软件学报 ISSN 1000-9825, CODEN RUXUEW
[doi: 10.13328/j.cnki.jos.007505] [CSTR: 32375.14.jos.007505]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

混合特征编码的 HTAP 工作负载识别方法*

杨建文[1,2], 丁治明[1], 严 瑾[1,2], 张秋鸿[1,2], 朱美玲[1]

[1](中国科学院 软件研究所, 北京 100190)

[2](中国科学院大学, 北京 100049)

通信作者: 丁治明, E-mail: zhiming@iscas.ac.cn

摘 要: HTAP 数据库在一套系统中同时支持 OLTP 和 OLAP 工作负载. 其中工作负载的识别是查询执行中路由分发的关键, 只有准确识别出查询属于 OLTP 或 OLAP, 才能对查询进行合理优化和分配资源. 因此, 准确识别工作负载类型是 HTAP 数据库性能的关键因素之一. 然而, 现有的负载识别方法主要基于 SQL 语句中的规则和成本代价, 以及传统机器学习的方法来区分工作负载. 这些方法没有考虑查询语句的自身特点, 也没有利用执行计划的结构信息, 影响识别工作负载的准确率. 为了提高负载识别的准确性, 提出了一种智能识别 OLTP 和 OLAP 工作负载的方法, 该方法通过对 SQL 语句和执行计划进行特征提取和特征编码, 基于 BERT 构建 SQL 语句编码器, 结合树卷积神经网络和注意力机制构建执行计划的编码器, 两种特征融合构建分类器, 该模型能够智能识别 HTAP 混合负载中的工作负载. 通过实验验证, 模型可以准确识别 OLTP 和 OLAP 工作负载, 具有较高的识别准确率. 同时, 在多种数据集中验证了模型的鲁棒性, 并将模型集成到 TiDB 数据库中验证了其对数据库性能的提升.

关键词: HTAP 工作负载识别; 执行计划编码; 树卷积神经网络; 混合负载分类; 数据库性能优化

中图法分类号: TP311

中文引用格式: 杨建文, 丁治明, 严瑾, 张秋鸿, 朱美玲. 混合特征编码的HTAP工作负载识别方法. 软件学报. http://www.jos.org.cn/1000-9825/7505.htm

英文引用格式: Yang JW, Ding ZM, Yan J, Zhang QH, Zhu ML. HTAP Workload Identification Method Based on Hybrid Feature Coding. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7505.htm

HTAP Workload Identification Method Based on Hybrid Feature Coding

YANG Jian-Wen[1,2], DING Zhi-Ming[1], YAN Jin[1,2], ZHANG Qiu-Hong[1,2], ZHU Mei-Ling[1]

[1](Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

[2](University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: HTAP databases are capable of simultaneously supporting OLTP and OLAP workloads within a set of systems. The workload identification is a critical entry point for routing distribution in query execution. The only way to reasonably optimize the queries and allocate resources is to accurately identify whether a query belongs to OLTP or OLAP. Therefore, accurate identification of workload types is a key factor in the performance of HTAP databases. However, existing workload identification methods are mainly based on rules and cost-based measures in SQL statements, as well as machine learning approaches to differentiate workloads. These methods do not consider the inherent characteristics of query statements and utilize structural information in execution plans, resulting in low workload identification accuracy. To improve workload identification accuracy, this study proposes an intelligent method for identifying OLTP and OLAP workloads. This method extracts and encodes features from SQL statements and execution plans, builds the SQL statement encoder based on BERT, and combines the convolutional neural networks and attention mechanisms to construct the encoder of execution plans, with two types of features integrated to build a classifier. The model enables intelligent identification of workloads in HTAP hybrid workloads. Experimental verification shows that the proposed model can accurately identify OLTP and OLAP workloads with high

* 基金项目: 国家重点研发计划 (2022YFF0503900); 山东省重点研发计划 (2021CXGC010104); 中国科学院软件研究所重大项目 (ISCAS-ZD-202401, ISCAS-ZD-202403)
收稿时间: 2024-06-18; 修改时间: 2024-12-28, 2025-06-16; 采用时间: 2025-07-04; jos 在线出版时间: 2025-11-26

identification accuracy. Additionally, the robustness of the model has been validated across multiple datasets, and the model is integrated into the TiDB database to verify its performance improvement on the database.

Key words: HTAP workload identification; execution plan encoding; tree convolutional neural network; mixed workload classification; database performance optimization

Gartner 公司在 2014 年提出了混合交易/分析处理 (HTAP)[1]这个术语, 即使用一站式架构来处理事务请求与查询分析请求的技术. 随后大量的 HTAP 数据库应运而生[2−6]. 为了同时支持 OLTP (online transaction processing) 和 OLAP (online analytical processing) 请求, 数据库面临多种技术挑战[7], 其中 OLTP 和 OLAP 的识别对 HTAP 数据库至关重要, 它有助于后续实现资源合理分配、执行引擎优化选择、查询计划精确生成以及负载动态调度. 例如, 像 TiDB[4]这样的 HTAP 数据库同时支持事务处理和实时分析, 能够根据负载类型动态路由到不同的计算引擎. 不同类型的负载对系统的资源需求和执行策略各不相同, 准确识别 OLTP 和 OLAP 负载可以确保数据库根据工作负载的特性优化性能, 避免资源冲突, 提升响应速度和资源利用率.

HTAP 数据库采用行列混合架构的核心动因在于 OLTP 与 OLAP 负载的本质差异: OLTP 工作负载以短事务、点查询为主, 需要行存引擎的随机读写优化; 而 OLAP 工作负载侧重批量扫描和聚合分析, 依赖列存引擎的向量化处理. 这种根本性差异使得存储引擎的选择直接影响性能表现. 如图 1 所示, 通过对比两个典型查询案例 query1 (TPC-H Q18, 代表 OLAP 分析型负载) 与 query2 (TPC-C 支付查询, 代表 OLTP 事务型负载), 清晰揭示了误用引擎的代价: 当 OLAP 查询被迫使用行存引擎时, 其全表扫描特性导致执行时间增长 6 倍, 内存消耗激增 1 000 倍; 反之, OLTP 点查询误用列存引擎时, 因无法利用索引而延迟增加 10 倍, 内存开销扩大 100 倍. 这种数量级的性能差异证明, 负载识别是 HTAP 数据库发挥混合架构优势的前置条件.

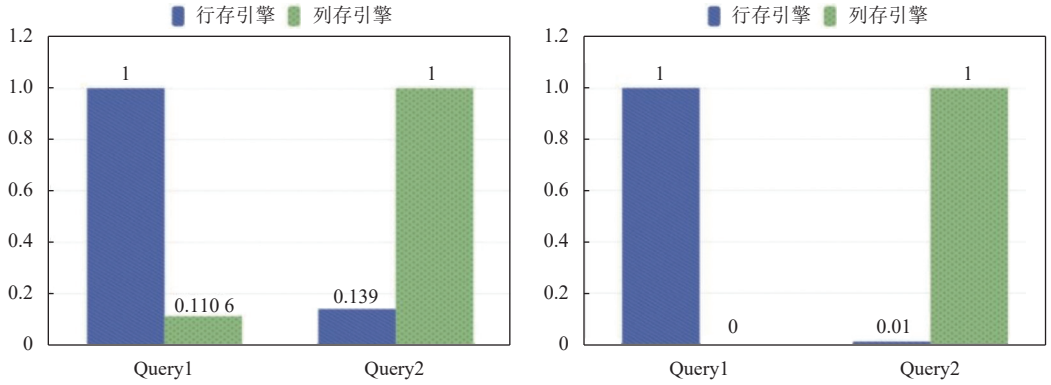


图 1 OLTP 和 OLAP 负载查询语句在不同执行引擎下的性能对比

现有的识别数据库工作负载的研究包括以下两个方面. 首先, 从不同的角度描述 OLTP 和 OLAP 负载[8,9], 例如通过分析 SQL 语句的语法和复杂度、观察执行过程中的动态属性和硬件资源的利用情况、分析查询执行计划等. 其次, 利用这些特性构建分类器, 用于识别和预测工作负载. 这些分类器有基于启发式规则的方法[4,6]或传统机器学习算法[10−12], 如推理方法和支持向量机等.

尽管上述研究在某种程度上解决了识别工作负载的问题, 但仍存在一些局限性. 首先, 现有工作负载识别方法常受限于特征片面性, 基于 SQL 规则的方法忽略执行动态性[13], 而资源监控方法 (如 CPU/内存) 受硬件环境影响显著[9], 基于执行计划的方法导致丢失静态的语义信息 (SQL 关键字和表名模式)[14]. 相比之下, 执行计划提供查询的物理执行路径 (算子类型、资源消耗、数据流依赖), SQL 语句则明确查询的语义意图 (操作类型、语法结构、业务场景), 二者结合同时覆盖了查询的动态执行特性和静态逻辑特征. 其次, 现有的编码方式不能充分提取特征的结构信息, 比如独热编码的 SQL 语句无法体现 SQL 语句的结构信息[15], 执行计划树编码没有体现算子的父子依赖关系[16,17]. 第三, 传统的分类方法需要手动特征工程并且无法自适应复杂多变的负载场景[11,12].

为了解决以上的限制和提高工作负载识别的性能, 我们提出了一种工作负载识别方法 PS-ATCNN (plan-SQL-attention-TreeCNN), 该方法同时使用 SQL 语句和执行计划作为识别 OLTP 和 OLAP 负载的特性, 对 SQL 语句构建预训练模型进行编码, 特别的, 我们将 SQL 语句的结构特征加入预训练模型, 同时, 为了保留执行计划的父子级的依赖关系设计了一种树形编码方式. 最后对二者进行特征提取和融合形成工作负载的特征向量, 输入结合树卷积神经网络和注意力机制的编码器中, PS-ATCNN 能够准确地识别输入的工作负载属于 OLTP 或 OLAP.

本文的主要贡献如下.

(1) 提出了一种融合 SQL 语句和执行计划特征的工作负载识别模型 PS-ATCNN. 该模型通过表征和编码两种特征, 并进行特征融合来处理不同结构的特征向量. 给定一条 SQL 语句, 模型可以准确地识别该 SQL 语句的负载类型.

(2) 构建并预训练 SQL 语句编码器 SQLEncoder, 同时感知 SQL 语句的语义信息和 SQL 语句的结构信息, 可以将不同长度的 SQL 语句编码为固定长度的向量.

