SafeLoad：用于识别云数据仓库中内存过载查询的高效准入控制框架

Yifan Wu 浙江大学†
yifan.wu@zju.edu.cn

Yuhan Li Zhenhua Wang 阿里云计算
{lyh200442,wzh420090}@alibaba-inc.com

Zhongle Xie 浙江大学软件技术学院,
xiezl@zju.edu.cn

Dingyu Yang Ke Chen Lidan Shou 浙江大学†
{yangdingyu,ck,should}@zju.edu.cn

Bo Tang 南方科技大学
ltangb3@sustech.edu.cn

Liang Lin 阿里云计算
yibo.ll@alibaba-inc.com

Huan Li*Gang Chen 浙江大学†
{lihuan.cs,cg}@zju.edu.cn

摘要

内存过载是云数据仓库中常见的一种资源耗尽形式。当数据库查询因内存过载而失败时，不仅会浪费 CPU 时间等关键资源，还会扰乱核心业务流程的执行，因为内存过载（MO）查询通常是复杂工作流的一部分。如果能提前识别此类查询，并将其调度到内存充足的无服务器集群，就能避免资源浪费和查询执行失败。因此，云数据仓库需要一种准入控制框架，具备高预测精确率、可解释性、效率和适应性，从而有效识别内存过载查询。然而，现有准入控制框架主要关注 SLA 满足和资源隔离等场景，识别 MO 查询的精确率有限。此外，目前缺乏公开可用的、带有 MO 标签和工作负载的数据集，可用于训练和基准测试。为应对这些挑战，我们提出了 SafeLoad，这是首个专门用于识别 MO 查询的查询准入控制框架。同时，我们发布了 SafeBench，这是一个面向该任务的开源工业级基准测试，包含 1.5 亿条真实查询。SafeLoad 首先使用可解释的判别规则筛除内存安全查询。随后，它采用混合架构，整合全局模型和集群级模型，并辅以误预测校正模块来识别 MO 查询。此外，自调优配额管理机制会动态调整各集群的预测配额，以提高精确率。实验结果表明，SafeLoad 实现了最先进的预测性能，同时在线和离线时间开销较低。具体而言，与最佳基线方法相比，SafeLoad 的精确率最高提升 66%，与不使用 SafeLoad 的场景相比，浪费的 CPU 时间减少了 8.09×。

PVLDB 引用格式：
Yifan Wu、Yuhan Li、Zhenhua Wang、Zhongle Xie、Dingyu Yang、Ke Chen、Lidan Shou、Bo Tang、Liang Lin、Huan Li、GangChen。SafeLoad：用于识别云数据仓库中内存过载查询的高效准入控制框架。PVLDB，9(4)：715 - 727，2025。

*Huan Li 是通讯作者。
†区块链与数据安全国家重点实验室

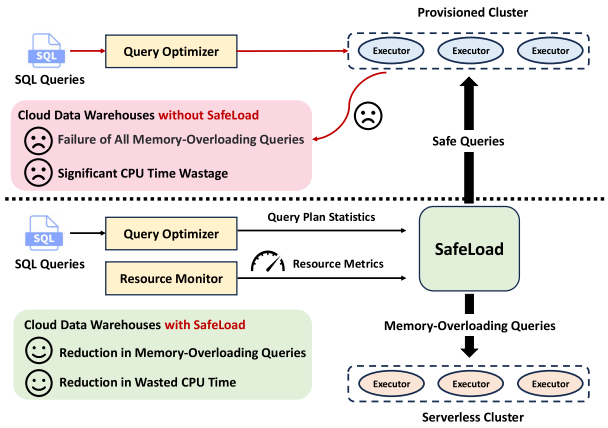


图 1：SafeLoad 在云数据仓库中的作用。

doi:10.14778/3785297.3785311

PVLDB 工件可用性：
源代码、数据和/或其他工件已在 https://github.com/SafeLoad-project/SafeBench 上提供。

1 引言

云数据仓库（例如 Redshift [1], 、Snowflake [8], 和 AnalyticDB [30],）的规模已达到前所未有的水平，每天处理数千亿条查询。这些系统通常在持续高负载下运行着数千个数据库集群，使资源争用成为常态而非例外。在此类环境中，有效的资源管理至关重要，因为不受管控的查询执行可能会危及系统稳定性和性能。查询准入控制 [19, 27] 在决定是否允许查询继续执行方面发挥着重要作用，其总体目标是防止资源耗尽并确保整体系统效率。

本作品依据知识共享署名-非商业性使用-禁止演绎 4.0 国际许可协议授权。访问 https://creativecommons.org/licenses/by-nc-nd/4.0/ 查看该许可协议副本。如需进行超出该许可协议范围的使用，请发送电子邮件至 info@vldb.org 获取许可。版权归所有者/作者所有。出版权已授权给 VLDB 基金会。VLDB 基金会会刊，第 9 卷，第 4 期，ISSN 2150-8097。doi:10.14778/3785297.3785311

内存过载是指查询需求超出可用容量，这一问题在云数据库中仍持续存在，并经常导致性能下降和执行失败。这些事件不仅会扰乱关键业务工作流，还会造成大量资源浪费。我们对从工业数据仓库 AnalyticDB [30] 收集的数据进行实证分析发现，由内存过载引起的执行失败每次故障事件最多会损失 17 个 CPU 小时。鉴于此类查询失败的规模和影响，一个能够在执行前主动识别 MO 查询的准入控制框架，将会为云服务提供商和用户带来显著益处。这促使我们设计了本文所介绍的专用于 MO 查询检测的准入控制框架。如图 1 所示，该框架名为 SafeLoad，可提前识别这些 MO 查询，并将其重定向至内存充裕的无服务器集群，以确保安全执行。如果没有这样的框架，MO 查询会被准入预配置集群，却因内存不足而失败，导致查询在失败前消耗的 CPU 时间被浪费。

结合云数据仓库处理 MO 查询的工业案例研究 [6, 7], 以及从开源系统中汲取的经验 [28], ，我们确定了理想的 MO 查询检测准入控制框架应实现的三个关键目标:

