): 944−963 (in Chinese)
(任泽众, 郑晗, 张嘉元, 等. 模糊测试技术综述[J]. 计算机研究与发展, 2021, 58(5): 944−963)

[22] She Doudou, Pei Kaixin, Epstein D, et al. NEUZZ: Efficient fuzzing with neural program smoothing[C]//Proc of the 2019 IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2019: 803−817

[23] Lan Wenwei, Huang Chen, Yu Tingting, et al. BaSFuzz: Fuzz testing based on difference analysis for seed bytes[J]. Journal of Systems and Software, 2025, 222: 112340

[24] Wang Mingzhe, Liang Jie, Chen Yanli, et al. SAFL: Increasing and accelerating testing coverage with symbolic execution and guided fuzzing[C]//Proc of the 40th Int Conf on Software Engineering: Companion Proc. New York: ACM, 2018: 61−64

[25] Dai Jiarun, Li Zhongrui, Zhang Wanqi, et al. Simulation-based fuzzing for autonomous driving systems: Landscapes, challenges and prospects[J]. Journal of Computer Research and Development, 2023, 60(7): 1433−1447 (in Chinese)
(戴嘉润, 李忠睿, 张琬琪, 等. 面向无人驾驶系统的仿真模糊测试: 现状、挑战与展望[J]. 计算机研究与发展, 2023, 60(7): 1433−1447)

[26] Cui Zhanqi, Zhang Jiaming, Zheng Liwei, et al. A survey of research on coverage-guided greybox fuzzing[J]. Chinese Journal of Computers, 2024, 47(7): 1665−1696 (in Chinese)
(崔展齐, 张家铭, 郑丽伟, 等. 覆盖率制导的灰盒模糊测试研究综述[J]. 计算机学报, 2024, 47(7): 1665−1696)

[27] Zhang Jiaming, Cui Zhanqi, Chen Xiang, et al. DeltaFuzz: Historical version information guided fuzz testing[J]. Journal of Computer Science and Technology, 2022, 37(1): 29−49

[28] Liu Xiaoyuan, Zhou Qiang, Joy A, et al. Automatic detection of performance bugs in database systems using equivalent queries[C]// Proc of the 44th Int Conf on Software Engineering. New York: ACM, 2022: 225−236

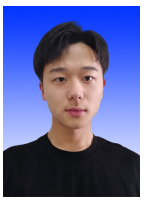
[29] Song Jiansheng, Dou Weishan, Cui Zeyu, et al. Testing database systems via differential query execution[C]//Proc of the 45th IEEE/ACM Int Conf on Software Engineering. Piscataway, NJ: IEEE, 2023: 2072−2084

[30] Fu Jingzhou, Liang Jie, Wu Zhiyong, et al. Understanding and detecting SQL function bugs: Using simple boundary arguments to trigger hundreds of DBMS bugs[C]//Proc of the 20th European Conf on Computer Systems. New York: ACM, 2025: 1061−1076



Zhang Ruichen, born in 2001. Master candidate. Student member of CCF. His main research interest includes software analysis and testing technology.

张瑞宸, 2001 年生。硕士研究生。CCF 学生会员。主要研究方向为软件分析及测试技术。



Zhang Yadong, born in 2002. Master candidate. His main research interest includes software analysis and testing technology.

张亚东, 2002 年生。硕士研究生。主要研究方向为软件分析及测试技术。



Lan Wenwei, born in 1999. Master. His main research interest includes software analysis and testing technolgy.

兰文尉, 1999 年生。硕士。主要研究方向为软件分析及测试技术。



Yue Qing, born in 1973. Master. Her main research interests include information systems and software engineering.

岳　清, 1973 年生。硕士。主要研究方向为信息系统、软件工程。



Wang Zhengwu, born in 1998. Master candidate. His main research interest includes software analysis and testing technology.

王正武, 1998 年生。硕士研究生。主要研究方向为软件分析及测试技术。



Cui Zhanqi, born in 1984. PhD, professor, PhD supervisor. Senior member of CCF. His main research interests include intelligent software engineering and trustworthy artificial intelligence.

崔展齐, 1984 年生。博士, 教授, 博士生导师。CCF 高级会员。主要研究方向为智能软件工程、可信人工智能。