(3) 提出了一组静态和动态结合的执行计划工作负载特性. 此外, 介绍了一种树编码方法, 该方法将执行计划的复杂结构转换为特征向量树, 向量树包含了基本的树结构信息, 同时保留了执行计划中每个节点的特征信息.

(4) 通过实验验证, 在多种 OLTP 与 OLAP 数据集上, 与规则方法、传统机器学习和深度学习模型对比, 并通过算子依赖结构、Transformer 层数和关键特征的消融实验, 全面评估了 PS-ATCNN 的性能与成本; 同时将其集成至 TiDB, 通过行存/列存的强制路由, 验证了模型的高精度、鲁棒性及实际应用价值.

本文第 1 节介绍工作负载识别相关的研究成果. 第 2 节描述问题定义. 第 3 节介绍智能识别模型的设计和实现. 第 4 节通过分析实验的结果, 验证模型的性能、鲁棒性和模型对数据库的实际性能影响. 第 5 节总结全文.

1 相关工作

如何识别 OLTP 和 OLAP 工作负载, 国内外已经展开了很多研究, 包括工作负载的表征方法和工作负载的识别方法. 然而 HTAP 数据库产品为了追求高性能和稳定性, 主要使用基于规则和代价的简单方法来识别 OLTP 和 OLAP 工作负载.

1.1 工作负载的表征

OLTP 和 OLAP 作为两种典型的工作负载, 具有不同的特性[18]. Yu 等人[19]通过研究 SQL 语句的结构和复杂性, 以及查询和事务两种工作场景中的特性, 提出通过建立评估数据库系统的基准来替代工作负载的表征. 还有研究分析数据库状态变量作为识别工作负载的特征, 如 Zewdu 等人[20]通过记录当前查询执行过程中所选数据库状态变量的值, 与历史状态变量取差值作为工作负载特征. Elnaffar 等人[8]通过分析查询过程中的性能快照, 发现 OLTP 和 OLAP 在索引使用率、排序时间、查询持有的锁个数、排序操作次数等属性差异较大. Sen 等人[9]从资源利用率的角度验证了 OLTP 和 OLAP 对资源敏感性的差异性, 发现 CPU 核数、内存容量和非易失性存储带宽是影响数据库性能的关键系统资源. 现有的研究大多关注查询在执行过程中动态属性 (如查询复杂度、执行时间、负载波动等) 和硬件资源 (如 CPU、内存、I/O 等) 变化情况, 这些方法通常受到硬件资源和物理环境的影响, 因而难以在不同的运行环境中通用.

最近的研究使用脱离物理条件的特征作为工作负载的表征. 执行计划常作为数据库优化任务的特征[21], Zhao 等人[14]提出了基于树形结构 Transformer[22]的查询计划表示模型 QueryFormer, 将数据库系统中的直方图信息集成到查询计划编码中, 同时使用注意力机制捕获查询计划中树结构的信息流. zero-shot[23]和 Stage[24]将执行计划构建为树特征, 使用图卷积网络 (graph convolutional network, GCN) 进行信息传递, 进行执行计划编码. Tang 等人[13]使用 SQL 语句作为特征, 提出了一种预训练 SQL 表示模型 PreQR, 通过注意力机制建立一个新的 SQL 编码器来编码查询结构. QueryFormer 和 PreQR 可以作为通用的编码器集成到数据库任务的机器学习中, 并将执行计划和 SQL 语句带到了工作负载描述的前沿. 我们同时考虑执行计划和 SQL 语句来表征工作负载.

1.2 工作负载的预测

在工作负载表征的基础上, 已有的研究使用机器学习分类算法来识别工作负载, Elnaffar 等人[11]将 DBMS 工

作负载的分类问题视为一个机器学习问题, 其中数据库管理系统必须学习如何识别工作负载组合的类型. 第 1 步, 收集快照, 通过给快照添加 OLTP 或 OLAP 的标签来训练分类模型; 第 2 步, 使用训练好的模型在线分类. Shaheen 等人[12]提出了一种用于工作负载类型预测的基于案例的推理模型 (case-based reasoning, CBR), 工作负载特性结合 MySQL 的状态变量和工作负载特性向量, 该模型还具有适应动态工作负载行为的能力, 具有较好的分类效果. Raza 等人[10]提出了一种具有预测性和自适应性的框架 AWPP, 可以自主对工作负载性能预测, 通过和其他机器学习分类模型对比, 证明该框架分类更准确. 但是传统的分类模型无法编码执行计划的树形结构和 SQL 语句的词结构信息.

随着人工智能的发展, 不少研究将深度学习应用到数据库工作负载预测问题中, Shaheen 等人[25]将深度学习方法用于大规模数据存储库的性能调优, 提出了一种性能预测模型, 基于优化生成对抗网络 (generative adversarial net, GAN) 的深度学习模型, 从自主管理的角度来看, 采用了 MAPE-K 模型来自主管理工作负载. Ma 等人[26]基于历史数据预测未来查询的预期到达率, 将历史查询作为模版, 使用循环神经网络 (recurrent neural network, RNN) 等模型去预测数据库将要执行查询语句的类型和数量. Zhou 等人[27]提出基于图嵌入的并发查询性能预测模型, 将查询执行计划表示为图结构, 使用图神经网络预测查询的性能. 现有深度学习的模型主要用于工作负载的性能预测, 在 OLTP 和 OLAP 工作负载识别问题中还没有应用. 并且 OLTP 和 OLAP 两种工作负载具有典型的特性, 我们构建专门的 OLTP 和 OLAP 工作负载识别模型.

1.3 HTAP 数据库中的工作负载识别

HTAP 数据库产品为了性能主要使用启发式规则方法和基于代价的方法来识别 OLTP 和 OLAP 工作负载. Oracle[2]和 SQL server[3]通过主行存支持 OLTP 负载和内存列存来支持 OLAP 负载, 执行查询时首先尝试在内存列存中执行查询, 如果有列存就在列存中执行, 如果没有列存则在行存中执行. 但是基于启发式规则的方法由于规则太简单, 识别不够准确并且无法做到自动区分工作负载. TiDB[4]使用 TiKV 行存引擎来支持 OLTP 负载, TiFlash 列存引擎来支持 OLAP 负载, 根据统计信息计算行扫描、列扫描和索引扫描这 3 种方式的执行代价, 选取最优的执行方式, 然后代理层将 SQL 语句路由到对应的 TiKV 节点或者 TiFlash 节点上执行. 相比之下, PolarDB-X[6]通过估计查询执行所需的核心资源, 如 CPU、内存、网络和 I/O, 将这些估计值与阈值进行比较, 将工作负载分类为 OLTP 或 OLAP. 所有 OLTP 请求都被路由到主读写节点, 而 OLAP 请求在 MPP 优化阶段被进一步处理. 基于代价的方法需要较高的成本代价.

2 问题定义

对于 OLAP 和 OLTP 工作负载识别问题, 可以看作二分类问题, 通过收集 OLTP 和 OLAP 类型的 SQL 语句作为原始数据集, 在原始数据的基础上提取特征并编码构成训练数据, 选择合适二分类模型进行训练, 使用训练好的分类器来识别负载属于 OLTP 或 OLAP 类型.

SQL 语句作为数据库通用的结构语言, 具有丰富的语义信息, 包括数据模式、操作类型和 SQL 语句的结构信息, 可以作为工作负载的特征. 但是, 同一 SQL 语句在不同执行引擎中会产生不同的执行计划, 只使用 SQL 语句作为识别特征无法真实体现执行过程; 执行计划可以真实反映执行过程使用的算子和动态特征. 所以, 我们选取 SQL 语句和其执行计划相结合作为 OLAP 和 OLTP 工作负载特征, 不仅保留 SQL 语句的语义信息和执行过程中的信息, 还具备执行过程中该语句的性能特性.

使用 SQL 语句和执行计划定义工作负载, 公式如下:

$$W = \{(q_1, p_1), (q_2, p_2), \ldots, (q_n, p_n)\} \tag{1}$$

其中, q_i 为查询语句 Query, p_i 为 Query 的执行计划. 具体的 $q_i = \{token_1, token_2, \ldots, token_n\}$, $token_i$ 为查询语句 Query 分词后的令牌; $p_i = \{node_1, node_2, \ldots, node_n\}$, $node_i$ 为执行计划 p_i 中的节点.

通过对工作负载 W 编码构建分类器的输入空间, 公式如下:

$$X = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \tag{2}$$

其中, x_i 为一条 SQL 语句的特征向量, y_i 为工作负载的类型, 具体公式如下:

$$x_i = Encoding_q(q_i) \times Encoding_p(p_i) \tag{3}$$

通过 SQL 语句的编码器 $Encoding_q$ 和执行计划的编码器 $Encoding_p$ 分别对 SQL 语句 q_i 和执行计划 p_i 编码, 再对编码的特征进行特征融合构成工作负载 W_i 的输入空间.

定义分类器: $h_\theta(x)$, 使得 $0 \leqslant h_\theta(x) \leqslant 1$, 代表 x 对 OLAP 或 OLTP 类型的隶属度.

根据下面的分类准则将隶属度转化为输出分类, 公式如下:

$$y_i = \begin{cases} 0, & h_\theta(x_i) < 0.5 \\ 1, & h_\theta(x_i) \geqslant 0.5 \end{cases} \tag{4}$$

输出空间为: $Y = \{0,1\}$, 0 代表 OLAP 类型, 1 代表 OLTP 类型.

3 PS-ATCNN 智能识别方法

本节介绍 PS-ACTNN 的主要组件, 包括执行计划编码和 SQL 语句编码. 此外, 详细介绍 PS-ACTNN 模型的构建.

3.1 执行计划编码

我们使用相同数量的数据进行 TPC-C 和 TPC-H 测试, 分别代表 OLTP 和 OLAP 工作负载. 我们分析这两类执行计划, 统计算子的使用频率和数值差异. 首先, 统计各算子的出现次数, 然后计算执行过程中的动态特性 (如 Total cost 和 Plan rows) 的平均值. 最后, 参考 OLTP 数据对 OLAP 数据进行归一化. 分析结果如图 2 所示, Gather 和 Index only scan 算子的使用频率较低,因而不适合作为特性. 我们从图 2 所示的算子中选择了 9 个差异显著的算子作为静态特性, 包括 ModifyTable、Aggregate、Gather merge、Sort、Index scan、Seq scan、Hash join、Hash 和 Nested loop. 此外, 基于 SQL 执行过程中的动态特性和资源使用情况, 我们选取了 7 个动态特性: Total cost、Plan rows、Plan width、Shared hit blocks、Shared read blocks、Temp read blocks 和 Temp write blocks, 这些特征共同构成执行计划节点的特征表示.

