

反向验证增强的 SQL 优化生成方法

廖雨桐, 李 熳, 谢炜业, 那 俊[†], 张 斌

东北大学软件学院, 辽宁 沈阳 110169

收稿日期: 2025-11-14 [†] 通信联系人 E-mail: naj@swc.neu.edu.cn

基金项目: 辽宁省科学技术计划重点研发计划(揭榜挂帅)(2025JH2/102800014)

第一作者: 廖雨桐, 男, 硕士生, 现从事基于大语言模型的智慧金融等方面的研究。E-mail: liaoyt@stumail.neu.edu.cn

摘 要: 针对当前基于大语言模型(Large Language Model, LLM)的自然语言转结构化查询语句(Text-to-Structured Query Language, Text-to-SQL)方法, 存在隐蔽语义错误难以闭环验证、执行准确率与执行效率难以兼顾的核心痛点, 提出一种基于反向验证增强的 SQL 优化生成方法。该方法通过反向验证增强机制, 以正向查询结果为已知条件构建双向语义验证闭环, 能够有效识别符合 SQL 语法但语义偏离的隐蔽错误; 同时设计查询范式演进机制, 通过相似查询逻辑结构树的复用降低推理开销, 优化验证流程的执行效率。实验结果表明, 在 Spider-test 和 BIRD Mini-dev 数据集上, 本文方法分别将各基线模型执行准确率最高提升 5.92 个百分点和 8.60 个百分点; 其核心的查询范式演进机制, 在 100% 命中率场景下可减少 57% 的反向验证耗时。基于该方法构建的股票信息查询智能 Agent 与智能选股平台, 验证了方法的落地实用性与场景适配能力。

关 键 词: 大语言模型; 代码生成; 智能系统; 文本转 SQL; 反向验证; 查询范式演进

中图分类号: TP181

文献标志码: A

文章编号: 1671-8836(2026)04-0474-13

A SQL Optimization Generation Method Based on Enhanced Reverse Validation

LIAO Yutong, LI Man, XIE Weiye, NA Jun[†], ZHANG Bin

Software College, Northeastern University, Shenyang 110169, Liaoning, China

Abstract: Current large language model (LLM)-based Text-to-Structured Query Language (Text-to-SQL) methods face two core challenges: difficulty in closed-loop verification of hidden semantic errors, and the inability to balance execution accuracy and efficiency. To address these gaps, we propose a SQL optimization generation method based on enhanced reverse validation, with a reverse validation mechanism that constructs a bidirectional semantic verification loop using forward query results as known conditions to effectively identify hidden errors that conform to SQL syntax but deviate semantically, and a query paradigm evolution mechanism to reduce the extra inference overhead from reverse validation. Experimental results on the Spider-test and BIRD Mini-dev datasets show that the proposed method improves the execution accuracy of various baseline models by up to 5.92 and 8.60 percentage points, respectively. Its core query paradigm evolution mechanism can reduce reverse verification time by 57% under a 100% hit rate scenario. The intelligent agent for stock information queries and the intelligent stock selection platform built on this method have demonstrated practical applicability and scenario adaptability in real-world deployments.

Key words: large language model; code generation; intelligent system; Text-to-SQL; reverse validation; query paradigm evolution

0 引 言

随着关系型数据库在各行各业的广泛部署, 如何

降低非专业用户的数据查询门槛成为了人机交互领域的重要课题。Text-to-SQL(又称 NL2SQL, 即自然语言转结构化查询语言)技术, 支持用户用日常语

引用格式: 廖雨桐, 李熳, 谢炜业, 等. 反向验证增强的 SQL 优化生成方法[J]. 武汉大学学报(理学版), 2026, 72(4): 474-486. DOI: 10.14188/j.1671-8836.2025.0215.

LIAO Yutong, LI Man, XIE Weiye, *et al.* A SQL Optimization Generation Method Based on Enhanced Reverse Validation [J]. *J. Wuhan Univ. (Nat. Sci. Ed.)*, 2026, 72(4): 474-486. DOI: 10.14188/j.1671-8836.2025.0215(Ch).

言直接与数据库交互,是实现数据普惠访问的核心技术。近年来,大语言模型(Large Language Model, LLM)凭借其强大的语义理解与代码生成能力,已逐渐取代传统的基于规则或小模型的方法,成为 Text-to-SQL 任务的主流实现范式。然而,在复杂业务场景(如金融科技、智能投顾等)下,面对海量数据与高度复杂的数据库模式,如何确保 LLM 生成 SQL 的准确性,仍是当前亟待突破的关键技术。

具体而言,现有基于 LLM 的 Text-to-SQL 系统面临的关键问题是隐蔽语义错误难以闭环验证:LLM 在理解复杂自然语言意图和适配多源异构数据库结构时,极易产生隐蔽的“语义错误”(如错误理解实体关系、遗漏关键约束条件或误用聚合函数^[1],包含此类错误的 SQL 语句往往能正常执行而不触发语法报错,导致返回错误数据;而现有的验证机制大多依赖静态的语法验证或单向的执行反馈,通常无法建立全面闭环的逻辑验证,难以保证查询语义的高准确性。

针对上述问题,本文提出了基于反向验证增强的 SQL 优化生成方法。相较于现有单向、静态的验证策略,本文的反向验证机制创新性地将查询结果作为已知条件逆推原始约束,实现了严谨的逻辑闭环语义校验;同时,本文设计了查询范式演进机制,通过复用相似历史范式,在保障验证精度的同时,将后处理资源的开销降至最低,实现了准确性与效率的协同破局。本文的主要贡献如下:

1) 提出反向验证增强机制,通过构建正-反双向查询验证闭环,将查询结果作为已知条件生成反向验证问题,并与原始约束进行逻辑一致性比对,从而实现语义错误的自主定位与精准修正;

2) 针对反向验证可能引入的效率损耗问题,提出查询范式演进机制,通过查询结构树(Query Structure Tree, QST)的树编辑距离度量实现相似范式的高效匹配与复用,在保障验证准确性的前提下显著降低推理开销。

两大机制形成“准确性-效率”的协同优化关系:反向验证机制确保查询语义正确,查询范式演进机制保障验证过程高效,二者共同构成了一套完整、实用且可扩展的 Text-to-SQL 后处理增强方案。

1 相关工作

为解决基于 LLM 的 Text-to-SQL 系统面临的“语义错误”问题,后处理技术通过对 LLM 初始生成的 SQL 进行优化、校验与筛选,已经成为提升 Text-to-SQL 系统可用性与可靠性的核心手段。

目前国内外学者已开展大量研究,涌现出 CHASE-SQL^[2]、XiYan-SQL^[3]等前沿方法:CHASE-SQL^[2]采用多智能体生成-选择机制,通过成对比较生成并排序多样化 SQL 查询,在基准测试中达到了较高水平;XiYan-SQL^[3]则通过集成多生成器协同策略、增强型模式表征以及结合微调与上下文学习的混合方法,构建了综合性的 SQL 生成与优化框架。纵观现有研究体系,当前主流的 Text-to-SQL 后处理研究可分为四大类:SQL 修正策略、输出一致性策略、执行引导策略与 N-Best 重排序策略。

1.1 SQL 修正策略

该类方法以修正 SQL 的显性错误为核心目标,针对初始生成 SQL 的语法错误、库表元素匹配错误等问题,通过提示工程、多阶段匹配等方式实现定向纠错,是后处理技术中最基础的研究方向。例如,针对 Text-to-SQL 可能产生的语法错误,DIN-SQL^[4]引入了一个在零样本设置下运行的自我纠正模块,其中模型仅接收有缺陷的 SQL 语句并尝试纠正。具体地,DIN-SQL 使用了两个提示:基于 CodeX 的通用提示直接要求错误识别和纠正,以及基于 GPT-4 的温和提示,在不假定错误的情况下寻找潜在问题。为了处理谓词预测中的错误,例如不正确的列或值,ZeroNL2SQL^[5]采用了多级匹配方法,以增量方式扩展列、表和数据库之间的匹配,允许将匹配的值返回给 LLM,以生成与数据库内容一致的 SQL 查询。然而,这些方法均侧重于修复语法错误,经常忽略语义错误^[6],因为这些方法缺乏真实数据的执行反馈,完全依赖模型自身的参数化知识。当面对复杂的“隐性语义错误”(如错误的表链接、不准确的聚合逻辑)且该错误恰好符合 SQL 语法规则时,此类方法通常会“视而不见”,无法保证查询结果在业务逻辑层面的准确性。