• **目标1：高预测精确率**。理想的解决方案应最大限度地准确识别 MO 查询，并将其重定向至内存充裕的无服务器集群。对于云服务提供商而言，这能确保系统稳定性；对于用户而言，则可避免查询失败并节省宝贵的资源。

• **目标 2：可解释性与效率**。理想的解决方案应向用户提供清晰的解释，使他们了解特定查询被拒绝的原因。此外，预测开销应尽可能低，以确保不影响查询延迟。对于云服务提供商而言，实施该解决方案应只产生极少的运维开销，从而使其能够广泛采用 [12]。

• **目标3：适应不同的数据库集群**。鉴于不同数据库集群之间的查询模式存在显著差异，理想的解决方案应能适应各集群的特定特征。这种适应性使解决方案能够更好地区分集群内的非 MO 查询和 MO 查询，从而减少错误预测的数量。

然而，现有的准入控制方法主要关注查询延迟 [19, 22, 24, 27],服务级别协议（SLA）合规性 [26], 资源隔离 [24], 和吞吐量优化 [19]。这些方法针对其他场景设计，因此无法直接应对识别 MO 查询这一挑战。最相关的研究 [19, 21, 29] 也仅限于估算查询的内存消耗。此外，如第 6 节所述，这些方法所达到的最高精确率仅为 0.5225，过低，无法支持实际部署。同时，可解释性和适应性这两个目标仍未得到探索。

接下来，我们讨论实现上述三个目标所面临的技术挑战。首先，对于**目标1**，由于缺乏深入的特征分析，目前尚不清楚哪些特征有助于识别 MO 查询。此外，由于缺少公开可用的 MO 查询带标签数据集，构建高效模型颇具挑战。此外，与非 MO 查询相比，MO 查询数量稀少，也进一步增加了准确识别 MO 查询的难度。具体而言，在我们分析的真实生产环境查询集中，非 MO 查询的数量最多可达 MO 查询的 20,276 倍。其次，对于**目标 2**，有必要探索高效且易于人类理解的启发式规则，以区分 MO 查询和非 MO 查询。这类启发式规则有助于用户理解准入控制框架中的决策过程。另一方面，虽然训练机器学习（ML）模型以捕捉 MO 查询复杂的特征分布至关重要，但仍需进一步研究，以实现高效训练并尽量降低预测开销。第三，对于**目标3**，不同数据库集群中的内存过载行为差异显著，导致某些集群出现明显的特征分布偏移。这种异质性使得在所有数据上训练的全局模型难以在所有集群中实现准确预测（见第3节）。

为应对这些挑战，我们提出了 SafeLoad，这是一种高效的内存过载查询准入控制框架。针对**目标1**，我们从查询和集群两个角度分析了七类特征。我们利用 SHAP [14],、KDE [16],和 t-SNE [23],量化并可视化特征重要性、分布和高维模式，为高精度检测奠定基础。为促进进一步研究，我们发布了 SafeBench，这是一个包含 1.5 亿条生产查询的开源基准。我们还设计了一种混合架构，将全局模型与集群级模型相结合，并由误预测校正模块加以增强。具体而言，SafeLoad 为每个集群构建基于FAISS的向量索引 [9] ，用于存储误分类的 MO 查询。新查询会与该索引进行匹配，以校正误预测，从而显著提高精确率。

针对**目标 2**，我们提出了一种判别式规则过滤模块，利用简单、易于人类理解的规则，高效地排除大多数非 MO 查询。该方法不仅降低了学习模型的计算开销，还通过为准入决策提供清晰且可执行的解释，增强了可解释性。针对**目标3**，我们提出了一种自调节配额管理模块，通过考虑模型预测熵和假阴性数量，动态估计每个查询的配额成本。配额是指被预测为正类的查询数量上限，该上限限制了转发至无服务器集群执行的查询数量。此外，该模块根据每个数据库集群中观察到的预测行为，实时调整配额成本。该模块减少了错误预测，并将实际发生内存过载的风险降至最低。

我们在工业级 SafeBench 基准测试上将 SafeLoad 与现有方法进行了比较。结果表明，与最佳基线方法相比，SafeLoad 的精确率最高提升了 66%，并且与未使用 SafeLoad 的情况相比，可将 MO 查询浪费的 CPU 时间最多减少 8.09× 。SafeLoad 将精心设计的技术整合到一个新颖、统一的 MO 查询检测框架中。我们的消融实验（见第6.3节）表明，每个组件都不可或缺，整合后所达到的准确率超越了任何单个组件的表现。我们进行了详细的消融实验，以考察各组件的贡献；结果证明，整体效果大于各部分效果之和，

每个组件对于实现框架的整体效能都发挥着不可或缺的作用。

本文的主要贡献总结如下：

• 我们提出了一个新的研究问题：在查询执行之前识别 MO 查询。该问题与现代数据仓库中遇到的现实问题相契合（第 2 节）。

• 我们提出了 SafeLoad，这是首个用于识别 MO 查询的查询准入控制框架。SafeLoad 的架构由三个关键组件构成：判别规则过滤、带校正的预测模型，以及自调节配额管理（第 4 节）。

• 我们发布了首个用于研究 MO 查询的工业级基准测试 SafeBench。该基准测试基于 AnalyticDB 收集的真实查询工作负载和集群级信息构建，包含 1.5 亿条查询。SafeBench 填补了这一研究领域的关键空白：此前一直缺乏公开可用的基准测试（第 5 节）。

• 我们在工业级基准测试 SafeBench 上，对 SafeLoad 与现有方法进行了深入比较。实验结果表明，SafeLoad 在预测开销较低的同时，实现了最先进的预测性能。我们还开展了消融研究，以验证 SafeLoad 各组件的必要性（第 6 节）。

此外，第3节介绍特征分析；第7节回顾相关工作；第8节总结全文并展望未来方向。

2 问题陈述

我们首先介绍在生产环境中观察到的内存过载查询的定义及其根本原因。随后，我们给出问题的形式化定义，并阐明与业务场景相关的关键指标。

内存过载查询。内存过载通常也称为内存不足（OOM）异常，是云数据库系统中的一个严重问题，常常导致查询失败和系统不稳定。当查询在执行过程中所需的内存超过可用系统资源时，就会发生此类异常，进而造成运行中断。形式化地，我们将内存过载查询（*MO* 查询）定义如下：