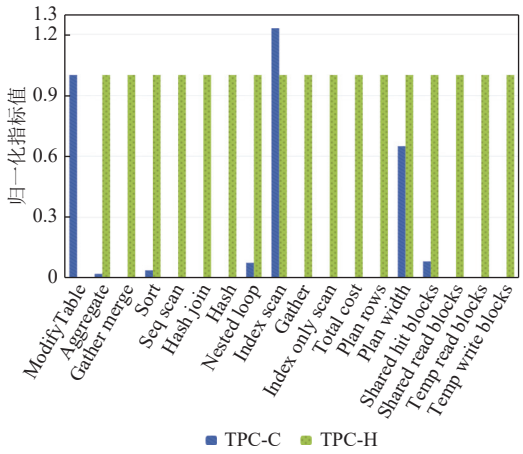


图 2 TPC-C 和 TPC-H 执行计划的特征对比

为了更好地提取执行计划中的特征, 在特征编码时分为两部分: 第一, 执行计划树结构编码; 第二, 执行计划树中的节点编码. 计划树结构编码, 如图 3(a) 我们使用 TPC-C 和 TPC-H 测试中出现的 11 种算子组成结构向量, 同时通过在子节点中添加父节点的算子信息来表征树形结构. 具体使用独热编码的方式为在子节点添加父节点的算子类型, 子节点的结构向量中父节点算子类型编码为 1, 其他算子类型编码为 0, 将不同的层级关联起来, 构成一棵树向量, 由于根节点没有父节点, 所有节点类型都编码为 0.

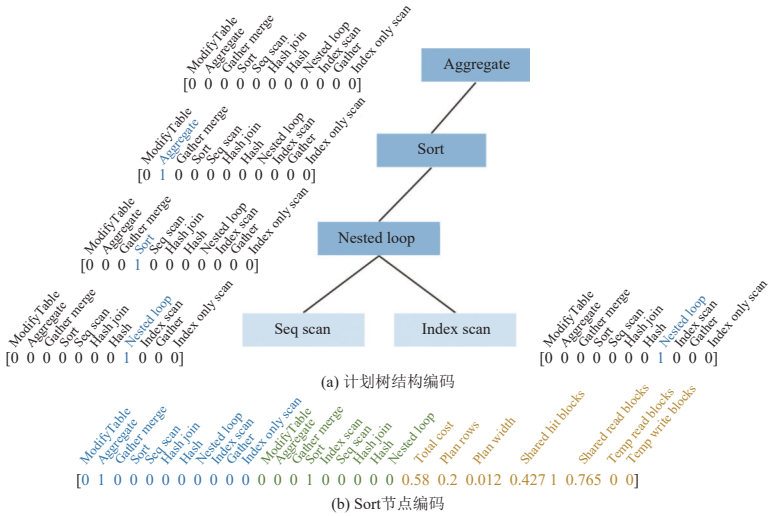


图 3 特征树编码和特征树节点编码

节点编码由 9 个静态算子和 7 个动态特性组成的 16 维特征向量构成. 静态算子使用独热编码, 若某节点类型属于 9 类静态算子, 则该类编码为 1, 其他 8 类为 0; 7 个动态特性为数值类型, 直接作为特征值. 由于 OLAP 场景下 Total cost 的数值通常为十万级别, 而独热编码为 0 或 1, 动态特性数值与独热编码差异较大, 可能影响分类器识别性能[28], 因此我们对动态特性数值进行最小-最大归一化, 将其调整至 [0, 1] 之间. 最后, 将计划树中的编码合并到每个节点, 构成完整的特征向量树. 如图 3(b) 所示, Sort 节点的完整特征向量由 3 部分组成: 父节点类型的独热编码、本节点类型的独热编码和动态特性归一化后的数值编码.

3.2 SQL 语句编码

在自然语言处理 (natural language processing, NLP) 领域中, BERT (bidirectional encoder representations from Transformers)[29]是语义理解任务中使用最广泛的预训练模型之一, BERT 是一种基于 Transformer 模型的预训练句子编码器. 它通过使用多个独立的注意力头, 分别计算注意力权重, 并将它们的结果进行拼接或加权求和, 从而获得更丰富的表示. BERT 通过在大规模的语料库上进行无监督的预训练, 学习丰富的句子表示, 可以用于下游自然语言处理任务.

我们对 BERT 模型进行剪裁和修改, 原始的 BERT 使用分块编码来支持多个句子的输入, 每个句子编码为不同的分块, 在对 SQL 语句编码时, 每次的输入只有一条 SQL 语句, 不再需要分块编码, 我们将分块编码替换为 SQL 语句的结构编码来提取 SQL 语句的结构信息. 对于 SQL 查询中的每个令牌 t_i, 我们创建了一个复合嵌入 $e(t_i)$, 它由 $t(t_i)$、$s(t_i)$、$p(t_i)$ 三者共同组成, $t(t_i)$ 代表 SQL 语句每个词的词编码, $s(t_i)$ 代表词在 SQL 语句中的结构编码, $p(t_i)$ 代表词的位置编码. 我们在图 4 中可视化了一个示例查询的嵌入, SQL 的第 1 个令牌总是一个特殊的分类令牌 ([CLS]). 在分类任务中, 令牌的隐藏状态 [CLS] 可以作为整个令牌序列的聚合表示. SQL 的结束标记也是一个特殊的标记 ([END]), 通过串联 3 个嵌入, 为每个令牌创建复合嵌入, 然后将其馈送到编码模型, 以生成 SQL 语句固定长度的向量表示.

在 SQL 语句中同一个表名或列名可能会同时出现在 SELECT 结构、WHERE 结构、ORDER BY 结构和 GROUP BY 结构中, 虽然 BERT 的位置编码可以捕捉输入句子中单词之间的相对位置信息, 但是 SQL 语句结构的位置不固定, 位置编码无法准确地描述不同位置的相同表名和列名, 我们对 SQL 语句进行结构拆分, 如表 1, 将 SQL 语句分为 14 种结构类型, 将关键字、符号和其他词区分开, 同时, 可以对不同结构下的表名和列名进行单独编码, 对每种结构进行分类编码作为 $s(t_i)$ 结构编码, 图 4 中结构编码的每一种类型代表一种结构类型, 如 s_B 代表所有关键字、s_C 代表 SELECT 结构中的表名和列名.

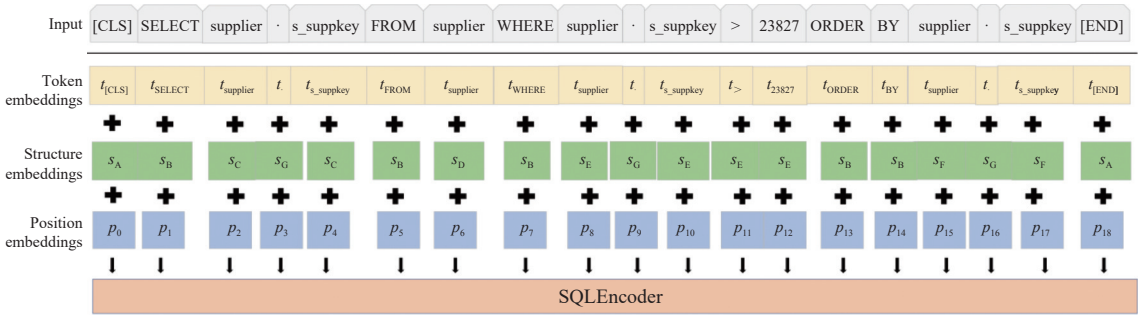


图 4 SQLEncoder 模型输入

表 1 SQL 语句的结构类型

SQL结构类型	包含值	编码
SQL开始结束标识符	[CLS]、[END]	0000
关键词	SELECT、UPDATE、INSERT、FROM...	0100
SELECT结构	表名、列名、列别名	0201、0202、0203
FROM结构	表名、表别名	0301、0302
WHERE结构	表名、列名、关系、值	0401、0402、0403、0404
HAVING结构	表名、列名、关系、值	0501、0502、0503、0504
ORDER BY结构	表名、列名	0601、0602
GROUP BY结构	表名、列名	0701、0702
LIMIT结构	表名、列名	0801、0802
INSERT INTO结构	表名、列名、值	0901、0902、0903
UPDATE结构	表名、列名、值	1001、1002、1003
DELETE结构	表名、列名、值	1101、1102、1103
ON结构	表名、列名、关系、值	1201、1202、1203、1204
符号结构	“.” “, ” “(” “)” “*”	1300

由于 SQL 语句中的词汇和自然语言的词汇有很大区别, SQL 语句中包含大量的数据库专有名词, 比如表名“lineitem”、列名“s_suppkey”和组合关系符“>=”等, BERT 模型使用的 WordPiece 词嵌入把原始单词拆分为较小的子词和字符, 比如字段“s_suppkey”会拆分为“s”“_”“su”“##pp”和“##key”这 5 个子词, 为了保留数据表名和列名等专有名词, 我们根据 SQL 语句的结构和 SQL 语句的词汇特点自定义分词算法, 将 SQL 语句中的词汇、运算符号和标点符号拆分, 分词后使用 WordPiece 将每个单词编码为输入令牌 $t(t_i)$. 同时, 我们使用数据库的关键词和数据库元数据组成训练数据, 训练数据的词汇量为 50 000 词, 使用预测被遮挡单词的任务, 在预训练模型权重的基础上对模型进行微调, 随机掩蔽 15% 比例的输入标记, 然后预测这些被掩蔽的标记, 将训练好的模型称为 SQLEncoder.