1.2 输出一致性策略

为了提高输出结果的稳定性,研究者引入了“自一致性(Self-consistency)”^[7]方法,通过对不同推理路径的采样并最终采纳其中最一致的答案,提升 SQL 查询的质量。该类方法以降低 LLM 生成随机性、提升输出稳定性为核心目标,通过多路径采样生成多组候选 SQL,基于投票、聚类统计方法筛选最优解。例如,DAIL-SQL^[8]框架内集成了自一致性策略,使其模型性能相较于基线配置提升了 0.4%。另一方面,为了降低 LLM 的随机性影响,FinSQL^[9]内部建立了一种聚类方法,首先生成 n 个候选 SQL 查询,然后基于关键词的一致性对它们进行分组,并从最大的分组中选取一个查询作为最终输出。这种自一致性策略通常通过提高采样温度增加输出的多样性,再通过

多数投票来确定最终结果。然而,有研究^[10]表明,仅仅依赖单个模型可能难以产生足够多样化的候选答案。为了突破这一局限,Li等^[11]提出了一种“交叉一致性”(Cross-consistency)策略,让多个不同的 LLM 在较低的采样温度下分别生成 SQL 查询,再根据这些查询的实际执行结果进行投票,选出最优查询。然而,输出一致性策略存在严重的“一致性幻觉”问题。当面对复杂约束时,LLM 极易产生系统性的语义理解偏差,从而“一致地”生成包含相同逻辑错误的语句。此外,多次采样和投票机制会成倍增加推理延迟,导致后处理资源开销较大。

1.3 执行引导策略

SQL 查询的实际执行结果通常是评估 Text-to-SQL 准确性的重要依据。执行引导策略模仿人类专家编写 SQL 的迭代调试过程,以 SQL 的实际执行反馈(报错、空值、异常结果等)为核心依据,迭代优化初始 SQL,核心目标是提升 SQL 的可执行性与结果合理性。例如,当查询执行后返回错误信息或空值(NULL)时,这通常暗示着生成的 SQL 查询可能存在问题。为了模仿人类专家编写复杂 SQL 查询的迭代过程,CHESS^[12]以接收数据库模式、自然语言问题、一个候选 SQL 查询以及该查询的执行结果作为输入,从一个初始草稿查询出发,根据执行反馈不断进行优化和调整,并在必要时修正语法错误。CodeS^[13]则采用了一种更为直接的生成方式,通过波束搜索(Beam Search)一次性生成四个完整的 SQL 候选查询,并简单地选择第一个能够成功执行的查询作为最终输出。执行引导策略存在的问题是,执行产生的验证信号过于单一,且是“单向”的。它仅能通过执行是否报错判断对错,但实际应用中最致命的错误往往是 SQL 执行成功但返回了错误的数据。由于缺乏对查询意图的逆向比对,这类方法极易忽视逻辑层面的严重漏洞。

1.4 N-Best 重排序策略

在跨域场景中,生成的 SQL 查询在结构和语义上可能略有不同。N-best 重新排序试图利用更大的模型或额外的知识源对 Top-N 模型输出进行重新排序。然而,重排序方法在实际应用中效果并不稳定,且对阈值(Threshold)的设置非常敏感,有时甚至会带来负面效果。为了解决这些局限性,G³R^[14]在结构中包含了一种特征增强的重排位器,利用基于预训练语言模型的混合提示调整(Hybrid Prompt Tuning)技术,在不增加额外参数的情况下弥合领域差异,并采用对比学习(Contrastive Learning)有效增大不同候选查询之间的区分度。与此类似,ReFSQL^[15]模型则通过从检索器(Retriever)和生成器(Generator)模块中分别提

取最相关的结果进行整合提高执行准确率。但是,该策略的性能高度依赖于初始候选池的质量和阈值敏感设置。如果 Top-N 的候选集中根本就没有语义正确的 SQL,重排序则无法起到作用。同时,调用庞大的排序模型同样带来了极高的计算成本。

针对现有研究存在的问题,本文将从语义闭环验证和高效推理两个维度展开设计。

2 本文方法

2.1 方法概述

本文提出一种基于反向验证增强的 SQL 优化生成方法。该方法由两大核心机制构成:反向验证增强机制与查询范式演进机制。其中,反向验证增强机制是保障系统执行准确率的核心框架,通过将查询结果作为已知条件逆推原始约束,建立严谨的语义闭环校验;查询范式演进机制则是内嵌于反向验证流程中的提效模块,通过精准匹配并复用相似的历史查询范式,大幅降低反向验证所带来的额外推理开销。两大机制属于包含与协同关系,共同构成了一套兼顾准确性与效率的完整 Text-to-SQL 后处理方案。

2.2 理论基础

2.2.1 反向验证增强机制的理论基础

反向验证增强机制的核心逻辑可以通过关系代数形式化地表达。一个标准的正向查询过程可表示为:

$$\text{Result}(\text{Condition}(\text{Table})) \quad (1)$$

其中,Table 表示所查的数据表,Condition 表示查询条件,Result 表示查询结果。

本机制通过将正向查询(1)式中的查询条件与结果进行逻辑反转,构建反向查询:

$$\text{Condition}(\text{Result}(\text{Table})) \quad (2)$$

通过验证正向查询与反向查询的结果是否在逻辑上保持一致,即可判断正向问题和反向问题的逻辑一致性,从而建立一个动态的、基于执行结果的验证闭环。

2.2.2 查询范式演进机制的理论基础

为实现相似查询范式的精准匹配,需先对反向验证问题进行结构化表示——此场景下的相似度度量核心是查询逻辑结构的一致性,与传统自然语言语义相似度存在本质区别。查询范式演进机制的目标是复用历史查询逻辑并微调,若仅将反向验证问题表征为嵌入向量,会丢失实体关系、约束条件等关键逻辑结构,因此不可取。

本研究受 NaLIR^[16]框架启发,设计 QST,以形式化表达反向验证问题的内部逻辑结构。一棵查

询结构树 T 可被定义为三元组:

$$T = (N, E, n_0) \quad (3)$$

其中, N 为节点集合, 每个节点对应查询逻辑中的原子组件 (如实体属性“上市时间”、值“2025年”、关系运算符“after(>)”、聚合函数“AVG”等); $E \subseteq N \times N$ 为边的集合, 表征节点间的逻辑依赖与修饰关系 (例如, 实体节点“上市时间”通过边连接至关系运算符节点“after(>)”, 再通过边连接至值节点“2025年”, 共同表达“股票上市时间 > 2025年”的筛选条件); n_0 为唯一根节点, 对应查询最外层操作 (通常为 SELECT 结构节点, 定义查询最终需返回的目标实体, 如“股票代码”“收盘价”等)。图1展示了典型QST的示例, 直观呈现了从自然语言查询到结构化逻辑的映射关系。

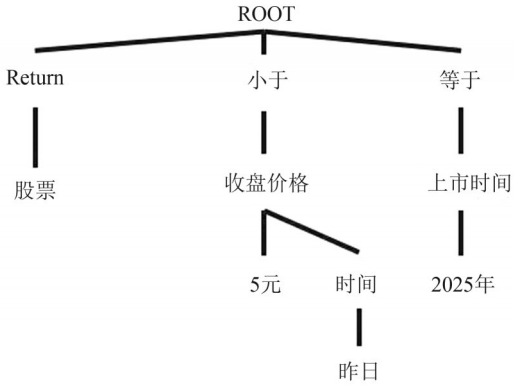


图1 查询结构树的结构

Fig. 1 The architecture of query structure tree

受到 ReFSQL^[15] 框架的启发, 本文采用树编辑距离 (Tree-Edit-Distance, TED) 量化两个反向验证问题的查询逻辑相似程度——其核心优势在于能精准捕捉 QST 的逻辑差异, 契合范式复用对“结构一致性”的核心需求。TED 的形式化定义如下:

$$TED(T_1, T_2) = \min_{S \in \mathcal{S}(T_1, T_2)} \{ \Gamma(S) \} \quad (4)$$

其中, $S \in \mathcal{S}(T_1, T_2)$ 表示将查询结构树 T_1 转换为 T_2 的所有可能修改操作结合 (包括节点插入、删除、替换); $\Gamma(S)$ 表示单个修改操作的代价 (以操作步骤数计); $\min_{S \in \mathcal{S}(T_1, T_2)} \{ \Gamma(S) \}$ 为所有修改操作中的最小代价, 直观反映两棵树的逻辑相似性: TED 值越小, 查询逻辑越相近; TED 值越大, 逻辑差异越显著。