定义 1（内存过载查询，MO 查询）。在云数据库系统中，内存过载查询是指在执行过程中的任意时刻，其内存需求超过为用户的数据库集群分配的可用内存容量，从而导致内存过载的数据库查询。

根据成因和对系统的影响，MO 查询可分为两种类型：(1)查询内存需求过高：当分组、排序、窗口和哈希连接等特定阻塞算子需要大量内存来物化中间结果时，就会出现这种情况。当这些算子的内存请求超出预留内存池容量时，查询就会失败。类似地，表扫描等流式算子也可能导致内存过载：并发查询中的增量内存分配耗尽共享内存池时，就会触发内存过载。(2) 集群级内存压力：当所有节点的总内存不足以支持所有活动查询时，就会出现这种情况。在这种情况下，即使高内存消耗查询尚未单独超出其限制，系统也可能终止这些查询以维持稳定性。这两种内存过载

表1：关键术语说明，从实际部署的角度阐述其优缺

Term	定义 / 解释
真正例 (TP)	被正确预测为正例的 MO 查询；云服务提供商：通过卸载 MO 查询来防止数据库过载；用户：确保 MO 查询可靠执行。
真负例 (TN)	正确预测为负的非 MO 查询；云服务提供商：节省无服务器资源并最大限度降低成本；用户：避免不必要的卸载和执行延迟。
假阳性 (FP)	错误预测为正的非 MO 查询；云服务提供商：浪费无服务器资源并推高成本；用户：导致延迟并造成不必要的配额消耗。
假阴性 (FN)	被错误预测为负类的内存过载查询；云服务提供商：有数据库过载和性能下降的风险；用户：导致内存过载查询执行失败并浪费资源。
准确率	$\frac{\text{真正例}+\text{真负例}}{\text{真正例}+\text{真负例}+\text{假正例}+\text{假负例}}$。高准确率反映了模型在平衡资源与用户需求方面的整体可靠性，并能兼顾资源和用户需求。
精确率	$\frac{\text{真正例}}{\text{真正例}+\text{假正例}}$。云服务提供商：限制不必要的卸载，提高资源效率；用户：减少对非 MO 查询的干扰。
召回率	$\frac{\text{真正例}}{\text{真正例}+\text{假负例}}$。云服务提供商：确保大多数 MO 查询被卸载，避免过载；用户：最大限度地处理 MO 查询。
F1分数	$\frac{2\times\text{精确率}\times\text{召回率}}{\text{精确率}+\text{召回率}}$。平衡精确率和召回率，以实现最佳资源利用并提升用户满意度。

类型在查询计划和系统级指标中呈现出不同特征——包含复杂查询算子的查询更容易出现第一种情况，而较高的集群范围内的内存利用率则会增加第二种情况发生的可能性。

问题定义与应用场景。接下来，我们介绍本研究要解决的研究问题。我们工作的任务是：根据查询计划统计信息和集群资源指标，预测一条 SQL 查询在经过查询优化器处理后是否会发生内存过载。在云数据库中，用户在预配置集群上执行 SQL 查询。在执行之前，系统会分析每条查询，以预测其是否可能导致内存过载。被预测为将超出内存限制的查询会被重定向到拥有充足资源的无服务器集群，该集群通常由云服务提供商维护，以保障系统稳定性。然而，在无服务器集群上执行查询会产生额外的配额成本，以 CPU 时间计量。

为了尽量降低成本，提高预测精确率并减少错误调度至关重要。此外，云服务提供商应实施相应机制，以限制被调度查询的数量，防止因错误预测而导致的过度资源消耗（参见我们在第 4.4 节中的讨论）。

在此 背景下，我们正式将该问题定义如下 允许·

问题 1（MO 查询检测）。设 s 为一条 *SQL* 查询，$P(s)$ 表示其优化后的查询计划统计信息，$R(c)$ 表示执行 s 的集群 c 的资源指标。我们将MO 查询检测定义为一个二分类问题：

$$f : \{P(s), R(c)\} \rightarrow \{0, 1\},$$

其中，$f = 1$ 表示预测该查询会导致内存过载，$f = 0$ 表示预测该查询不会导致内存过载。

区分 MO 查询检测与内存估算。MO 查询检测与查询内存消耗估算相关，但目标和典型方法有所不同。前者被表述为一个二分类问题：

在当前系统状态下，判断查询是否会触发内存过载故障；后者则是预测峰值内存消耗的回归问题 [19, 21, 29]。基线启发式方法会估算查询的峰值内存消耗，并将其与可用内存进行比较；然而，我们的实证评估（第 6.2 节）表明，这种方法的准确性远低于 SafeLoad。造成这种差异的原因有：1）动态内存需求（瞬时峰值可能在任何时刻引发过载事件，不一定是在预测的峰值时）；2）时变可用性（可用内存预算可能会在查询提交与执行之间发生变化，因此依据初始快照进行比较并不可靠）。

关键绩效指标。导致内存过载的查询标记为正例，而成功的查询则标记为负例。表1结合业务场景介绍了假阳性、假阴性、精确率、召回率和 F1分数。在实践中，过多的假阳性会将非 MO 查询错误分类，浪费用户配额并增加成本；而过多的假阴性则会漏掉 MO 查询，给预配置集群带来压力并扰乱工作流。

从业务角度看，更高的精确率 可降低用户成本，而更高的召回率 则能提升资源利用和查询成功率。兼顾二者的*F1*分数提供了全面的衡量指标。因此，本研究强调**精确率** 和**F1 分数** 是部署时最**关键的指标** 。

3 特征分析

为解决 MO 查询检测问题，我们首先使用真实工作负载上的朴素预测器进行深入探索，重点关注第 1 节概述的三个目标。具体而言，我们的目标是定量识别指导 MO 查询预测的关键特征，以提高精确率（目标1）、可解释性和效率（目标 2），以及适应性（目标3）。