3.3 PS-ATCNN 模型设计

常用的全连接神经网络 (fully connected neural network, FCNN)、卷积神经网络 (convolutional neural network, CNN) 需要固定大小的输入张量, 为了保留执行计划树中的结构特性, 我们使用树卷积神经网络 (tree convolutional neural network, TreeCNN)[30]构建分类模型, 实现 OLTP 和 OLAP 负载的识别. 树卷积中引入一种可组合可微的用于有监督程序分析的可微神经网络算子, 与图像的卷积转换类似, 树的卷积会在查询树的每个部分上局部滑动一组共享的过滤器, 生成一个已转换成相同大小的执行计划树. 这些过滤器可以查找散列连接对等模式, 或者在一个非常小的关系上进行索引扫描. 树的卷积运算符是堆叠的, 每一个树的卷积层都比上一个树有更大的接受域. 因此, 第 1 个树的卷积层将学习简单的特征, 而最后一个树的卷积层将学习复杂的特征.

获取整个执行计划的表示具有挑战性, 因为现成的方法很难捕获所有重要的信息. 现有的 TreeCNN 方法[31,32]

使用平均池化或动态池化方法从节点中聚合特征. 这些池化操作以暴力破解的方式对信息进行下采样, 会造成信息的丢失. 例如, 平均池化将所有节点同等对待, 而不考虑它们的数据和执行成本. 然而, 每个节点对最终查询特征的贡献并不相等, 不同的层级在负载识别中的重要程度也不同, 比如在 INSERT、UPDATE 和 DELETE 语句的执行计划中, 根节点为 ModifyTable 算子, 通过根节点一般可将该 SQL 语句识别为 OLTP 类型, 下级节点在识别过程中所占权重较低. 为了解决上述问题, 我们在模型中引入了空间注意力机制[33], 空间注意力机制是一种用于卷积神经网络的注意力模块, 可以增强网络对于重要特征的关注, 并抑制无关特征的影响. 具体而言, 空间注意力机制通过实现注意力加权: 用于计算每个空间位置的重要性权重, 以便网络更关注重要位置的特征, 而忽略不重要位置的特征. 通过这种方式, 空间注意力机制能够提高网络对于重要特征的提取能力, 所以我们在树卷积神经网络中嵌入空间注意力机制.

PS-ATCNN 模型结构如图 5, 模型分为 3 部分, 第 1 部分为执行计划的编码子网络, 第 2 部分为 SQL 语句的编码子网络, 第 3 部分为两个编码子网络融合之后进行分类任务的子网络. 第 1 部分由 SQL 语句编码的 SQLEncoder 模型和对编码进行特征提取的网络层组成, SQLEncoder 分为嵌入层、编码层和输出层, 嵌入层的输出来源于 3 个嵌入向量 $e(t_i)$ 的相加, 编码层由 8 个 Transformer 的编码层组成, 每一层包含一个多头自注意力层和一个前馈全连接层, 词向量长度是 768, 注意力头的个数为 12. 每个注意力头通过计算目标词与句子中的所有词汇的相关度, 对目标词重新编码. 3 个权重矩阵对输入的序列向量做线性变换, 分别生成 query、key 和 value 这 3 个新的序列向量, 用每个词的 query 向量分别和序列中的所有词的 key 向量做乘积, 得到词与词之间的相关度, 然后这个相关度再通过 *Softmax* 进行归一化, 归一化后的权重与值加权求和, 得到每个词新的编码, 如公式 (5):

$$Attention(Q,K,V)=Softmax\left(\frac{QK^{\mathrm{T}}}{\sqrt{d_k}}\right)V \tag{5}$$

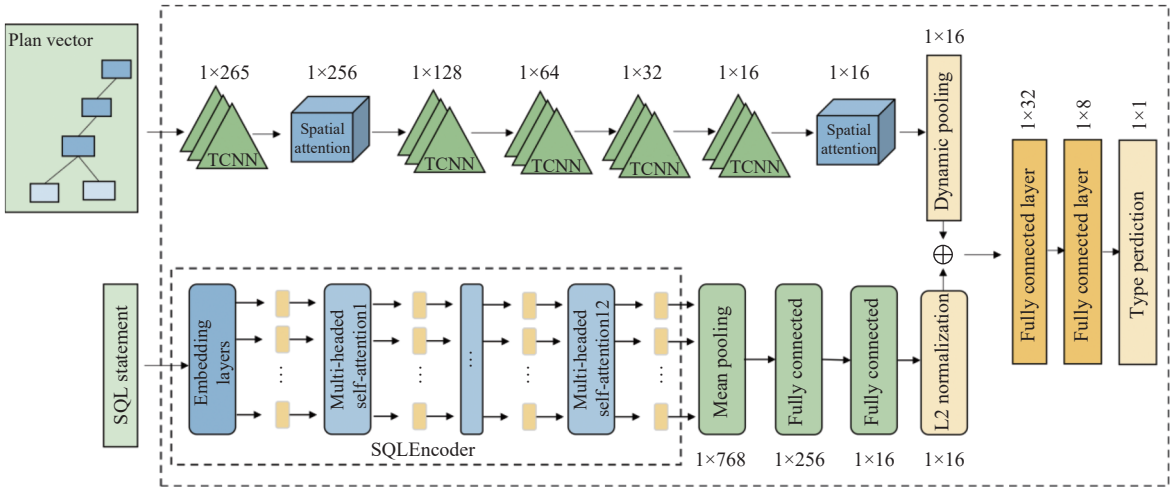


图 5 PS-ATCNN 模型结构

经过编码层将每个单词编码为 768 长度的向量. SQL 语句经过 SQLEncoder 编码之后输入到 PS-ATCNN 模型中, 语句中的多个词通过全局平均池化形成一个向量, 再经过两个全连接层降低特征的维度, 最后通过 L2 归一化层形成归一化的固定长度一维向量 e_s.

第 2 部分是对执行计划树进行特征提取的子模型, 包括卷积层和全连接层, 卷积层由 5 层树卷积和 2 层空间注意力机制构成, 编码好的特征树向量作为第 1 层树卷积的输入, 树卷积的卷积核为 3, 每一层树卷积进行一次归一化, 再经过一层空间注意力机制进行空间特征提取, 依次输入 4 层树卷积进行特征提取, 卷积层最后使用注意力机制进一步特征提取, 在最后一层卷积之后, 使用动态池化[30]将树的结构转化为固定长度的一维向量 e_p.

第 3 部分首先将 SQL 语句的固定长度特征向量 e_s 和执行计划动态池化输出的固定长度特征向量 e_p 进行拼

接融合形成最终的特征向量 y, 如公式 (6):

$$y = Concat(e_s, e_p) \tag{6}$$

新特征向量同时体现执行计划和 SQL 语句的特征信息. 使用两个全连接层来将合并的向量映射到一个一致性预测, 最后使用 Sigmoid 对结果映射为类别比例.

4 实　验

4.1 实验环境

我们所有的工作在一台服务器上完成, 服务器的配置参数如下.

- 系统: CentOS Linux release 7.9.2009 (Core) 64 位.
- CPU: 10 核 Intel(R) Xeon(R) Silver 4210R CPU @ 2.40 GHz.
- GPU: NVIDIA GeForce RTX 3080.
- 内存: 128 GB.
- 磁盘: 16 TB.

使用 PostgreSQL 14.0 数据来生成查询执行计划, 使用 TiDB v8.5.1 数据库集成 PS-ATCNN 模型. PyTorch 实现神经网络模型, 并使用 Python 3.7 解释器来运行模型.

4.2 数据集和基准测试及评价标准

4.2.1 数据集

数据集包括两类: 训练数据集和验证数据集. 训练数据集由两类 SQL 语句组成: OLTP 工作负载和 OLAP 工作负载. 记录 TPC-C 测试过程中运行的 SQL 语句, 并使用 1 000 个随机选择的 SQL 语句作为 OLTP 工作负载. 由于 TPC-H 决策支持基准测试只有 22 个查询模式, 我们使用 LearnedSQLGen[34]方法来生成 1 000 个类 TPC-H 的查询语句. 验证数据集包括两个 OLAP 工作负载数据集 (TPC-H 和 JOB[35]) 以及 3 个 OLTP 工作负载数据集 (SmallBank[36]、TATP[37]和 Epinions[38]), 它们使用 OLTP-Bench[39]生成, 并选择 SQL 语句的随机样本, 具体如表 2 所示.

表 2　训练和验证数据集

数据集作用	数据集	表个数	查询个数
训练	OLTP workload	9	1 000
训练	OLAP workload	8	1 000
验证	TPC-H	8	22
验证	JOB	21	113
验证	SmallBank	3	1 529
验证	TATP	4	20 017
验证	Epinions	5	21 244

4.2.2 基准测试

使用基于启发式规则的方法 (heuristic rule, RULE)、3 个常用的机器学习分类算法人工神经网络 (artificial neural network, ANN)、支持向量机 (support vector machine, SVM)、朴素贝叶斯 (naive Bayes, NB)、3 个最新的研究 (AWPP[10]、QueryFormer[14]和 Stage[24]) 和两个开源大语言模型 (DeepSeek-V3、Qwen2.5-32B-Instruct[40]) 与 PS-ATCNN 模型进行性能对比.

RULE: 我们使用常用的两种工作负载分类启发式规则设计二级判定逻辑实现工作负载分类, 首先基于 SQL 语法特征进行快速判断, 若查询包含 GROUP BY、HAVING 等聚合语法或窗口函数则直接标记为 OLAP 类型, 若包含 INSERT/UPDATE 等写操作语法则判定为 OLTP 类型; 其次结合运行时资源消耗阈值进行兜底判断[6], 当 CPU 使用率≥20% 或内存占用≥1 GB 或执行时间≥300 ms 时判定为 OLAP, 反之判定为 OLTP.

ANN: 由大量的节点 (神经元) 组成, 节点之间通过连接 (突触) 相互作用. ANN 的训练过程通常采用反向传播

算法, 即通过不断地调整连接权重来使 ANN 输出结果与实际结果之间的误差最小化. 模型设计和训练过程尽量和 PS-ATCNN 模型保持一致, 我们使用 5 个隐藏层来构建 ANN 模型, ReLU 作为激活函数, 训练 2000 次.

SVM: 一种常见的判别方法, 在机器学习领域, 是一个有监督的学习模型, 通常用来进行模式识别、分类以及回归分析. 我们使用径向基函数作为核, 适用于非线性多维特征的分类场景.

NB: 一种基于贝叶斯定理的分类算法, 它假设所有的特征都是独立的, 并且每个特征的重要性相等. 同时, 朴素贝叶斯模型所需估计的参数很少, 对缺失数据不太敏感, 算法也比较简单. 由于本问题为多特征的二分类问题, 我们选择伯努利贝叶斯算法.