2.3 方法流程

本文方法的流程如图2所示, 在进入核心验证流程前, 系统首先完成 Text-to-SQL 基础前置过程: 接收用户问题 q , 经由基于 LLM 的 Text-to-SQL 基础过程生成初始 SQL 语句 sql_0 , 在查询数据库中执行该查询语句以获取初始查询结果 r_0 , 并以此初始查

询结果为基础, 进行系统的核心验证与优化流程。

2.3.1 生成反向问题与验证标准

系统以初始查询结果 r_0 为基础, 并结合用户问题 q 中的约束条件, 利用 LLM 生成一个或多个“反向自然语言问题”及其对应的验证标准。这个生成的反向问题集合 Q_{rev} 可以形式化地表示为:

$$Q_{rev} = \{ (q_{rev_i}, C_i) \}_{i=1}^n \quad (5)$$

其中, C_i 是为 q_{rev_i} 量身定制的、可被自动判定的验证标准。反向问题的生成数量 n 由 LLM 根据查询中识别出的潜在语义易错点自适应决定。该机制遵循“一对一”原则, 即为每一个识别出的易错点生成一个对应的反向验证问题。这种设计旨在确保验证的完备性与非冗余性达到平衡, 既避免了因验证不足导致逻辑谬误遗漏, 也防止了因问题冗余而引发不必要的性能开销。

此外, 为解决多个反向问题间可能存在的逻辑冲突, 本文在生成反向问题的提示工程中明确设计了“关键执行约束”。该约束规定: 每个反向问题的验证标准必须是独立的, 严禁反向问题之间交叉依赖或依赖其他反向问题的执行结果。每个问题仅聚焦于其对应的特定易错点进行原子化验证, 其判断逻辑仅依赖于正向 SQL 执行结果与当前反向问题的查询结果的组合, 从而从机制上确保了验证过程的独立性, 避免了冲突。

表1展示了反向问题生成的提示词结构框架, 该提示词采用“角色定义-核心任务-核心指导原则-关键执行约束”的结构化模板。

2.3.2 基于查询范式演进机制生成反向 SQL 语句

系统遍历上一步生成的验证集 Q_{rev} 。针对集合中的每一个元组 (q_{rev_i}, C_i) , 借助融合查询范式演进机制的 Text-to-SQL 模块, 将反向自然语言问题 q_{rev_i} 转换为对应的反向 SQL 查询语句 sql_{r_i} 。详细流程如图3所示。

为缓解系统初始无历史数据时的“冷启动”问题, 查询范式演进机制具有一套前置过程: 在系统首次部署时, 选取若干组高频查询范例 (如股票查询场景中的“收盘价查询”“涨幅统计”) 与领域特定核心查询 (如金融领域的“市盈率筛选”“成交量排序”), 预填充至查询范式库 $\mathcal{P}$ 。该前置过程旨在加速系统“初始预热”, 确保即使用户查询模式尚未被充分学习, 也能为常见查询提供高效响应, 规避冷启动阶段的性能损耗。

在接收到反向自然语言问题 q_{rev_i} 后, 依托 LLM 的结构化解析能力, 采用 7B (B 为 Billion 的缩写, 用

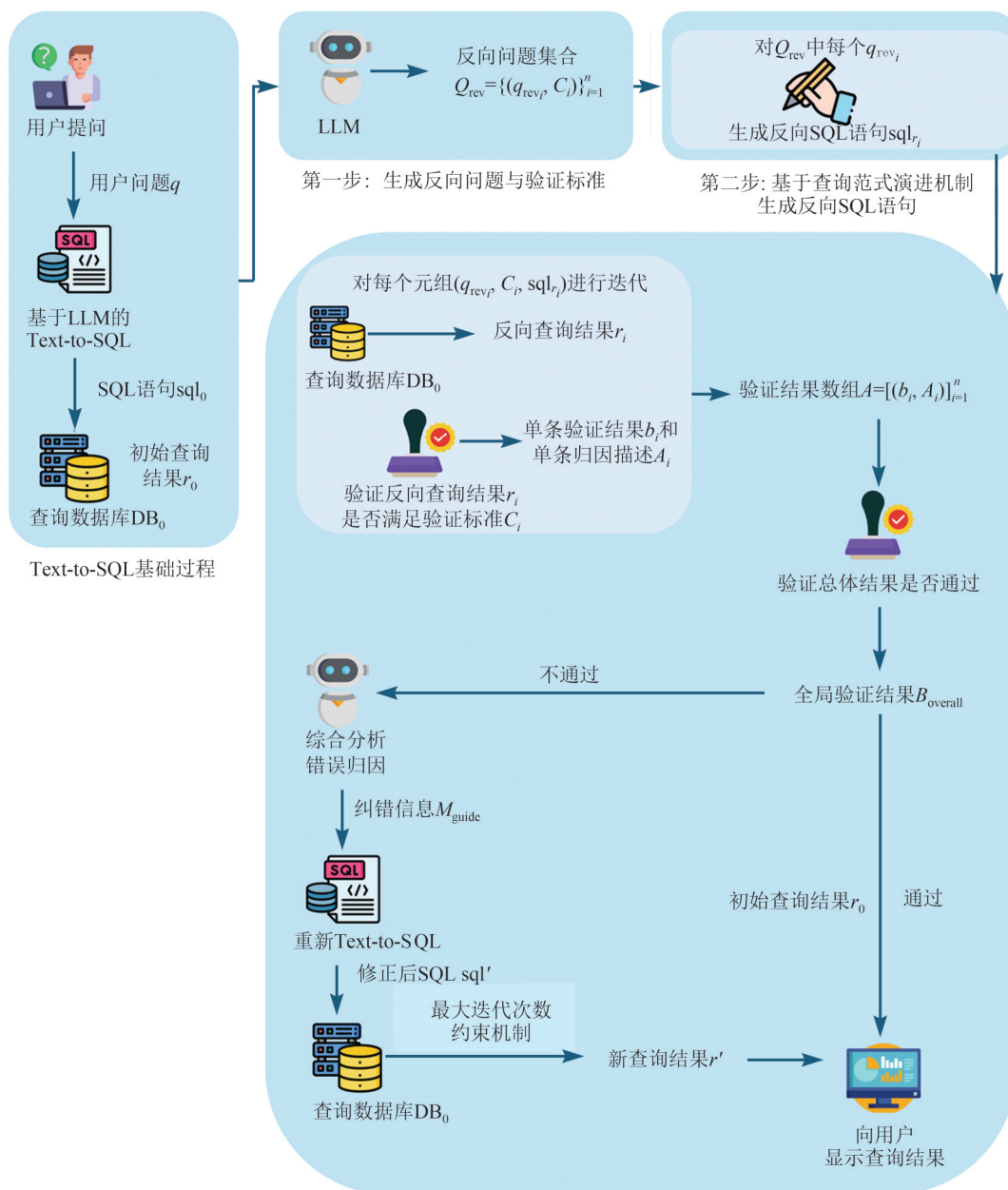


图2 本文方法流程示意图

Fig. 2 Schematic illustration of the proposed methodology

于表示模型参数量级)的模型将 q_{rev} 解析为 QST:

$$T_1 = \text{LLM}(\text{Samples}, q_{\text{rev}}, \text{Instruction}) \quad (6)$$