设置与基本统计数据。我们收集了来自 AnalyticDB [30], 的真实生产环境数据，即连续多个自然日采集的两个代表性数据集中的查询日志。第一个数据集包含 854 个集群和 5200 万条查询，其中 4,014 条为 MO 查询。第二个数据集包含 862 个集群和 5000 万条查询，其中 2,508 条被识别为 MO 查询。我们采用广泛使用的 XGBoost模型作为朴素预测器 [19, 21, 29]。模型在第一个数据集上训练，随后在第二个数据集上评估，以衡量其泛化性能。模型训练所用的输入特征取自查询优化器和资源监控器。优化器可提供查询级统计数据，而资源监控器则可提供数百项集群级或硬件级性能指标。

目标1的分析。第一个目标是提高预测器的精确率，这需要有效选择能够指示内存过载的特征。因此，我们使用 Shapley 加性解释（SHAP） [14], 进行了特征重要性分析，该方法量化了每个特征对模型预测的贡献。基于这项分析，我们将关键特征划分为七组，如观察1所述。这一分类体系既为我们的特征提取过程提供了指导，也为学习模型的设计提供了依据（参见第4.1节和第4.3节）。

观察1。 如表*2*所示，*SHAP* 分析确定了四个对检测内存过载查询至关重要的查询级特征组——算子数量、算子基数、内存密集型算子和执行配置，因为它们能够捕捉查询复杂度、数据量和执行行为。在集群级，三个特征组——资源指标、集群配置和OOM 指标 ——能够有效反映资源可用性和故障趋势，而这些是预测内存过载的关键因素。

目标 2 分析。为确保可解释性和效率，我们可以尝试确定是否存在少数几个能够有效区分 MO 查询与非 MO 查询的特征。这些特征可作为解释查询为何被预测为内存过载的线索，也可用于筛除非 MO 查询，以满足效率要求。我们重点分析第 2 节中提到的两类 MO 查询的共有特征，包括 Join、Window、聚合和排序节点的数量、TableScan 节点的总输出大小，以及集群此前是否发生过 OOM 事件，并评估这些特征的区分能力。为便于可视化，我们对这些特征应用了核密度估计（KDE） [16] ，结果如图 2 所示。图中，蓝色区域表示 MO 查询，红色区域表示非 MO 查询。 x 轴表示特征值， y 轴表示概率密度。我们观察到了一些明显的规律，例如，大多数非 MO 查询的 Join 节点数为 0 或 1，表扫描输出大小低于 1 MB，且对应的集群此前没有发生过 OOM 事件。因此，我们得出以下观察：

观察 2。 某些特征（例如 *Join* 节点的数量和 *TableScan* 的输出大小）对于区分非 *MO* 查询表现出很强的能力。

观察 2 揭示了使用简单、易于解释的特征规则，高效过滤掉大量非 MO 查询的潜力，从而降低 MO 查询检测的计算负担。这启发我们在应用训练得到的分类器之前设计判别规则过滤模块，详见第 4.2 节。

此外，我们评估了用于 MO 查询检测任务的业界启发式方法。与观察 2 一致，结构特征（例如 TableScan 节点的大小和 Join 节点的数量）包含了显著的区分信号，可用于区分 MO 查询和非 MO 查询。然而，我们的实证结果（第 6.2 节）表明，这些启发式方法虽然整体准确率非常高（最高达 0.99），但在 MO 类上的精确率却极低（低于 0.0258）；由此产生的假阳性会带来不可忽视的运营成本，并限制其在生产环境中的实用性。这些发现促使我们采用 SafeLoad框架的多层设计：先用启发式方法作为首轮过滤器，剔除明显的非 MO 查询，再使用轻量级机器学习分类器，大幅提高精确率并增强 MO 查询检测的可靠性。

观察 3. 尽管启发式规则速度快且易于解释，但它们不足以实现高精确率的 *MO* 查询检测。要进一步区分具有模糊特征、启发式规则无法辨别的 *MO* 查询，还需要机器学习模型。

目标3分析。对于最后一个目标，即对不同数据库（DB）集群的适应性，我们以集群内和集群间两种方式分析 MO 查询的特征分布。使用 t-SNE

表2：用于识别 MO 查询的非穷举特征集。详细说明请参阅 SafeBench 基准测试（第 5 节）。

类别	特征组	描述	数量
查询级别	算子数量	不同算子的数量（共 23 种），例如查询中连接节点的数量。	23
	算子基数	13 种算子的基数统计，每种算子涵盖 8 项指标，包括输出大小和行数的总值和最大值。这些算子是根据 MO 查询的错误报告选出的。	104
	内存密集型算子	通常与高内存使用量相关的算子（即连接、聚合、窗口、排序）的细粒度特征；包括连接的子节点基数，以及聚合中 varchar 类型分组键的总数。	19
	执行配置	查询执行调度期间的配置信息，包括查询并行度总数以及一项用于指示 SQL 查询是否显式指定了执行模式（例如批处理模式）的指标，这可能会影响内存行为。	2
集群级	资源指标	以 1 分钟为粒度收集的集群资源使用指标，包括每秒查询数、平均内存池利用率和 CPU 利用率。	8
	集群配置	已配置集群的静态配置，包括 CPU 核心数和资源组 ID。	6
	内存溢出指示器	前一天在相应集群中观察到的 OOM 事件数量。	1