AWPP: 我们实现了 AWPP 框架中的特征提取和负载预测部分, 在训练数据集中获取 7 个特征的工作负载特征向量: 选择谓词个数、排序列个数、相等选择谓词个数、连接谓词个数、不相等选择谓词个数、嵌套子查询个数和聚合列个数作为负载特征, AWPP 的负载预测分类器使用 CBR 模型.

QueryFormer: 一个基于树结构 Transformer 的执行计划表示模型, 可用于各种数据库机器学习任务, 我们使用相同的训练数据集重新训练 QueryFormer, 将训练好的 QueryFormer 模型来代替 PS-ATCNN 的编码部分.

StageGlobal: 我们实现了 Stage 模型中的全局预测模型 StageGlobal, 该模型将执行计划构建为树形特征, 使用 GCN 进行消息传递, 对执行计划进行编码. 使用相同的训练数据集重新训练 StageGlobal, 将训练好的 StageGlobal 模型来代替 PS-ATCNN 的编码部分.

DeepSeek-V3: 一款拥有 6710 亿参数的混合专家 (mixture of experts, MoE) 语言模型, 每个令牌激活 370 亿参数, 旨在实现高效推理与低成本训练. Qwen2.5-32B-Instruct (后文简称为 Qwen2.5-32B) 是阿里云开发的开源代码生成模型, 基于 Qwen2.5 架构, 参数规模 320 亿, 具有突出的代码生成能力、长上下文支持和多模态应用等特点. DeepSeek-V3 和 Qwen2.5-32B 通过调用模型 API, 输入 SQL 语句及其执行计划, 经过模型推理后, 返回该 SQL 语句属于 OLTP 或 OLAP 类型.

PS-ATCNN: 使用 ReLU 激活函数, Adam 作为优化器, 学习率为 0.01, 损失函数为二元交叉熵 (binary cross-entropy, BCE), 训练过程使用的批大小为 8, 每个批次计算损失并通过反向传播更新网络参数, 通过两个条件来终止训练过程: (1) 训练达到 20 个批次; (2) 达到收敛 (通过训练损失的减少来衡量, 超过 3 个批次的比例小于 0.01%).

4.2.3 模型评价标准

通过测量准确性和有效性标准来评估性能预测和适应性. 实际的和预测的类可以是正的或负的. 当实际的类是正类, 预测的类是正类, 这就代表真正的 *tp*, 即, 有正确的结果; 当实际的类是正类, 预测的类是负类, 这代表假阴性 *fn*, 即缺失的结果. 类似地, 当实际的类是负类时, 判断类和预测类是正类, 这代表假阳性 *fp*, 即意外结果; 当实际类是负类, 预测类是负类, 这代表真负 *tn*, 即没有正确的结果. 其中 *tp* 为真阳性, *fp* 为假阳性, *fn* 为假阴性, *tn* 为真阴性.

通常机器学习模型评价指标包括精度、召回率、*F-measure* 测量值和准确率, 通过 *tp*、*fp*、*fn*、*tn* 计算得到, 其计算公式如公式 (7)–公式 (10) 所示.

精度公式如下:

$$Precision = \frac{tp}{tp + fp} \tag{7}$$

召回率公式如下:

$$Recall = \frac{tp}{tp + fn} \tag{8}$$

F-measure 测量值公式如下:

$$F\text{-}measure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{9}$$

准确率公式如下:

$$Accuracy = \frac{tp + tn}{tp + tn + fp + fn} \tag{10}$$

4.3 实验结果

4.3.1 模型性能对比

实验使用 1 000 条 OLTP 负载数据和 1 000 条 OLAP 负载数据组成数据集, 80% 的数据作为训练集, 20% 的数据作为测试集. 对比 PS-ATCNN 模型和 5 种基准模型的性能情况. 为了适应 ANN、SVM 和 NB 这 3 种模型, 将树形结构的执行计划压缩为一维向量, 通过分别统计执行计划树中 9 个静态指标的出现次数, 并计算 7 个动态指标的累加和, 以此构建一个 16 维的特征向量作为执行计划树的特征向量, 完成 3 种模型的训练.

表 3 和图 6 为 10 种模型实验结果性能对比, PS-ATCNN 的精度、准确率和 *F-measure* 测量值均为最高, ANN、SVM 和 NB 模型使用的特征经过压缩之后保留了各个算子的数量信息, 执行计划的结构信息丢失, 导致模型的性能不如 PS-ATCNN. QueryFormer 只使用执行计划作为工作负载的特征准确率低于 PS-ATCNN 同时使用执行计划和 SQL 语句的特征. AWPP 的准确率较低, 和其他模型相比, AWPP 只提取了 SQL 语句的 7 个统计信息无法体现执行过程中的动态特征, 不能较好地表征工作负载. 实验说明执行计划的结构和 SQL 语句结合可以作为对识别 OPTP 和 OLAP 工作负载问题的特征信息; 其次 QueryFormer 和 ANN 模型的性能优于 SVM、AWPP 和 NB, 神经网络比传统机器学习更适合于本研究. 同时, ANN、SVM 和 NB 分类模型准确率虽然没有 PS-ATCNN 高, 但是高于之前的研究[10,12], 说明我们选取的执行计划特征能够较好地区分 OLTP 和 OLAP 工作负载. StageGlobal、DeepSeek-V3 和 Qwen2.5-32B 这 3 个模型更倾向于将 SQL 识别为 OLTP 负载类型, StageGlobal 通过邻居节点的汇聚来更新节点信息, 执行计划树结构的父子节点关系和叶子节点的信息传播不足以准确捕捉树的层次结构和语义. Qwen2.5-32B 主要通过 SQL 语句的复杂性和执行计划的代价估计来区分工作负载类型, 缺少执行计划结构、操作类型等信息. DeepSeek-V3 的参数数量远大于 Qwen2.5-32B, 使其在推理工作负载的类别时关注更多细节信息, 例如, 执行计划的操作类型以及执行路径等信息, DeepSeek-V3 分类效果较好. 而启发式规则方法虽能达到 0.97 的 *Precision*, 但其 *Recall* 仅为 0.64, 表明大量 OLAP 查询因资源阈值或语法未命中而被漏判为事务型. 相比之下, PS-ATCNN 在同样数据集上 *Recall* 和 *Accuracy* 均接近 0.99, 有效避免了规则法的误判和漏判问题.

表 3 算法性能对比表

模型	*Precision*	*Recall*	*Accuracy*	*F-measure*
PS-ATCNN	**0.978**	0.988	**0.985**	**0.983**
RULE	0.97	0.64	0.81	0.77
ANN	0.885	0.982 5	0.93	0.931 2
SVM	0.88	0.95	0.917 5	0.913 6
NB	0.825	0.92	0.855	0.869 9
AWPP	0.78	0.985	0.855	0.870 5
QueryFormer	0.947 5	0.977 5	0.962 5	0.962 3
StageGlobal	0.68	0.992 5	0.757 5	0.807 9
DeepSeek-V3	0.912 5	**0.995**	0.952 5	0.952 2
Qwen2.5-32B	0.79	0.992 5	0.862 5	0.880 2

表 4 通过 3 个典型查询实例直观展示 PS-ATCNN 模型的特征编码机制与分类决策过程, 其中特征编码的前 16 维为执行计划树的特征表示, 后 16 维为 SQL 语句的特征表示. 第 1 条 SQL 语句模型过度关注执行计划通道中的聚合算子特征 (第 3 维 2.590 4、第 4 维 2.602 6 等高激活值), 这些来自子查询 SELECT AVG 的 OLAP 特征淹没了外层简单投影的 OLTP 特性造成误识别; 第 2 条 SQL 语句通过执行计划通道的索引扫描特征 (第 6 维 1.938 6、第 12 维 3.491 9) 与 SQL 通道的主键查询模式 (WHERE i_id=14048) 形成强协同效应准确识别为 OLTP 类型; 第 3 条 SQL 语句通过执行计划通道捕获到聚合算子 (第 3 维 2.827 1) 与全表扫描 (第 5 维 1.842 5) 的组合模式, SQL 通道检测到子查询中的聚合函数准确识别为 OLAP 类型.

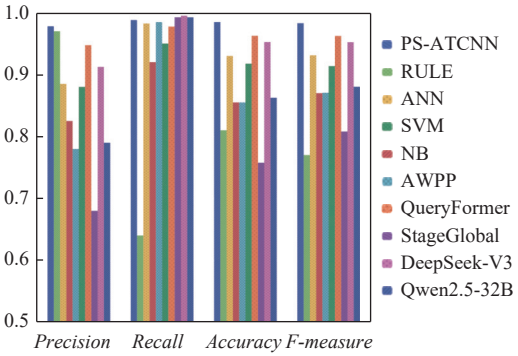


图 6 算法性能对比图

表 4 具体识别案例分析

SQL语句	特征编码	所属类型	识别结果
SELECT supplier.s_acctbal FROM supplier WHERE supplier.s_nationkey <= (SELECT AVG(supplier.s_nationkey) FROM supplier WHERE supplier.s_nationkey != 1 AND supplier.s_suppkey = 24976)	[−0.0036, −0.0042, 2.5904, 2.6026, 2.1867, −0.0042, −0.0041, −0.0042, 2.1955, 1.7685, −0.0042, −0.0042, −0.0042, −0.0041, −0.0042, −0.0038, −0.4745, −0.4646, 0.3691, −0.0794, −0.0127, 0.1430, −0.2191, 0.1322, 0.3971, −0.1362, −0.0661, 0.0892, −0.3551, −0.0699, −0.0859, −0.0580]	OLTP	0.1
SELECT i_price, i_name, i_data FROM item WHERE i_id = 14048	[−0.0040, −0.0040, −0.0040, 0.7116, −0.0040, 1.9386, −0.0040, 0.9080, −0.0040, −0.0035, −0.0040, 3.4919, 3.0238, −0.0033, −0.0040, 0.4860, −0.5256, −0.4638, 0.3015, −0.1887, −0.0281, 0.1043, −0.1734, 0.1677, 0.3582, −0.1358, −0.1347, 0.1188, −0.3313, −0.1504, −0.0278, 0.0100]	OLTP	0.997
SELECT supplier.s_suppkey FROM supplier WHERE supplier.s_suppkey = (SELECT AVG(supplier.s_suppkey) FROM supplier WHERE supplier.s_acctbal > 9683)	[0.0969, 0.4736, 1.6036, 2.8271, 1.8425, −0.0025, −0.0023, 0.2197, 2.6132, 1.7728, 0.0161, −0.0007, −0.0051, −0.0028, −0.0051, −0.0041, −0.5490, −0.3549, 0.3716, −0.2656, −0.0121, 0.1457, −0.1304, 0.1753, 0.2558, −0.2381, −0.0980, 0.2050, −0.2401, −0.2446, −0.0054, −0.0613]	OLAP	0.0161

4.3.2 鲁棒性验证

在其他数据集验证训练好的 PS-ATCNN 模型的鲁棒性, 分别在 OLAP 的验证集: TPC-H 和 JOB; OLTP 的验证集: TATP、SmallBank 和 Epinions, 这 5 种真实数据集做测试, 通过模型预测每种数据集中 OLAP 负载所占比例.