其中, Instruction 为 QST 的定义规范, 明确节点类型、边的逻辑关系及输出格式约束; Samples 为经典转换示例(含自然语言查询、对应 QST 及逻辑说明), 辅助模型理解结构化输出要求; LLM() 代表模型的推理; T_1 为转换后的 QST, 以嵌套 JSON 格式输出(便于后续树编辑距离计算与范式匹配)。

选择 7B 模型的核心考量是准确性与效率的工程平衡, 具体依据如下:

1) 规避高成本模型的性能损耗: 查询范式演进机制的初衷是降低反向验证的开销, 若 QST 转换依

赖大型模型(模型参数规模大于 70 B), 会引入巨额计算资源消耗与延迟, 违背机制设计初衷;

2) 满足结构化解析的指令遵循要求: QST 转换是高度结构化任务, 需模型同时理解查询逻辑与复杂输出约束(如嵌套 JSON 格式)。参数量过小的模型(1B-3B)指令遵循能力不足, 会导致 QST 节点缺失、逻辑关系错乱, 进而引发后续树编辑距离(TED)匹配失效。

随后, 系统计算当前问题 T_1 与查询范式库 $\mathcal{P}$ 中所有已有历史范式 T_{p_i} 的 TED 距离。若存在满足 $\text{TED}(T_1, T_{p_i}) < \tau$ (对于不同类型的 Text-to-SQL 任务, 阈值 τ 通过经验验证确定) 的相似范式, 系统直

表 1 反向问题生成提示词设计示例
Table 1 Example of prompt design for reverse question generation

部分	具体内容
角色定义	你是一个“反向问题”生成专家,负责 Text-to-SQL 框架的验证修正机制。
核心任务	你的任务是根据提供的正向查询、SQL、执行结果及数据库模式,生成一个或多个“反向自然语言问题”。这些问题必须能全面覆盖正向 SQL 可能存在的关键易错点(如聚合运算、条件筛选、连接逻辑等),以逆向验证正向 SQL 的语义正确性。
核心指导原则	(此处仅列举一些核心原则,可根据具体业务领域做出调整与细化) 原则一:聚合与运算错误 策略:针对聚合结果(如 SUM, AVG),设计一个反向问题,通过改变计算路径或交叉验证的方式来核实该结果的准确性。(附带一些示例,此处省略) 原则二:条件理解与映射错误 策略:针对 WHERE 或 HAVING 子句中的关键条件(如日期范围、特定值约束、“至少”等),设计反向问题来专门验证该条件的满足情况,或查询其对立面以寻找反例。(附带一些示例,此处省略) 原则三:结构性错误与意图误解 策略:如果正向 SQL 返回了错误的列(如将“国籍”查为“货币”)或错误的结构(如返回列表而非总数),反向问题应直接查询正确的列或结构,通过对比结果差异来定位错误。(附带一些示例,此处省略)
关键执行约束	为保证验证的原子性和可靠性,提示词中明确强调了以下“最重要”的执行规则: 独立验证:每个反向问题的验证标准必须是独立的,绝对禁止反向问题之间交叉依赖或依赖其他反向问题的执行结果。 数据局限性:验证标准的判断逻辑,必须仅依赖于正向 SQL 执行结果和当前反向问题的查询结果的组合,不得引入任何需要额外查询才能获得的先验知识。
输出格式	{ "RNLQ_nums": "反向问题的总数", "RNLQs": { "RNLQ1": { "Question": "", "Verify_Standard": "" }, // ... 可包含 RNLQ2, RNLQ3 ... } }

接复用其对应的 SQL 模板并进行微调,显著规避了全量生成的资源消耗:

$$\text{sql}_{r_i} = \text{LLM}(\text{question}_{\text{old}}, \text{sql}_{\text{old}}, q_{\text{rev}_i}, \text{Instruction}_1) \quad (7)$$

其中, $\text{question}_{\text{old}}$ 代表相似范式的自然语言问题, sql_{old} 代表相似范式的 SQL 语句, Instruction_1 为预定义的转换规则提示词模板。

若不存在相似范式,系统则调用全量 Text-to-SQL 模块生成 SQL,并将该“问题-结构-SQL”三元组作为新范式写入范式库:

$$\mathcal{P}_{i+1} = \mathcal{P}_i \cup (q_{\text{rev}_i}, T_1, \text{sql}_{r_i}) \quad (8)$$

其中, $\mathcal{P}_i$ 为更新前的查询范式库, $\mathcal{P}_{i+1}$ 为更新后的查询范式库。

另外,为了解决大量极度不相似的“一次性”查询可能导致的范式库膨胀问题,本机制引入范式库驱逐策略。具体而言,当需要进行范式库写入时,若

范式库未满,则直接将新范式($q_{\text{rev}_i}, T_1, \text{sql}_{r_i}$)写入;若范式库已满,则触发最不使用(Least Frequently Used, LFU)策略驱逐范式:计算各范式近 30 天的调用频次,移除频次最低的条目;若频次相同,则移除最早创建的范式。该策略可以为范式库设置一个合理的规模上限,有效控制范式库的规模,保障机制长期运行的稳健。

2.3.3 全局裁决与生成修正指令

对于每个 sql_{r_i} , 执行得到反向查询结果 r_i 并和预设的验证标准 C_i 进行比对。若一致,则保存单条验证结果 b_i 为 True; 若不一致,则保存单条验证结果 b_i 为 False, 并从意图理解偏差、实体映射误差、条件逻辑疏漏等维度推断结果不一致的根本原因并保存为单条归因描述 A_i 。对所有的 sql_{r_i} 遍历后,得到一个结构化分析结果数组,其定义如下:

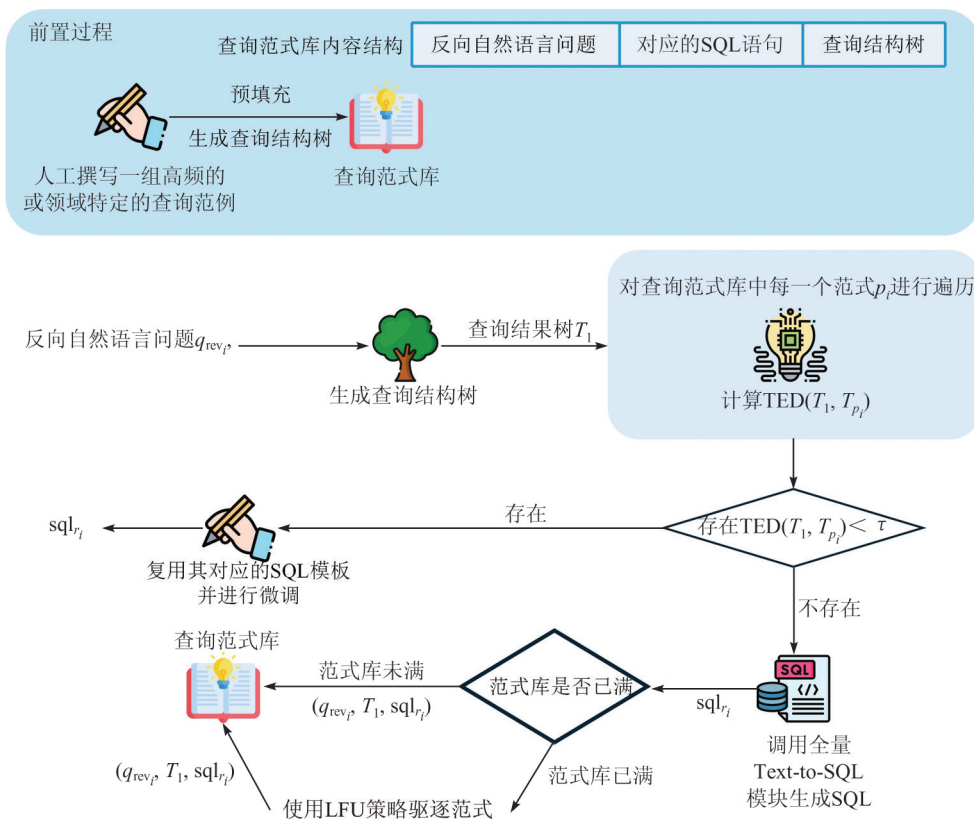


图3 基于查询范式演进机制生成反向SQL语句的内部工作过程

Fig. 3 Internal working process of generating reverse SQL statements based on the query paradigm evolution mechanism

$$A = [(b_i, A_i)]_{i=1}^n \quad (9)$$

其中, b_i 是一个布尔标志 (True 表示该项验证一致, False 表示不一致); A_i 是归因描述字符串 (仅当 $b_i = \text{False}$ 时, 明确给出具体错误原因, 如“反向结果未包含原始查询‘沪深300成分股’的约束条件, 推测初始SQL遗漏了板块筛选逻辑”)。