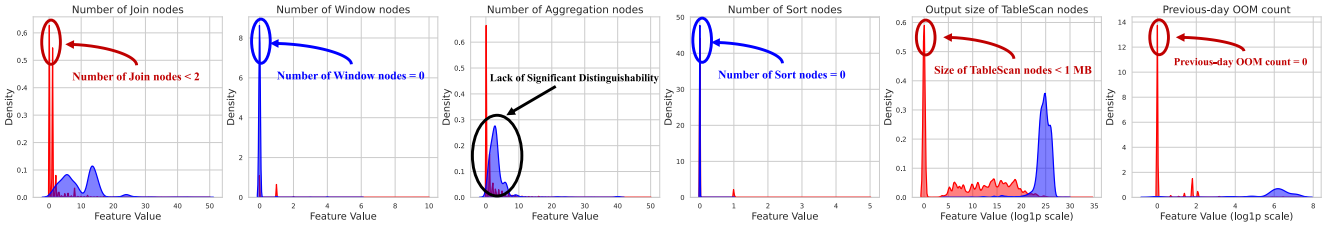


图 2：用于区分 MO 查询与非 MO 查询的选定特征的核密度估计。蓝色区域对应 MO 查询，红色区域则表示非 MO 查询。为处理大规模特征，最后两个子图采用了对数（$1 + x$）变换。

算法 [23], 将高维 MO 查询特征投影到二维空间中进行可视化。如图 3 所示，我们分析了 MO 查询数量最多的前 5 个 DB 集群*（使用从 A 到 E 的伪集群名称进行标记）。结果显示，MO 查询往往会形成许多小型集群。同一集群中的查询通常比不同集群中的查询表现出更高的特征相似度，但即使在同一集群内也存在差异。因此，我们提出如下观察：

观察 4。 *MO 查询的特征在不同 DB 集群间差异显著。此外，每个 MO 查询通常都有一小组与其高度相似的查询。*

观察 4 揭示了两个关键见解：(1) 集群间显著的特征差异意味着需要同时采用全局学习模型和集群专属学习模型。(2) MO 查询往往会形成规模较小、彼此高度相似的群组，这是因为用户在首次遇到内存过载故障后，常会重复发出相同的查询，尝试重试任务。此外，查询群组内部高度相似的特征为设计误预测校正机制提供了机会：时间上相邻的查询可以提供纠正信号。我们将在第 4.3 节进一步探讨这些思路。

此外，类似查询在真实工作负载中会多频繁地重现？这一问题对于验证我们的误预测校正机制至关重要，因为较高的重复率有助于从历史误预测中有效学习。我们的分析揭示了显著的季节性模式，为我们的设计提供了有力支持。首先，在查询层面，分析的数据集中有 77.78% 和 77.30% 的查询具有完全相同的 SQL

*根据 MO 查询数量选出排名前 5 的簇。

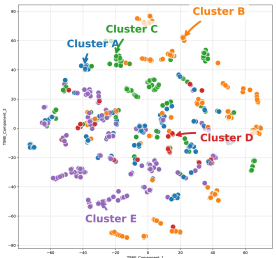


图 3：MO 查询数量最多的前 5 个簇中 MO 查询特征的 t-SNE 可视化。

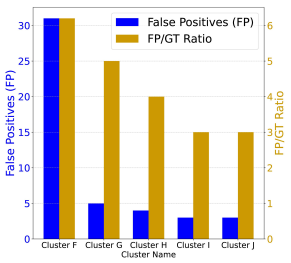


图 4：FP/GT 比率最高的前 5 个簇中的误分类行为分析。

文本。更值得注意的是，在分析查询级特征时，重复率达到 91.67% 和 91.29%，这表明不同的 SQL 文本往往呈现相同的统计模式。其次，结合查询级特征和集群配置特征后，重复率仍高达 80.97% 和 79.72%。这些频繁重现的查询—配置对表明，我们的校正机制能够有效利用历史误预测模式，为在设计中纳入历史反馈提供了有力的实证支持。

需要注意的是，对于集群间适应，MO 查询检测面临的一项关键挑战是假阳性造成的成本。因此，我们进一步分析了各数据库集群中的误分类行为，重点关注假阳性（FP）与真实标注（GT）比率最高、模型性能明显较差的前五个集群（暂称为 F 至 J）。例如，在集群 F

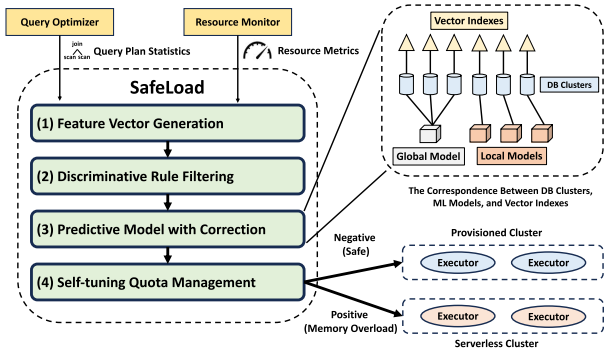


图 5：SafeLoad 概览。

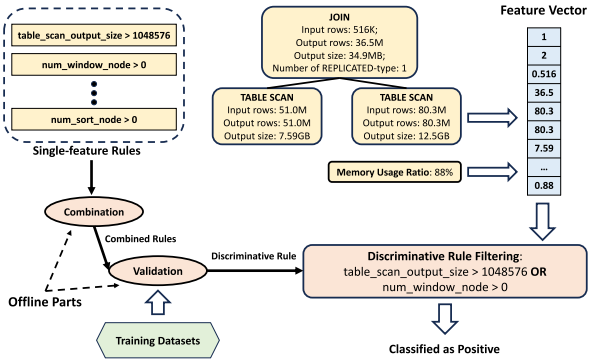


图 6：判别规则过滤示意图。

（图 4）中，模型仅正确识别出 5 个真实 MO 查询，却将 31 个非 MO 查询错误地预测为 MO 查询。FP/GT 比率因此达到 6.2，导致配额消耗比必要水平高出 6.2 倍。同样，G 至 J 集群中的假阳性数量也明显多于真实 MO 查询，严重程度各不相同。为深入了解这一现象，我们引入并考察模型的置信度分数，该分数表示某个查询被分类为正例的可能性。平均而言，真正例的置信度分数为 0.99，假阳性为 0.91，假阴性为 0.04，真阴性为 0。这些分数清晰地表明，正确预测与错误预测之间存在置信度差异。