后文图 7 为模型在 TPC-H、JOB、TATP、SmallBank 和 Epinions 这 5 种数据集的验证结果, 识别准确率均达到 86% 及以上, JOB 和 SmallBank 两种测试数据可以完全识别, TPC-H、TATP 和 Epinions 这 3 种测试数据中有小部分数据识别错误. Epinions 数据中有大量特殊符号组成的值, 会给 SQL 语句的分词带入噪声, 影响 SQL 语句的编码效果. 通过验证 PS-ATCNN 模型基本能够适应不同类型的工作负载, 鲁棒性较高.

4.3.3 模型组件的影响

通过对比 PS-ATCNN 模型和其他 4 个对比模型的训练损失收敛速度、训练时间和准确率来验证 PS-ATCNN 模型中各个组件对模型整体性能的影响. 第一, 对比无注意力机制模块的 PS-TCNN 模型来验证注意力机制模块对模型性能的影响; 第二, 对比无 SQL 编码的 P-ATCNN 模型来验证 SQL 编码对模型性能的影响; 第三, 对比使用 Word2Vec[41]编码 SQL 语句的 PS-WATCNN 模型来验证 SQLBERT 编码对模型性能的影响; 第四, 去除执行计划树中的父节点类型编码和模型中的树卷积层, 使用平均池化和两层全连接层进行执行计划特征编码的 PS-MLP 模型来验证算子依赖关系对模型性能的影响.

图 8(a) 为 5 个模型的模型训练损失收敛速度对比, PS-ATCNN 模型和 PS-TCNN 模型的损失收敛更快, 20 批

次达到终止训练的条件, 而 P-ATCNN 模型需要训练 50 批次, 说明 SQL 编码可以更好地提取特征加快模型收敛, 而 PS-WATCNN 模型训练损失收敛速度较低, Word2Vec 编码对 SQL 语句特征提取效率较低, PS-MLP 模型缺少执行计划的树结构信息导致训练损失增大. 图 8(b) 为 5 个模型的训练时间对比结果, PS-ATCNN 模型和 PS-TCNN 模型的训练时间更长, SQL 编码对 SQL 语句分词和 SQLBERT 模型编码的过程需要额外的时间, PS-WATCNN 模型的训练时间低于 PS-ATCNN 模型, 使用结构简单的 Word2Vec 编码 SQL 语句比 SQLBERT 模型编码所需时间较短. PS-MLP 模型通过简化模型结构将训练时间缩短至 PS-ATCNN 的 18.7%. 同时, PS-ATCNN 模型和 PS-TCNN 模型的损失收敛速度和训练时间基本一致, 说明模型中添加的注意力机制模块对模型训练过程影响较小.

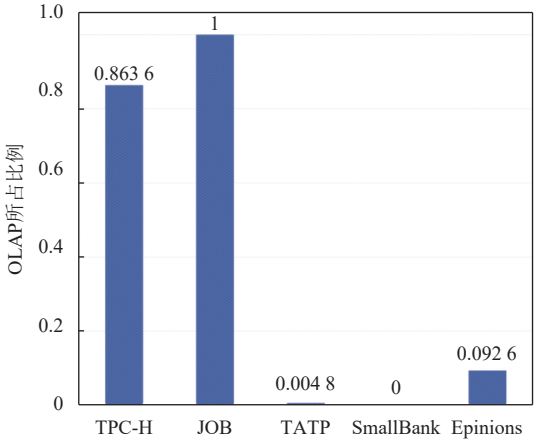
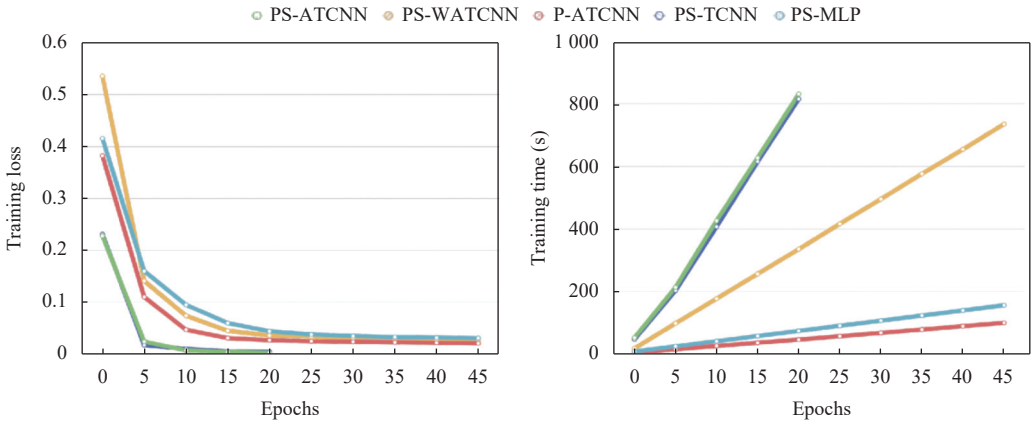


图 7 PS-ATCNN 模型鲁棒性验证



(a) 模型训练损失收敛速度对比 (b) 模型训练时间对比

图 8 4 种模型训练过程对比

表 5 对比 5 个模型的预测准确率和在 5 种真实数据集的 OLTP 负载的比例, 4 个模型的预测准确率都达到 94% 及以上, 但是 PS-TCNN 模型和 PS-WATCNN 模型在 Epinions 数据集的准确率较低, 说明 Epinions 数据中的特殊符号对 SQL 编码的影响较大, 且简单的 Word2Vec 编码无法有效地编码复杂的 SQL 语句. PS-TCNN 模型和 P-ATCNN 模型在 JOB 数据集的准确率都较低说明只使用执行计划作为区分工作负载的特征不够全面, 泛化性较差. 同时结合执行计划和 SQL 语句编码, 通过注意力机制进行特征提取可以很好地完成工作负载的区分. 最后, 保留执行计划树结构编码的 PS-ATCNN 模型在复杂负载场景 (JOB/TPC-H) 中相比扁平化处理的 PS-MLP 实现 12.4%–50.1% 的准确率提升, 证明算子依赖关系是 OLAP 负载识别的关键特征.

表 5　5 种模型准确率和鲁棒性对比

Model	*Precision*	JOB	TPC-H	TATP	SmallBank	Epinions
PS-ATCNN	0.978	0	0.1363	0.9952	1	0.9074
PS-TCNN	0.965	0.8318	0.2727	0.8999	1	0.1981
P-ATCNN	0.9367	0.8495	0.409	0.9977	1	0.8134
PS-WATCNN	0.8954	0	0.1363	0.9938	1	0.5951
PS-MLP	0.9428	0.7079	0.3636	0.9921	1	0.8724

4.3.4　SQL 语句结构编码的影响

通过在 SQL 编码中使用结构编码和不使用结构编码做对比, 在 TPC-H 数据集上验证 SQL 语句的结构编码对模型性能的影响.

表 6 中使用带有结构编码的模型和无结构编码的模型对 TPC-H 进行测试, 结果为 22 条 SQL 语句预测属于 OLTP 的隶属度, 根据本文定义, 预测结果大于 0.5 归类为 OLTP 类型、预测结果小于 0.5 归类为 OLAP 类型. 带有结构编码的模型在 TPC-H 数据集的准确率为 86%, 无结构编码的模型的准确率为 59%, 同时结构编码提高了每条 Query 的准确率, 说明原始的 BERT 模型并不适用于 SQL 语句的编码, 加入结构编码能够更好地提取 SQL 语句的特征. 其中 Q15 创建了新的视图, 在训练数据集中并未使用视图, SQL 语句编码对新词的编码不准确造成识别错误, 说明 SQL 语句编码模块对新词的敏感度较高. Q3 和 Q18 可能是因为执行计划和 SQL 语句两种特征融合之后特征变稀疏, 模型无法获取重要的特征信息造成识别错误.

表 6　TPC-H 的 22 条 Query 预测结果

TPC-H	有结构编码	无结构编码	TPC-H	有结构编码	无结构编码
Q1	0.3926	0.4694	Q12	0.4023	0.4062
Q2	0.3843	0.4847	Q13	0.3691	0.5884
Q3	0.5076	0.5043	Q14	0.3749	0.4971
Q4	0.4212	0.4567	Q15	0.6451	0.6397
Q5	0.4219	0.5132	Q16	0.4518	0.4696
Q6	0.3845	0.5393	Q17	0.2363	0.3589
Q7	0.2624	0.4758	Q18	0.5339	0.6225
Q8	0.2741	0.4631	Q19	0.2993	0.3535
Q9	0.3252	0.5431	Q20	0.3275	0.5312
Q10	0.4784	0.4926	Q21	0.2528	0.5867
Q11	0.2384	0.4719	Q22	0.2357	0.4052

4.3.5　SQLEncoder 中 Transformer 层数的影响

我们使用掩码预测任务重训练不同层数 (4、6、8、10、12) Transformer 的 SQL 语句编码器 SQLEncoder, 验证 Transformer 层级对 SQLEncoder 模型重训练成本以及模型的识别准确率和预测延迟的影响, 训练轮次为 10.