接下来, 系统对上面输出的结构化分析数组 A 进行整体评判, 通过对所有布尔标志 b_i 执行逻辑与运算, 得出全局验证结果 B_{overall} , 运算公式如下:

$$B_{\text{overall}} = \bigwedge_{i=1}^n b_i \quad (10)$$

若 $B_{\text{overall}} = \text{True}$, 表明所有反向验证均一致, 初始SQL查询 sql_0 完全契合用户意图, 可直接将初始查询结果 r_0 反馈给用户; 反之, 即 $B_{\text{overall}} = \text{False}$, 则由LLM聚合所有 $b_i = \text{False}$ 对应的归因描述 A_i , 提炼形成一条宏观的、具有明确指导性的“纠错信息” M_{guide} (如“初始SQL存在两处语义偏差: 一是误将‘近30天’映射为‘自然月’, 二是遗漏‘行业为新能源’的筛选条件”)。该纠错信息将反馈至Text-to-SQL模块, 引导其生成一条修正后的SQL查询 sql' , sql' 将重新进入上述反向验证过程进行迭代验证, 为防止模型在复杂或存在歧义的查询场景下陷入无

限纠错循环, 系统为此迭代过程设定了最大次数 (如3次) 约束作为强制终止条件: 若在限定次数内验证通过, 则直接输出修正后的新查询结果 r' ; 若超限仍未通过全局验证, 系统将触发兜底机制强制回退至初始语句 sql_0 , 并向用户输出包含逻辑风险提示的最终结果, 从而在兼顾语义纠错深度的同时, 有效确保了系统工程落地时的鲁棒性与响应速度。

通过这种“生成-验证-评审-修正”的闭环协作, 反向验证增强机制实现了对查询正确性的高效验证, 更达成了语义错误的自主定位与精准修正, 提升了Text-to-SQL的可靠性。图4以具体查询案例呈现了反向验证增强机制的工作逻辑。

3 实验结果与分析

3.1 实验环境

本研究的实验平台由本地物理服务器部署与云端API调用两部分协同构成。具体的软硬件配置及模型部署细节如表2所示。

在模型推断阶段, 针对参数量为7B的基线Text-to-SQL模型, 利用本地GPU服务器进行统一部署与推理计算; 而对于反向验证增强机制中具有较高逻辑

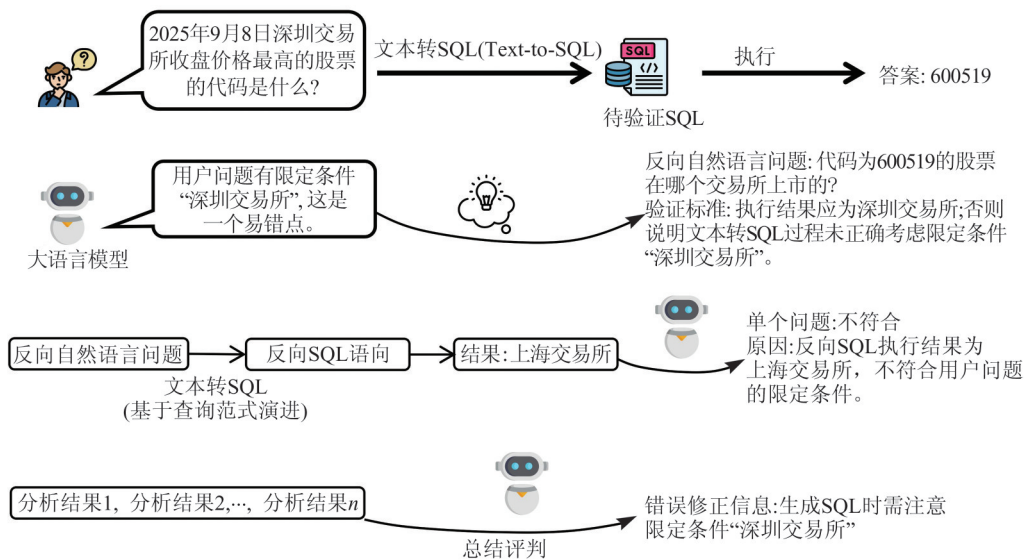


图4 反向验证增强机制工作逻辑

Fig. 4 Logic of the reverse verification enhancement mechanism

表2 本文实验的软硬件环境与模型配置

Table 2 Software and hardware environments and model configurations of the experiments

类别	详细配置	部署方式与用途
硬件环境	CPU: Intel Xeon(Skylake) Platinum 8163	本地模型部署与推理计算
	内存: 64 GB	
	GPU: 2 × NVIDIA V100	
软件环境	操作系统: Ubuntu 20.04	提供基础运行环境、模型推理支持与 SQL 语句执行环境。
	编程语言: Python 3	
	推理环境: vLLM 0.11.0	
	数据库引擎: sqlite3	
云端模型	硅基流动服务	云端 API 接入

推理要求的环节(即反向问题生成与正反向结果一致性比对),则通过调用 DeepSeek-V3 的云端 API 完成。

3.2 实验数据集和评价指标

Spider^[17]是一个大规模跨领域语义解析基准数据集,旨在推动面向数据库的自然语言交互技术发展。该数据集包含 10 181 个人工标注的问题,对应 5 693 个独特的复杂 SQL 查询,覆盖金融、医疗、电商、教育等 138 个领域的 200 个多表数据库。官方通过对问题和 SQL 样本进行随机拆分,并对多表数据库进行完整分组划分,得到 Spider-dev、Spider-test 等子集。其 1.0 版本通过确保训练集与测试集在 SQL 查询和数据库模式上零重叠,确立了严格评估机制,要求模型必须能够泛化到未见过的查询结构和未知数据库模式。

BIRD(大规模数据库驱动文本到 SQL 评测基准)^[18]是一个开创性的跨领域数据集,涵盖金融、医疗、电商、教育等 37 个专业领域,旨在探究大规模数据库内容对文本到 SQL(Text-to-SQL)解析性能的影响。

本文的实验采用了其中的 Mini-Dev 集(即 BIRD Mini-Dev),这是一个由社区共同驱动构建的子集,包含了源自 11 个异构数据库的 500 个高质量文本到 SQL 样本。这一资源高效的评测集不仅便于进行可复现的测试,同时也捕捉了 SQL 生成任务中的核心挑战。

执行准确率(Execution Accuracy, EX)通过对比标准答案 SQL 查询与模型预测 SQL 查询的执行结果集是否完全一致,评估自然语言转 SQL 系统的性能,是 Text-to-SQL 领域通用的评价指标。EX 的数学定义如下:

$$EX = \frac{1}{N} \sum_{i=1}^N \mathbb{I}(\text{Exec}(\text{SQL}_{\text{pred}}^i) = \text{Exec}(\text{SQL}_{\text{gold}}^i)) \quad (11)$$

其中, N 为测试集的样本总数; i 为样本索引; $\text{SQL}_{\text{pred}}^i$ 和 $\text{SQL}_{\text{gold}}^i$ 分别表示第 i 个样本的模型预测 SQL 语句与标准答案 SQL 语句; $\text{Exec}(\cdot)$ 为数据库查询语句的执行,用于返回 SQL 语句执行后的结果集; $\mathbb{I}(\cdot)$ 是指示函数,当预测 SQL 与标准 SQL 的执行结果集