观察 5。 学习模型在某些集群中可能呈现较高的假阳性率（或假阳性/真实值比率）。此外，置信度分数可以作为识别错误预测的有效指标。

观察 5 提出了两项提升模型性能的关键原则：(1) 需要一种稳健的机制，通过限制学习模型做出的正向预测数量，来抑制过多的假阳性。(2) 越接近 0 或 1 的置信度分数，越能表明预测正确。基于这些认识，我们设计了一种自调优配额管理机制，详见第 4.4 节。

4 SafeLoad 设计

4.1 概述

如图 5 所示，SafeLoad 运行于查询优化器与执行器之间，在执行前拦截查询，以预测潜在的内存过载。如果检测到风险，它会将查询重新路由至内存充裕的无服务器集群，从而避免执行失败和资源浪费（例如 CPU 时间或内存）。SafeLoad 由四个关键模块组成，这些模块逐步预测内存过载风险：

(1) 特征向量生成。该模块通过结合查询计划统计信息（例如算子基数、节点数）和集群级资源指标（例如内存利用率），为每个查询构建多维特征向量。这些特征既用于训练，也用于推理。

(2) 判别规则过滤。该模块利用经过特征化的训练数据中的带标签数据，推导出启发式规则，在过滤非 MO 查询的同时保留大多数 MO 查询。结合逻辑运算符（and/or）后，这些规则可用于排除明显的负例，从而最大限度地降低开销并确保可解释性。

(3) 带校正的预测模型。如图 5 右上角所示，SafeLoad 采用混合架构：训练样本不足的集群共用一个全局模型，而其他集群则使用各自的局部模型。此外，每个集群都会维护一个用于误预测校正的向量索引。对于未被规则过滤的查询，SafeLoad 会应用集群专属或全局的基于树的模型来评估内存过载风险。为增强鲁棒性，每个集群都会维护一个包含历史假阴性的内存向量索引。与以往误分类的 MO 查询高度相似的查询会被归类为正例，从而绕过预测。

(4) 自调优配额管理。SafeLoad 动态调整集群特定配额，以缓解不确定预测导致的过多假阳性。配额成本基于预测熵和历史假阴性，在配额有限的情况下限制低置信度预测，同时在资源允许时放行临界查询，从而确保工作负载稳定性。

SafeLoad 框架的通用性与可移植性框架。SafeLoad 的核心设计原则是其与系统无关的架构，这使其能够广泛适用于异构数据库引擎。该框架刻意避免依赖特定的系统内部机制。相反，SafeLoad 的各个模块建立在广泛可用且可迁移的抽象之上。尽管采用了优化器派生特征，但这些特征符合 ANSI SQL（ISO/IEC 9075）标准，因此在开源系统（例如 Presto）和商业数据仓库（例如 Redshift、Snowflake）中都有直接对应项。此外，SafeLoad 还基于用户驱动的工作负载规律，例如查询重复模式；这些规律与特定的系统内部机制无关。

4.2 判别规则过滤

如观察 2 所述，可以根据特定特征有效地区分 MO 查询和非 MO 查询。例如，如果表扫描算子扫描的数据少于 1 MB，则该查询不太可能导致内存过载。SafeLoad 通过分析训练集中的正样本和负样本，利用这些洞见生成判别规则，具体定义如下。

具体而言，单特征规则 涉及基于阈值的比较，例如“表扫描输出大小 >1 MB”，形式化表示为 table_scan_output_size $>$ 1048576。利用观察1，我们确定了三种对于检测 MO 查询非常有效的关键特征类型：算子数量、算子基数和 OOM 指示器（说明见表2）。SafeLoad 构建了一个库

其中包含针对这些特征的单特征规则，该库可通过 [2] 公开获取。特征规则及其阈值源自广泛的业务部署经验。判别规则是由这些单特征规则通过 and/or 运算符连接而成的逻辑组合。在本研究中，判别规则具体由最多四条单特征规则组合构成，因为规则生成过程可在一小时内完成，这对于实际部署而言是可接受的离线构建时间。

判别规则的生成过程如图 6 所示。SafeLoad 将查询计划统计信息和集群资源指标编码为特征向量。为确保规则选择的稳健性，SafeLoad 采用以下标准：(a) 高正样本保留率：对于每种特征类型，选择一条单特征规则，在尽量减少负样本保留的同时，保留大多数正样本（例如超过 95%）。(b) 高精确率规则：额外选择一条负样本保留率极低（例如低于 3%）且正样本保留率最高的规则。在验证子集上（见第 5 节）穷举评估这四条规则的逻辑组合（使用 and/or）。最终判别规则从非 MO 查询过滤率最高的规则组合中选出，同时最大限度地保留 MO 查询。

最终，生成的判别规则将在线应用于初始过滤。被识别为非 MO 的查询会立即标记为负样本，跳过进一步分析。符合该规则的潜在 MO 查询则会被转发至带校正的预测模型，以进行进一步评估。

4.3 带校正的预测模型

基于规则的过滤会移除容易识别的非 MO 查询，之后，剩余的复杂模式则由学习模型处理，这类模型擅长捕捉复杂的查询行为 [19, 21, 25, 29]。为进一步提升检测精确率，SafeLoad 采用混合架构，将集群专属模型与全局模型相结合，并辅以误预测校正机制。

用于 MO 查询检测的基础模型。SafeLoad 利用轻量级基于树的模型进行 MO 查询分类，默认选择 XGBoost，因为它具有高精确率、低推理延迟，并且已在查询性能建模方面证明了其有效性 [19, 21, 25, 29]。如第6.3节所示，XGBoost 的表现始终优于多层感知机、随机森林和 LightGBM 等替代方案。该模块采用与模型无关的设计，因此在需要时可兼容其他分类器。

一项关键挑战是训练数据 [17], 中存在严重的类别不平衡：非 MO 查询的数量远远超过 MO 查询（例如，SafeBench A2 中的比例为 1:20,276）。不过，判别式规则过滤模块通过消除大量非 MO 查询缓解了这一问题，显著改善了类别平衡（例如，将 SafeBench A2 中的比例降至 1:464）。这一预处理步骤不仅加快了训练速度，还增强了模型有效区分 MO 查询的能力。