通过表 7, 我们发现模型性能与计算成本之间存在显著的非线性关系, 当层数从 4 层增加到 8 层时, 准确率从 92.8% 显著提升至 97.8%, 训练时间仅增加 0.71 h, 每 1% 准确率提升的成本为 0.14 h; 然而当层数从 8 层继续增加到 12 层时, 准确率仅微增 0.8%, 训练时间却增加 0.64 h, 每 1% 准确率提升的成本急剧上升到 0.8 h, 同时预测延迟增长 43%. 这表明 8 层模型在 97.8% 的准确率水平上达到了最佳的成本-性能平衡点, 既满足生产环境对高精度的需求, 又保持了高效的训练速度和实时推理能力, 适合 HTAP 数据库等对延迟敏感的在线路由场景. 而且 SQL-Encoder 作为预训练模型, 其参数在 PS-ATCNN 主模型训练过程中保持冻结, 仅作为静态特征提取器使用, 因此不会增加主模型的训练参数量或计算开销.

4.3.6　PS-ATCNN 模型集成分析

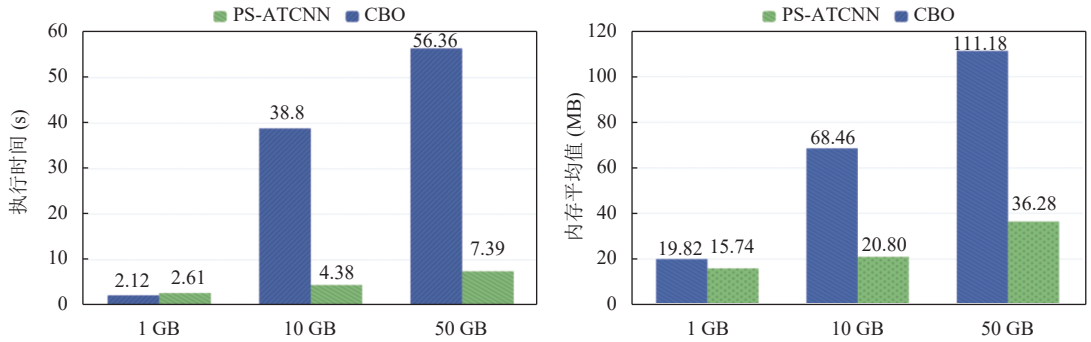
我们将负载分类模型 (PS-ACNN) 集成到 TiDB 数据库的查询优化模块中, 通过 PS-ACNN 识别 SQL 属于

OLTP 或 OLAP 类型, 将 OLTP 类型查询并强制路由至 TiKV 行存引擎执行, 将 OLAP 类型的查询强制路由到 TiFlash 列存引擎执行, 与 TiDB 原生基于成本优化器 (cost-based optimizer, CBO) 的智能执行计划选择策略形成对比实验. 为验证方案有效性, 我们首先采用 TPC-H 基准测试集, 构建 1 GB (SF=1), 10 GB (SF=10) 和 50 GB (SF=50) 这 3 种差异化数据规模, 从执行时延与内存效率两个维度评估系统性能. 接着使用 OLTP 和 OLAP 混合的工作负载对比 TiDB 原生 CBO 和不同配置的 PS-ATCNN 模型 (4/8/12 层 Transformer) 的负载识别准确率及其对数据库性能的影响.

表 7　SQLEncoder 中 Transformer 层数对模型训练和性能的影响

层数	训练时间 (h)	*Accuracy* (%)	*F-measure*	预测延迟 (ms/查询)
4	3.12	92.8	0.924	8.3
6	3.39	95.3	0.949	10.6
8	3.83	97.8	0.983	13.9
10	4.28	98.4	0.988	16.5
12	4.47	98.6	0.992	19.9

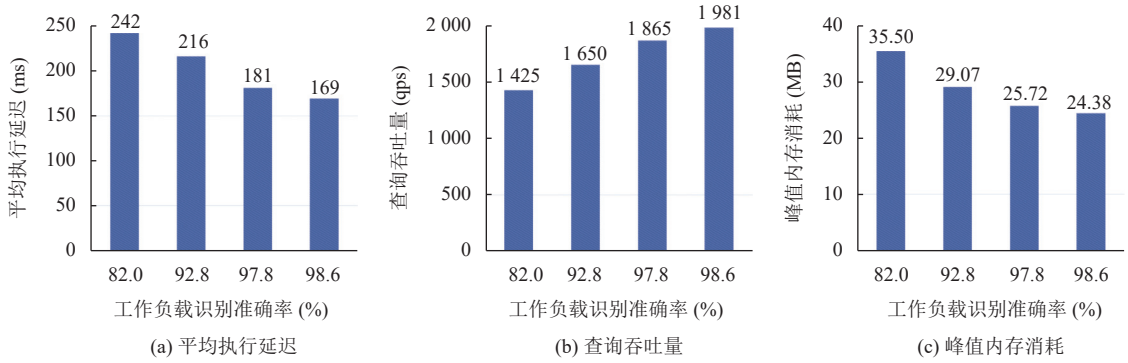
图 9(a) 为不同数据规模下两类优化策略的 TPC-H 的 22 条查询平均查询执行时间对比. 在 1 GB 轻量级数据场景中, PS-ACNN 平均执行时间为 2.61 s, 较 CBO 的 2.12 s 增加 23.11%, 由于小规模数据下 CBO 统计信息更新及时性较高, 而 PS-ACNN 的模型推理过程产生约 25 ms/查询的固定开销, 导致轻量查询出现边际性能损耗. 当数据规模扩展到 10 GB 和 50 GB 的数据量, PS-ACNN 的优化效果显著, 平均执行时间分别降低 88.77% 和 86.92%, 其核心归因于 PS-ACNN 有效规避 CBO 在大数据量下面临的统计信息滞后问题 (50 GB 场景统计信息更新延迟达 18 min), 并通过强制路由机制使 91% 的 OLAP 查询精准命中 TiFlash 列存引擎. 图 9(b) 为 TPC-H 执行过程中平均内存的使用量, 这 3 种数据规模下峰值内存消耗分别降低 20.69%、69.5% 和 67.3%, 列存引擎 TiFlash 对 OLAP 查询的内存管理更高效, 避免行存引擎 TiKV 因随机读放大导致的内存浪费. 实验表明 PS-ACNN 通过负载分类与执行路径的强绑定, 显著提升 TiDB 数据库在实际查询中的性能, 尤其在大规模 OLAP 场景中优势突出.



(a) TPC-H执行时间平均值　(b) TPC-H使用内存平均值

图 9　PS-ACNN 集成 TiDB 性能分析

通过图 10 可以看出, 在 TiDB 的 HTAP 混合负载场景下, 当负载识别准确率从 TiDB 原生 CBO 的 82% 提升至 12 层 PS-ATCNN 的 98.6% 时, 系统性能获得全方位提升: 平均查询延迟大幅降低 30.2%, 查询吞吐量显著提升 39.0%, 同时峰值内存消耗减少 31.3%. 这一系列优化主要得益于 PS-ATCNN 模型将错误路由率从 18% 降至 2%, 特别是避免了 OLAP 查询误入行存引擎 (TiKV) 所导致的资源浪费和 OLTP 查询误用列存引擎 (TiFlash) 造成的性能损耗, 有效优化了资源利用和系统性能.



(a) 平均执行延迟 (b) 查询吞吐量 (c) 峰值内存消耗

图 10 工作负载准确率对数据库性能的影响

5 总 结

本文提出一种基于树卷积神经网络结合注意力机制的智能识别 OLTP 和 OLAP 工作负载模型 PS-ATCNN. 模型通过输入 SQL 语句和其执行计划组成的工作负载, 识别出负载的类型, 从而为后续的查询执行过程提供关键的指导信息. 例如, 负载类型的识别可以帮助数据库系统动态选择适合的执行策略和优化方案. 特别地, 本文通过实验对比在执行计划中提取出识别 OLTP 和 OLAP 工作负载的特征, 并验证了所提取的特征具有很好的识别度, 进一步, 对提取的识别特征进行编码, 设计了一种编码方法, 能够将树结构特征和节点特征结合在一起生成树特征向量. 同时为 SQL 语句编码设计了预训练编码模型 SQLEncoder, 能够有效提取 SQL 语句语义信息和结构信息. 通过实验验证, 模型的准确率达到 97%, 并在多个数据集上验证了模型的鲁棒性. 最后, 将 PS-ATCNN 集成到 TiDB 数据库中, 通过与 TiDB 原始的 CBO 优化器对比 PS-ATCNN 能够优化数据库的查询性能.

References

[1] Özcan F, Tian YY, Tözün P. Hybrid transactional/analytical processing: A survey. In: Proc. of the 2017 ACM Int'l Conf. on Management of Data. Chicago: ACM, 2017. 1771–1775. [doi: 10.1145/3035918.3054784]

[2] Lahiri T, Chavan S, Colgan M, Das D, Ganesh A, Gleeson M, Hase S, Holloway A, Kamp J, Lee TH, Loaiza J, Macnaughton N, Marwah V, Mukherjee N, Mullick A, Muthulingam S, Raja V, Roth M, Soylemez E, Zait M. Oracle database in-memory: A dual format in-memory database. In: Proc. of the 31st IEEE Int'l Conf. on Data Engineering. Seoul: IEEE, 2015. 1253–1258. [doi: 10.1109/ICDE.2015.7113373]

[3] Li C, Markl V, Larson PÅ, Birka A, Hanson EN, Huang WY, Nowakiewicz M, Papadimos V. Real-time analytical processing with SQL server. Proc. of the VLDB Endowment, 2015, 8(12): 1740–1751. [doi: 10.14778/2824032.2824071]

[4] Huang DX, Liu Q, Cui Q, Fang ZH, Ma XY, Xu F, Shen L, Tang L, Zhou YX, Huang ML, Wei W, Liu C, Zhang J, Li JJ, Wu XL, Song LY, Sun RX, Yu SP, Zhao L, Cameron N, Pei LQ, Tang X. TiDB: A raft-based HTAP database. Proc. of the VLDB Endowment, 2020, 13(12): 3072–3084. [doi: 10.14778/3415478.3415535]

[5] Verbitski A, Gupta A, Saha D, Brahmadesam M, Gupta K, Mittal R, Krishnamurthy S, Maurice S, Kharatishvili T, Bao XF. Amazon aurora: Design considerations for high throughput cloud-native relational databases. In: Proc. of the 2017 ACM Int'l Conf. on Management of Data. Chicago: ACM, 2017. 1041–1052. [doi: 10.1145/3035918.3056101]