完全一致时取 1;否则取 0,最终通过对所有样本的指示函数值取平均得到执行准确率。

3.3 反向验证增强机制执行准确率实验

3.3.1 组件分析实验

为验证本文所提出的反向验证增强机制的有效性,开展了组件分析实验。在集成反向验证增强机制时,反向问题的生成以及正反向查询结果的逻辑比对均统一采用 DeepSeek-V3 模型完成。为确保验证的全面性,实验选取了代表当前 3 种不同主流范式的基线模型:代表多生成器协同框架的前沿模型 XiYan-SQL-7B^[3]、代表监督微调(SFT)范式的 CodeS-7B^[13],以及代表强化学习优化范式的

Arctic-Text2SQL-R1-7B^[19]。实验分别对比了 3 个基线模型(模型)与集成该机制后的模型(模型+反向验证增强机制)在 Spider-test 和 BIRD Mini-Dev 上的执行准确率,实验结果如表 3 所示。

表 3 的结果表明,反向验证增强机制作为一种后处理模块,对所有测试的基线模型均带来了显著的性能提升。在 Spider-test 上,该机制使各模型的执行准确率最高提升了 5.92 个百分点。而在更具挑战性的 BIRD Mini-Dev 集上,这一提升效果更为明显,准确率增幅最高提升了 8.60 个百分点。这说明本文所提出的“正向查询-反向验证”闭环校验逻辑,能有效识别并修正传统方法难以发现的复杂语义错误。

表 3 集成反向验证机制前后在 Spider-test 和 BIRD Mini-Dev 集上的执行准确率对比
Table 3 Comparison of execution accuracy on the Spider-test and BIRD Mini-Dev before and after integrating the reverse validation mechanism

模型	Spider-test	BIRD Mini-Dev	%
XiYan-SQL-7B ^[3]	80.95	58.00	
XiYan-SQL-7B ^[3] +反向验证机制	85.21	66.60	
CodeS-7B ^[13]	81.90	55.20	
CodeS-7B ^[13] +反向验证机制	86.51	63.40	
Arctic-Text2SQL-R1-7B ^[19]	77.92	56.60	
Arctic-Text2SQL-R1-7B ^[19] +反向验证机制	83.84	60.20	

3.3.2 基线方法对比实验

为系统评估反向验证机制的有效性与优越性,本文选择 Text-to-SQL 领域的代表性模型 XiYan-SQL-7B^[3]作为基线模型,对比了现有代表性增强方法(CSC-SQL^[20]、Alpha SQL^[21])与反向验证增强方法在 BIRD Mini-Dev 集上的执行准确率。该数据集在完整保留了 BIRD 基准中异构数据库、复杂业务逻辑等深层技术挑战的同时,具备更为紧凑的样本规模,从而便于在有界的计算资源下开展轻量高效、高度可复现的端端增强机制评测。实验结果如表 4 所示。

表 4 的实验结果表明,本文提出的反向验证增强机制在执行准确率上优于对比方法。尽管绝对数值的提升幅度有限,但在高基线水平与 BIRD

Mini-Dev 数据集极具挑战性的双重约束下,这一进步依然具有重要的实际价值。这一优势的根本原因在于后处理验证假设的本质差异:CSC-SQL 和 Alpha-SQL 的核心评价逻辑均高度依赖于“自一致性”(即分别通过多数表决和一致性奖励来筛选评估 SQL)。然而,在面对复杂约束查询时,LLM 极易产生系统性的语义偏差,从而“一致地”生成包含相同逻辑错误的语句,导致基于投票或一致性评估的方法陷入“一致性幻觉”且无法纠错。相比之下,本文的反向验证机制彻底打破了对候选共识的依赖,创新性地将正向查询结果作为已知条件,逆向检验其是否满足原始自然语言的特定约束。这种基于逻辑闭环的反转校验策略,能够有效识别并修正那些语法正确但语义偏离的隐蔽错误,从而在复杂的真实查询场景中实现了更高的准确率。

3.4 树编辑距离阈值(τ)敏感性分析

为了科学界定 QST 的 TED 匹配阈值 τ ,本文在 Spider 数据集上进行了阈值敏感性分析与测试。Spider 数据集具有极高的查询结构复杂性与跨领域零重叠特性,能够提供丰富的逻辑结构梯度,从而客观呈现不同阈值下的匹配效果。

采用重生准确率 Acc_{regen} (即基于相似范式成功

表 4 不同增强方法下 XiYan-SQL-7B 模型在 BIRD Mini-Dev 集上的执行准确率对比

Table 4 Comparison of execution accuracy on the BIRD Mini-Dev of XiYan-SQL-7B with different enhancement methods

模型	EX/%
XiYan-SQL-7B ^[3] +CSC-SQL ^[20]	65.20
XiYan-SQL-7B ^[3] +Alpha SQL ^[21]	66.20
XiYan-SQL-7B ^[3] +反向验证机制	66.60

修正错误的比例)作为评价指标,其数学定义如下:

$$\text{Acc}_{\text{regen}} = \frac{N_{\text{corrected}}}{N_{\text{error}}} \times 100\% \quad (12)$$

其中, N_{error} 表示初始推断错误的样本总数; $N_{\text{corrected}}$ 表示在 N_{error} 个错误样本中,通过引导模型复用相似范式,成功改写并生成正确 SQL 语句的样本数量。

具体实验流程如下:首先,将 Spider-dev(作为候选查询范式库)与 Spider-test(作为测试集)中的自然语言问题全量转化为 QST。这种划分方式有效模拟了真实系统中“历史范式积累”与“未知新请求”的物理隔离场景,从根本上规避了同源数据可能导致的数据泄露问题。其次,利用 XiYan-SQL-7B^[3]对 Spider-test 进行初始 Text-to-SQL 推断,筛选出生成错误的样本构建待修正集。选择 XiYan-SQL 作为推断基线,是因为其代表了当前前沿的生成水平,生成的错误多为复杂的隐性语义偏差而非低级语法错误,能够更真实地检验阈值设定对深层语义纠错能力的实际影响。随后,针对该集合中的每个问题,设定不同的 TED 阈值 τ ,在 Spider-dev 中检索 Top-5 最相似的历史查询范式,并引导 7B 小参数模型基于检索到的范式进行 SQL 的改写与重生。测试结果如表 5 所示。

表 5 不同树编辑距离阈值 τ 对重生成准确率的影响

Table 5 The impact of different tree edit distance thresholds on regeneration accuracy

τ	$\text{Acc}_{\text{regen}}/\%$
1	98.20
2	95.50
3	90.30
> 3	62.40

如表 5 的测试结果所示,LLM 基于查询范式的重生成准确率与树编辑距离阈值 τ 呈现出显著的负相关趋势。当 $\tau \leq 3$ 时,由于检索到的历史查询范式在逻辑结构上与当前待修正问题高度相似,系统能够维持 90.30% 以上的高重生成准确率(在 $\tau = 1$ 时最高,可达 98.20%);然而,一旦阈值 $\tau > 3$,匹配条件过度放宽导致检索到的范式与目标问题存在较大的结构差异,从而向大模型引入了严重的语义噪声,使得重生成准确率出现断崖式下跌(降至 62.40%)。这一结果充分表明,为确保查询范式演进机制在 SQL 改写阶段的可靠性,必须将树编辑距离阈值严格控制在合理区间(如 $\tau \leq 3$),以有效屏蔽结构异质性对模型推理造成的干扰。

3.5 动态查询范式演进机制查询效率实验

为了量化评估查询范式演进机制对反向验证机

制查询效率的提升作用,本文设计了一项基于时间模型的实验。该实验的核心在于对比引入范式库前后,系统在不同查询命中率下的平均响应时间,从而验证该机制在处理重复性查询模式时的加速效果。

本文未在静态数据集上做全量实际耗时测试,是为规避特定查询样本的随机性偏差,由于不同 SQL 执行耗时差异极大,直接测试结果会严重依赖查询复杂度,无法客观反映机制本身的真实增益。本研究采用“核心模块平均耗时实测+理论命中率区间仿真”的评估方法,严格控制变量并剥离单一数据集分布局限,能更科学普适地展现范式演进机制的效率提升效能与理论边界。