局部模型与全局模型的混合架构。为确保在各种工作负载和集群配置下实现良好的泛化能力，SafeLoad 采用混合架构（如图 5 右上角所示）：部分集群拥有自己的局部模型，而其余集群共享一个全局模型。对于具有充足带标签数据的集群（例如训练集中有超过 100 个正样本），会训练局部模型以捕捉集群特定模式。对于正样本较少的集群，则使用基于所有集群聚合数据训练的全局模型。该设计确保了在数据丰富和数据稀缺场景下都能进行可靠的预测。

基于向量搜索的误预测纠正。观察 4 揭示，每个 MO 查询通常都有少量高度相似的对应查询。为利用这一模式，SafeLoad 引入了基于向量搜索的机制，以发现更多真实 MO查询，从而降低将 MO 查询误判为非 MO 查询的风险。为减少假阴性，SafeLoad 集成 FAISS [9]，为每个数据库集群维护一个内存向量索引。该索引存储在线预测期间被误判为非 MO 查询的 MO 查询的特征向量，以便对未来查询采取纠正措施。初始时，向量索引为空，并会随着假阴性的发现而动态更新。当某个查询被学习模型误判为非 MO 查询后执行并触发内存过载时，其特征向量会被添加到集群专属索引中（见图 7(a)）。这使系统能够记录和跟踪具有过载特征的查询。

在未来的预测过程中，SafeLoad 使用余弦相似度，将传入查询的特征向量与索引中存储的向量进行比较，并采用严格的阈值。该阈值通常高达 0.9999，确保只有与先前导致过载的查询差异极小的查询才会匹配；这在生产环境中特别重要，因为用户经常会对失败的查询稍作修改后重试。如果发现高度相似的向量，系统便绕过学习模型，直接将该查询分类为 MO 查询（见图7(b)）。通过根据运行时反馈动态调整，该机制提高了检测精确率。

4.4 自调优配额管理

SafeLoad 集成了一种自调优配额管理机制，旨在适应不同数据库集群的特征，从而提高预测精确率。在现实系统中，这种机制尤其关键，因为假阳性频发会导致资源使用效率低下，并削弱人们对自动化决策的信任。

动机与关键场景。该模块为每个传入的查询分配配额成本，并根据系统的预测置信度和历史性能动态调整该成本。当某个查询被预测为正例时，它会消耗集群可用每日配额的一部分。如果剩余配额不足以承担接受该查询的成本，SafeLoad 会自动将其归类为负例。这一机制在两种场景下尤其有价值。第一，在配额短缺且预测具有高置信度的情况下，大多数传入查询都是真阳性，系统会根据近期假阴性的累计数量按比例降低配额成本。此项调整使系统能够接受更多临界样本，从而提高召回率和整体F1分数。第二，在配额盈余且预测置信度较低的情况下，系统会提高高不确定性预测的成本，抑制对不可靠查询的接受，并降低假阳性的可能性。这些动态调整确保

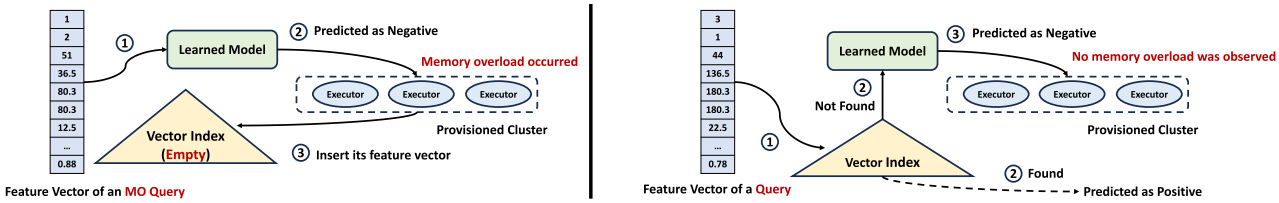


图 7：SafeLoad 中的误预测校正。

系统能够在不同集群之间有效平衡精确率和召回率。

配额计费机制。SafeLoad 中的配额管理机制会动态调整接受正类预测的成本，以平衡资源利用率和预测可靠性。

具体而言，每个数据库集群 $i \in \mathrm{I}$ 都会分配一个初始每日配额 Q_i^{daily}，其大小与前一天在该集群上观察到的 MO 查询数量成正比。集群 i 在时间 t 的剩余配额（记为 $Q_i(t)$）初始化为 $Q_i(0) = Q_i^{\text{daily}}$，并会随着查询被接受而递减。

对于在时间 t_j 到达、关联集群 i 的传入查询 s_j，学习模型会分配一个预测标签 $\hat{y}_j \in \{0,1\}$，以及相应的置信度分数 $p_j \in [0,1]$，该分数表示模型对 s_j 属于正类的置信程度。

接受正类预测（即 $\hat{y}_j = 1$）的决策由动态计算的配额成本决定，记为 $C(s_j)$。该成本取决于样本特征，以及对应集群的历史预测可靠性。具体而言，$C(s_j)$ 由以下因素决定：

（1）预测熵：模型对 p_j 样本属于正类的置信度评分。使用二元熵函数对预测的不确定性进行量化：

$$H(p_j) = -p_j \log_2(p_j) - (1-p_j)\log_2(1-p_j). \tag{1}$$

低熵（接近 0）表示高置信度预测（例如，p_j 接近 0 或 1），而较高的熵则表示不确定性（例如，p_j 接近 0.5）。不确定的预测会产生更高的配额成本，从而抑制系统接受不可靠的决策。预测熵的设计也与观察 5 相符：越接近 0 或 1，模型越有可能做出正确预测。p_j

（2）假阴性计数（FNC）：$FNC_i(t)$ 记录截至时间 t，集群 i 的假阴性累计数量。假阴性计数越高，配额成本越低，从而优先提升召回率并提高预测覆盖率。

配额成本 $C(s_j)$ 由这些因素加权组合计算得出，确保系统能够根据预测置信度和集群特定的性能动态调整，从而有效平衡精确率与召回率：

$$C(s_j) = \max\left(1 + \gamma \cdot H(p_j) - \beta \cdot \mathrm{FNC}_i(t), C_{\min}\right). \tag{2}$$