[6] Cao W, Li FF, Huang G, Lou JH, *et al*. PolarDB-X: An elastic distributed relational database for cloud-native applications. In: Proc. of the 38th IEEE Int'l Conf. on Data Engineering. Kuala Lumpur: IEEE, 2022. 2859–2872. [doi: 10.1109/ICDE53745.2022.00259]

[7] Zhang C, Li GL, Feng JH, Zhang JT. Survey of key techniques of HTAP databases. Ruan Jian Xue Bao/Journal of Software, 2023, 34(2): 761–785 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6713.htm [doi: 10.13328/j.cnki.jos.006713]

[8] Elnaffar S, Martin P, Schiefer B, Lightstone S. Is it DSS or OLTP: Automatically identifying DBMS workloads. Journal of Intelligent Information Systems, 2008, 30(3): 249–271. [doi: 10.1007/s10844-006-0036-6]

[9] Sen R, Ramachandra K. Characterizing resource sensitivity of database workloads. In: Proc. of the 2018 IEEE Int'l Symp. on High Performance Computer Architecture. Vienna: IEEE, 2018. 657–669. [doi: 10.1109/HPCA.2018.00062]

[10] Raza B, Kumar YJ, Malik AK, Anjum A, Faheem M. Performance prediction and adaptation for database management system workload using case-based reasoning approach. Information Systems, 2018, 76: 46–58. [doi: 10.1016/j.is.2018.04.005]

[11] Elnaffar S, Martin P, Horman R. Automatically classifying database workloads. In: Proc. of the 11th Int'l Conf. on Information and Knowledge Management. McLean: ACM, 2002. 622–624. [doi: 10.1145/584792.584898]

[12] Shaheen N, Raza B, Malik AK. A CBR model for workload characterization in autonomic database management system. In: Proc. of the 14th Int'l Conf. on Emerging Technologies. Islamabad: IEEE, 2018. 1–6. [doi: 10.1109/ICET.2018.8603615]

[13] Tang X, Wu S, Song ML, Ying SS, Li FF, Chen G. PreQR: Pre-training representation for SQL understanding. In: Proc. of the 2022 Int'l Conf. on Management of Data. Philadelphia: ACM, 2022. 204–216. [doi: 10.1145/3514221.3517878]

[14] Zhao Y, Cong G, Shi JC, Miao CY. QueryFormer: A tree Transformer model for query plan representation. Proc. of the VLDB Endowment, 2022, 15(8): 1658–1670. [doi: 10.14778/3529337.3529349]

[15] Sun J, Li GL. An end-to-end learning-based cost estimator. Proc. of the VLDB Endowment, 2019, 13(3): 307–319. [doi: 10.14778/3368289.3368296]

[16] Marcus R, Papaemmanouil O. Deep reinforcement learning for join order enumeration. In: Proc. of the 1st Int'l Workshop on Exploiting Artificial Intelligence Techniques for Data Management. Houston: ACM, 2018. 3. [doi: 10.1145/3211954.3211957]

[17] Ding BL, Das S, Marcus R, Wu WT, Chaudhuri S, Narasayya VR. AI meets AI: Leveraging query executions to improve index recommendations. In: Proc. of the 2019 Int'l Conf. on Management of Data. Amsterdam: ACM, 2019. 1241–1258. [doi: 10.1145/3299869.3324957]

[18] John LK, Maynard AMG. Workload Characterization for Computer System Design. New York: Springer, 2000. [doi: 10.1007/978-1-4615-4387-9]

[19] Yu PS, Chen MS, Heiss HU, Lee S. On workload characterization of relational database environments. IEEE Trans. on Software Engineering, 1992, 18(4): 347–355. [doi: 10.1109/32.129222]

[20] Zewdu Z, Denko MK, Libsie M. Workload characterization of autonomic DBMSs using statistical and data mining techniques. In: Proc. of the 2009 Int'l Conf. on Advanced Information Networking and Applications Workshops. Bradford: IEEE, 2009. 244–249. [doi: 10.1109/WAINA.2009.159]

[21] Zhao Y, Li ZDH, Cong C. A comparative study and component analysis of query plan representation techniques in ML4DB studies. Proc. of the VLDB Endowment, 2023, 17(4): 823–835. [doi: 10.14778/3636218.3636235]

[22] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I. Attention is all you need. In: Proc. of the 31st Int'l Conf. on Neural Information Processing Systems. Long Beach: Curran Associates Inc., 2017. 6000–6010.

[23] Hilprecht B, Binnig C. Zero-shot cost models for out-of-the-box learned cost prediction. Proc. of the VLDB Endowment, 2022, 15(11): 2361–2374. [doi: 10.14778/3551793.3551799]

[24] Wu ZN, Marcus R, Liu ZC, Negi P, Nathan V, Pfeil P, Saxena G, Rahman M, Narayanaswamy B, Kraska T. Stage: Query execution time prediction in Amazon redshift. In: Proc. of Companion of the 2024 Int'l Conf. on Management of Data. Santiago: ACM, 2024. 280–294. [doi: 10.1145/3626246.3653391]

[25] Shaheen N, Raza B, Shahid AR, Malik AK. Autonomic workload performance modeling for large-scale databases and data warehouses through deep belief network with data augmentation using conditional generative adversarial networks. IEEE Access, 2021, 9: 97603–97620. [doi: 10.1109/ACCESS.2021.3096039]

[26] Ma L, Van Aken D, Hefny A, Mezerhane G, Pavlo A, Gordon GJ. Query-based workload forecasting for self-driving database management systems. In: Proc. of the 2018 Int'l Conf. on Management of Data. Houston: ACM, 2018. 631–645. [DOI: 10.1145/3183713.3196908]

[27] Zhou XH, Sun J, Li GL, Feng JH. Query performance prediction for concurrent queries using graph embedding. Proc. of the VLDB Endowment, 2020, 13(9): 1416–1428. [doi: 10.14778/3397230.3397238]

[28] Juszczak P, Tax DMJ, Duin RPW. Feature scaling in support vector data description. In: Proc. of the 14th National Conf. on Artificial Intelligence and 9th Conf. on Innovative Applications of Artificial Intelligence. AAAI Press, 1997. 199–204.

[29] Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of deep bidirectional Transformers for language understanding. In: Proc. of the 2019 Conf. of the North American Chapter of the Association for Computational Linguistics. Minneapolis: ACL, 2019. 4171–4186. [doi: 10.18653/v1/N19-1423]

[30] Mou LL, Li G, Zhang L, Wang T, Jin Z. Convolutional neural networks over tree structures for programming language processing. In: Proc. of the 30th AAAI Conf. on Artificial Intelligence. Phoenix: AAAI, 2016. 1287–1296. [doi: 10.1609/aaai.v30i1.10139]

[31] Marcus R, Negi P, Mao HZ, Tatbul N, Alizadeh M, Kraska T. Bao: Making learned query optimization practical. In: Proc. of the 2021 Int'l Conf. on Management of Data. ACM, 2021. 1275–1288. [doi: 10.1145/3448016.3452838]

[32] Marcus R, Negi P, Mao HZ, Zhang C, Alizadeh M, Kraska T, Papaemmanouil O, Tatbul N. Neo: A learned query optimizer. Proc. of the VLDB Endowment, 2019, 12(11): 1705–1718. [doi: 10.14778/3342263.3342644]

[33] Woo S, Park J, Lee JY, Kweon IS. CBAM: Convolutional block attention module. In: Proc. of the 15th European Conf. on Computer Vision. Munich: Springer, 2018. 3–19. [doi: 10.1007/978-3-030-01234-2_1]

[34] Zhang LX, Chai CL, Zhou XH, Li GL. LearnedSQLGen: Constraint-aware SQL generation using reinforcement learning. In: Proc. of the 2022 Int'l Conf. on Management of Data. Philadelphia: ACM, 2022. 945–958. [doi: 10.1145/3514221.3526155]

[35] Chaudhuri S, Haritsa J, Leis V, Gubichev A, Mirchev A, Boncz P, Kemper A, Neumann T. How good are query optimizers, really? Proc. of the VLDB Endowment, 2015, 9(3): 204–215. [doi: 10.14778/2850583.2850594]

[36] Cahill MJ, Röhm U, Fekete AD. Serializable isolation for snapshot databases. In: Proc. of the 2008 ACM SIGMOD Int'l Conf. on Management of Data. Vancouver: ACM, 2018. 729–738. [doi: 10.1145/1376616.1376690]

[37] IBM Software Group. Telecommunication application transaction processing (TATP) benchmark description. 2009. https://tatpbenchmark.sourceforge.net/TATP_Description.pdf

[38] Massa P, Avesani P. Controversial users demand local trust metrics: An experimental study on Epinions. com community. In: Proc. of the 20th National Conf. on Artificial Intelligence. Pittsburgh: AAAI, 2005. 121–126.

[39] Jagadish HV, Zhou AY, Difallah DE, Pavlo A, Curino C, Cudre-Mauroux P. OLTP-bench: An extensible testbed for benchmarking relational databases. Proc. of the VLDB Endowment, 2013, 7(4): 277–288. [doi: 10.14778/2732240.2732246]

[40] Victoria L. DeepSeek-V3 and Qwen2.5-max in solving resource and process optimization problems. In: Proc. of the 2025 Int'l Teleconference of Young Researchers. Chişinău, 2025. 30–34. [doi: 10.53486/csc2025.06]

[41] Mikolov T, Chen K, Corrado G, Dean J. Efficient estimation of word representations in vector space. arXiv:1301.3781, 2013.

附中文参考文献

[7] 张超, 李国良, 冯建华, 张金涛. HTAP 数据库关键技术综述. 软件学报, 2023, 34(2): 761–785. http://www.jos.org.cn/1000-9825/6713.htm [doi: 10.13328/j.cnki.jos.006713]

作者简介

杨建文, 博士生, 主要研究领域为智能数据库, 机器学习, 时空数据管理.

丁治明, 博士, 研究员, 博士生导师, CCF 高级会员, 主要研究领域为数据库与知识库系统, 时态空间数据库, 物联网, 大数据与云计算, 信息检索.

严瑾, 博士, 主要研究领域为数据分析, 云边端数据处理, 通讯网络分析.

张秋鸿, 博士生, CCF 学生会员, 主要研究领域为机器学习, 智能数据库, 时空数据管理.

朱美玲, 博士, 主要研究领域为计算机视觉, 多模态大模型, 多源异构大规模数据智能分析处理, 流式大数据实时分析与关联发现.