首先对 BIRD Mini-Dev 数据集中的数据进行推理计算,获取以下流程的平均消耗时间,以代表不同操作的时间成本:

1) Text-to-SQL 转换:代表调用大型语言模型(LLM)进行一次完整的自然语言到 SQL 转换。

2) 范式匹配:代表从查询范式库中检索相似查询。

3) 范式修改:代表对已有范式进行修改以得到用户所需要的 SQL 语句。

4) SQL 执行:代表 SQL 语句的执行过程。

测得的平均推理计算时间如表 6 所示。基于表 6 的结果,可以计算出不同场景下的平均总查询时间:

1) 基准模型(无范式库):任何查询都必须经过完整的 Text-to-SQL 转换和执行,总时间为 7.3 (Text-to-SQL)+0.4(执行)=7.7 s。

2) 引入范式库(命中时):查询通过范式匹配直接找到 SQL 模板,总时间为 1.2(匹配)+1.7(范式修改)+0.4(执行)=3.3 s。

3) 引入范式库(未命中时):查询需要先进行一次失败的范式匹配,然后再回退到 NL2SQL 流程,总时间为 1.2(匹配)+7.3(Text-to-SQL)+0.4(执行)=8.9 s。

表 6 不同操作的平均消耗时间

Table 6 Average time consumption of different operations

操作	平均消耗时间/s
Text-to-SQL 转换	7.3
范式匹配	1.2
范式修改	1.7
SQL 执行	0.4

本文使用命中率(Hit Rate)衡量范式库对用户意图的覆盖能力,将命中率定义为:在系统接收到的所有自然语言查询请求中,能够成功在查询范式

库中检索并匹配到相似历史范式(即两者的 QST 编辑距离小于预设阈值 τ) 的查询所占的比例。

通过调整范式库的命中率,观察平均查询时间与时间节省百分比的变化,结果如表 7 所示;当命中率为 0% 时,系统的平均查询时间为 8.9 s,比基准模型的 7.7 s 要慢 1.2 s。这清晰地表明,范式匹配步骤本身存在一个固定的时间开销。对于完全没有重复查询的场景,该机制会引入额外的性能负担。当命中率达到 40% 时,系统的平均查询时间降至 6.7 s,首次实现了显著的性能提升,比基准模型的查询时间快 1.0 s。这说明只要用户的查询模式存在一定程度的重复性,该机制就能带来正向收益。随着命中率从 40% 提升至 100%,相对基准模型节省的时间从 1.0 s 增长至 4.4 s。尤其是在高命中率场景下,平均查询时间可以被压缩至基准的一半以下,展现了查询范式演进机制在处理常规化、高频化查询请求时的巨大优势。

表 7 不同命中率下查询时间	
Table 7 Query time under different hit rates	
命中率/%	平均查询时间/s
0	8.9
20	7.8
40	6.7
60	5.5
80	4.4
100	3.3

4 系统实现与工程化验证

对本文方法进行工程化验证,本文设计并实现了基于反向验证增强的股票信息查询 Agent 以及基于此 Agent 形态的、基于多层分离架构的智能选股平台。该系统的架构分为表现层、应用层与数据

层,各层技术选型明确,职责清晰。

系统的表现层,即用户交互界面,采用 React 框架进行构建。React 以其声明式的 UI 和组件化的开发模式,为构建高效、动态的单页面应用提供了有力支持,能够为用户提供流畅的自然语言对话式查询体验,并对返回的结构化数据进行实时的可视化呈现。

应用层是系统的核心,其后端服务基于企业级的 Django 框架搭建。Django 负责处理所有的业务逻辑,包括实现 RESTful API 端点、管理应用状态与用户会话。Agent 的认知与推理能力则由 LangChain 框架进行编排与实现,它被用于构建和管理执行“动态查询范式演进”与“反向验证增强”两大核心机制所需的、与 LLM 交互的复杂调用链。

数据层采用双库模式:系统的查询范式库通过 Django ORM 进行管理,并持久化存储于关系型数据库中,以支持高效的增删改查操作;而海量的股票行情数据则存储于 ClickHouse 数据仓库中。整个系统通过各层之间的 API 调用进行解耦与通信,最终形成了一个模块化、可扩展且高性能的智能 Agent 应用实例。

系统的实现效果见图 5。该界面展示了本文所构建的股票信息查询 Agent 的实际应用效果:系统成功接收并理解了用户的自然语言查询(“2025 年 9 月 8 日收盘价格最高的上证所上市的股票”),并生成了正确的、包含多重连接与复杂条件(如交易所、日期、排序和限制)的 SQL 语句,最终精准地从数据库中检索到了目标股票(“贵州茅台”)。这直观验证了本文方法在真实工程场景下的可行性与有效性。

为了进一步量化验证本文方法在实际工程落地中的场景适应性与综合性能,在所构建的智能选股平台上开展了小范围的内部系统测试。依据真实金融业务场景的用户交互习惯,本文收集并筛选了 150 条高频自然语言查询样本,将其划分为 3 个

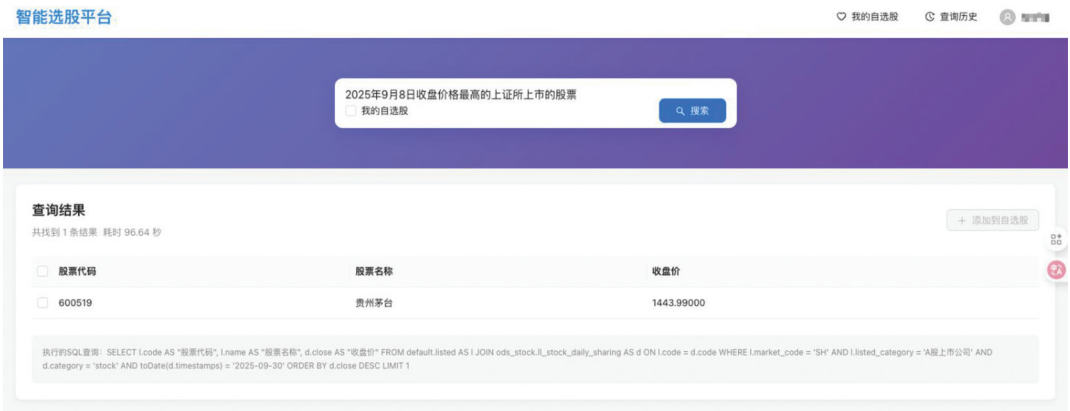


图 5 智能选股平台实现效果

Fig. 5 Implementation of the intelligent stock-picking platform

核心业务场景:基本面选股类(如“查询市盈率低于15且净利润连续三年增长的股票”)、技术面选股类(如“今日收盘价突破60日均线的科技股有哪些”)以及统计与组合筛选类(如“统计各行业板块涨幅前三的股票及其平均市值”)。

实验将部署传统 Text-to-SQL 模块(即未引入反向验证与范式演进机制的基线系统)与完整部署本文架构的增强系统进行了端到端的性能对比。测试指标涵盖最终执行准确率(EX),具体测试结果如表8所示。

表8 系统性能测试表
Table 8 System performance test table

查询场景类别	样本量	执行准确率/%	
		基线系统	本文系统
基本面选股类	60	83.33	96.70
技术面选股类	50	90.00	94.00
统计与组合筛选类	40	87.50	97.50
整体	150	86.70	96.00

如表8的测试结果所示,本文系统在智能选股平台的三大类(基本面选股类、技术面选股类和统计与组合筛选类)真实业务查询场景中,整体执行准确率相比基线系统从86.70%提升至96.00%。具体来看,在涉及多条件组合与复杂逻辑的“基本面选股类”和“统计与组合筛选类”任务中,本系统的执行准确率分别实现了13.37个百分点和10.00个百分点的大幅跨越,充分验证了反向验证机制在处理复杂语义纠错时的卓越效能;而在基础执行准确率本就较高的“技术面选股类”任务中,本文方法依然带来了4.00个百分点的稳步提升。这表明,本文提出的优化生成方法不仅具备极高的执行准确性,且在金融垂直领域的复杂工程化场景中具有极强的鲁棒性与多场景适应能力。

5 结 语

本文针对 LLM 在 Text-to-SQL 任务中面临的语义理解偏差与执行效率挑战,提出并实现了一种创新的基于反向验证增强的 SQL 优化生成方法。本研究的理论与工程贡献与实验结果紧密印证:在理论层面设计了正反双向验证的闭环行为规则,有效捕捉并修正了传统静态语法检查难以覆盖的隐蔽语义错误。实验数据表明,该机制在 Spider-test 和 BIRD Mini-Dev 数据集上将各基线模型的执行准确率分别最高提升 5.92 和 8.60 个百分点;其次,针对反向验证引入的算力与时间开销,构建了基于

QST 的动态查询范式演进机制,将高频用户的查询模式显式化为可复用范式。仿真实验证实,在 100% 命中率场景下可减少 57% 的反向验证耗时。在工程实践上,本文基于上述两大机制成功搭建了具备持续学习能力的智能选股 Agent 及应用平台,为 Agent 技术在垂直领域的“高准确率-高效率”自优化落地提供了全新的设计范式。

尽管本文方法在准确率与效率上获得了显著提高,但机制本身仍存在固有局限性。一方面,反向验证的质量高度依赖基座大模型零样本生成反向问题的能力,面对嵌套层次极深或包含复杂领域业务逻辑的 SQL 查询时,生成的反向问题可能缺乏足够的鉴别力或产生逻辑发散;另一方面,范式演进机制中基于 TED 的匹配策略主要依赖浅层的句法结构,在处理句法差异巨大但语义完全等价的极端查询时,匹配精度与泛化性仍有待提升。

针对上述局限性,未来的工作将围绕以下两个维度展开:一是引入强化学习框架以突破反向问题生成器的能力瓶颈。计划将反向验证过程中的反馈信号(如错误识别的成功率、验证结果与真实意图的对齐度)建模为奖励函数,持续微调生成模型,使系统自主学习生成更具“针对性”的高质量反向问题,从源头上提高验证闭环的鲁棒性。二是探索基于图神经网络或深层语义图拓扑的表征方法来替代现有的 QST,以进一步增强同义查询范式的检索命中率与复杂场景下的复用准确度。

参考文献:

- [1] LIU X Y, SHEN S Y, LI B Y, *et al.* A survey of text-to-SQL in the era of LLMs: Where are we, and where are we going [J]. *IEEE Transactions on Knowledge and Data Engineering*, 2025, **37**(10): 5735-5754. DOI:10.1109/TKDE.2025.3592032.
- [2] POURREZA M, LI H L, SUN R X, *et al.* CHASE-SQL: Multi-path reasoning and preference optimized candidate selection in text-to-SQL [EB/OL]. 2024: arXiv: 2410.01943. <https://arxiv.org/abs/2410.01943>
- [3] LIU Y F, ZHU Y, GAO Y Q, *et al.* XiYan-SQL: A novel multi-generator framework for text-to-SQL [J]. *IEEE Transactions on Knowledge and Data Engineering*, 2026, **38**(4): 2474-2487. DOI:10.1109/TKDE.2026.3657851.
- [4] POURREZA M, RAFIEI D. DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction[C]// *Advances in Neural Information Processing Systems* 36. New Orleans: Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2023: 36339-36348. DOI:10.52202/075280-1577.

- [5] GU Z H, FAN J, TANG N, *et al.* Interleaving pre-trained language models and large language models for zero-shot NL2SQL generation[EB/OL]. 2023; arXiv: 2306.08891. <https://arxiv.org/abs/2306.08891>.
- [6] LIU X Y, SHEN S Y, LI B Y, *et al.* NL2SQL-BUGs: A benchmark for detecting semantic errors in NL2SQL translation[C]//Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2. Toronto ON Canada. New York: ACM, 2025: 5662–5673. DOI:10.1145/3711896.3737427.
- [7] WANG X Z, WEI J, SCHUURMANS D, *et al.* Self-consistency improves chain of thought reasoning in language models[EB/OL]. 2022; arXiv: 2203.11171. <https://arxiv.org/abs/2203.11171>.
- [8] GAO D W, WANG H B, LI Y L, *et al.* Text-to-SQL empowered by large language models: A benchmark evaluation[J]. *Proceedings of the VLDB Endowment*, 2024, **17**(5): 1132–1145. DOI:10.14778/3641204.3641221.
- [9] ZHANG C, MAO Y R, FAN Y J, *et al.* FinSQL: Model-agnostic LLMs-based text-to-SQL framework for financial analysis [C]//Companion of the 2024 International Conference on Management of Data. New York: ACM, 2024: 93–105. DOI:10.1145/3626246.3653375.
- [10] RENZE M. The effect of sampling temperature on problem solving in large language models [C]//Findings of the Association for Computational Linguistics: EMNLP 2024. Cambridge: Association for Computational Linguistics, 2024: 7346–7356. DOI:10.18653/v1/2024.findings-emnlp.432.
- [11] LI Z S, WANG X, ZHAO J J, *et al.* PET-SQL: A prompt-enhanced two-round refinement of Text-to-SQL with cross-consistency [C]//Database Systems for Advanced Applications. Singapore: Springer, 2026: 193–208. DOI:10.1007/978-981-95-3830-0_13.
- [12] TALAEI S, POURREZA M, CHANG Y C, *et al.* CHES: Contextual harnessing for efficient SQL synthesis [EB/OL]. 2024; arXiv: 2405.16755. <https://arxiv.org/abs/2405.16755>.
- [13] LI H Y, ZHANG J, LIU H B, *et al.* CodeS: Towards building open-source language models for text-to-SQL [J]. *Proceedings of the ACM on Management of Data*, 2024, **2**(3): 1–28. DOI:10.1145/3654930.
- [14] XIANG Y Z, ZHANG Q W, ZHANG X, *et al.* G³R: A graph-guided generate-and-rerank framework for complex and cross-domain text-to-SQL generation[C]//Findings of the Association for Computational Linguistics: ACL 2023. Cambridge: Association for Computational Linguistics, 2023: 338–352. DOI:10.18653/v1/2023.findings-acl.23.
- [15] ZHANG K, LIN X X, WANG Y Z, *et al.* ReFSQL: A retrieval-augmentation framework for text-to-SQL generation[C]//Findings of the Association for Computational Linguistics: EMNLP 2023. Singapore: Association for Computational Linguistics, 2023: 664–673. DOI:10.18653/v1/2023.findings-emnlp.48.
- [16] LI F, JAGADISH H V. NaLIR: An interactive natural language interface for querying relational databases[C]//Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. New York: ACM, 2014: 709–712. DOI:10.1145/2588555.2594519.
- [17] YU T, ZHANG R, YANG K, *et al.* Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task[C]//Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing. Cambridge: Association for Computational Linguistics, 2018: 3911–3921. DOI:10.18653/v1/D18-1425.
- [18] CHANG K, CHENG R, CHENHAO M, *et al.* Can LLM already serve as a database interface A big bench for large-scale database grounded text-to-SQLs[C]//Advances in Neural Information Processing Systems 36. New Orleans: Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2023: 42330–42357. DOI:10.52202/075280-1835.
- [19] YAO Z W, SUN G H, BORCHMANN L, *et al.* Arctic-Text2SQL-R1: Simple rewards, strong reasoning in Text-to-SQL[EB/OL]. 2025; arXiv: 2505.20315. <https://arxiv.org/abs/2505.20315>.
- [20] SHENG L, SHUAI X S. CSC-SQL: Corrective self-consistency in text-to-SQL *via* reinforcement learning[C]//Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics. Cambridge: Association for Computational Linguistics, 2025: 1473–1496. DOI:10.18653/v1/2025.findings-ijcnlp.91.
- [21] LI B Y, ZHANG J Y, FAN J, *et al.* Alpha-SQL: Zero-shot text-to-SQL using Monte Carlo tree search[EB/OL]. 2025; arXiv: 2502.17248. <https://arxiv.org/abs/2502.17248>.

□