其中，$\gamma > 0$ 是预测不确定性惩罚的权重（熵 $H(p_j)$），$\beta \geq 0$ 用于调整假阴性计数（$\mathrm{FNC}_i(t)$），而 $C_{\min}$ 则确保最低配额成本，以维持正值。在理想条件下——模型置信度高且集群中不存在假阴性时，配额成本默认为 1。

表 3：SafeBench 的规格说明。

子集	#（查询）	#（簇）	#（位置）	G1	G2
A1	52,080,130	854	4,014	训练	
A2	50,856,068	862	2,508	测试	训练
A3	48,821,677	856	2,331		测试

若集群 i 在时间 t_j 的剩余配额 $Q_i(t_j)$ 大于或等于成本 $C(s_j)$，则接受预测 $\hat{y}_j$，即当且仅当 $Q_i(t_j) \geq C(s_j)$ 时接受 $\hat{y}_j = 1$。接受后，配额更新为 $Q_i(t_j + \Delta t) = Q_i(t_j) - C(s_j)$。若拒绝，配额保持不变：$Q_i(t_j + \Delta t) = Q_i(t_j)$。

该机制会动态平衡预测成本、模型置信度与集群性能。通过惩罚不确定且易出错的预测，并在需要时优先考虑精确率，该机制能够提升决策质量，最大限度地减少假阳性和漏检阳性。这既增强了 SafeLoad 的运行稳健性，也提高了其整体精确率。

5 SAFEBENCH

本节介绍 SafeBench，这是一个工业级基准测试，现已开源，旨在推动学术界对 MO 查询进行预先识别的研究。据我们所知，SafeBench 是首个专为此目的开发的基准测试。该基准测试可在 [2] 公开获取。

基准测试配置。表 3 概述了 SafeBench 的规格。SafeBench 基于从阿里云数据仓库系统 AnalyticDB [30] 收集的真实生产环境数据构建。该数据集涵盖连续三天的数据，分为三个不同的子集：SafeBench A1（第 1 天）、A2（第 2 天）和 A3（第 3 天）。这些子集包含在 800 多个生产环境数据库集群中执行的超过 1.5 亿条分析查询。每条查询的平均 CPU 时间分别为 7.9 秒（A1）、8.6 秒（A2）和 8.6 秒（A3）。为评估 MO 查询检测方法的有效性，我们将数据集划分为两个评估组。G1 组使用 A1 进行训练、A2 进行测试；G2 组使用 A2 进行训练、A3 进行测试。默认情况下，SafeBench 使用前一天的数据进行训练，并使用当天的数据进行测试。

MO 查询的标注。真实标签来自对表 3 工作负载进行的基线运行，期间未采取任何预防措施。在此次运行中，如果某个查询的执行日志包含 OOM 错误或其他与 MO 相关的故障，就会将其标记为 MO。随后，我们使用运行时跟踪对这些标注进行交叉验证：如果某个查询在执行期间的任意时刻，其内存需求超过可用内存，

则确认该查询为 MO。这个两步流程确保即使在集群级条件变化的情况下，也能准确识别。尽管 MO 查询数量较少（A1 中为 4014 个，A2 中为 2508 个，A3 中为 2331 个），但它们造成的成本却不成比例地高，平均分别浪费 0.56、0.59 和 0.51 个 CPU 小时。这凸显了准确检测这些影响重大的查询的重要性。

特征集。SafeBench 从查询级和集群级两个角度，对每个查询进行详细分析。每个查询都有唯一的查询 ID 和提交时间戳，并附带集群名称、总 CPU 时间等元数据，以及一个用于标记是否发生内存过载的二元标签。每个查询还由一个 163 维特征向量表示，其中包含 147 个查询级特征和 16 个集群级特征。尽管这些特征源自 AnalyticDB，其查询级特征严格遵循 ANSI SQL 抽象，而集群级指标则仅包含基础的、与系统无关的信号（例如 CPU 负载、内存消耗比）。表2对这些特征进行了分类汇总，详细定义可参阅开源 SafeBench 代码仓库 [2]。

6 评估

我们从内存过载查询预测性能和时间开销两方面评估 SafeLoad，并围绕以下四个评估问题展开：**EQ1** 预测性能（第 ~~6.2 节）：在识别内存过载查询方面~~，SafeLoad与现有基线方法相比表现如何？**EQ2** 消融实验（第6.3节）：SafeLoad的各个组件分别有何贡献？**EQ3** 时间开销（第 6.4 节）：SafeLoad的时间开销是否具有竞争力，并适用于真实世界的部署？**EQ4** 参数敏感性（第 6.5 节）：配额管理中的参数 γ 和 ~~β在合理范围内是否稳健，并且~~能否推广到不同设置？

6.1 实验设置

硬件配置实验均在一台配备 Ubuntu 22.04.5、128 个虚拟 CPU（Xeon Platinum 8369B @ 2.70GHz）和 992GiB 内存的中央处理器服务器上进行，用于大规模处理和模型评估。**基线方法**。由于目前没有专门用于识别内存过载查询的现有解决方案，我们基于相关先前研究构建了七种基线方法：
(1) **基于 SQL 的 XGB**：将从 SQL 语句中进行的词频-逆文档频率特征提取与 XGBoost 相结合，其灵感来自 Twitter [21] 和 CASA [29], 关于内存消耗预测的两项研究。(2) **Plan-based-XGB**：使用优化器提供的查询计划统计信息训练 XGBoost模型，其灵感来自 Auto-WLM [19], 这一 RedShift 查询延迟估算框架，并将其应用于内存消耗估算。(3) **MART**：Li 等 [13] 提出了 MART，该方法使用提升回归树 [5]基于执行计划特征预测查询 CPU 和 I/O 时间。我们对 MART 进行调整，用于识别内存过载查询。(4) **LearnedWMP**：LearnedWMP [11] 预测并发查询的内存使用量。我们使用其表现最佳的变体 LearnedWMP-DNN，作为识别内存过载查询的基线方法。(5) **LS-Rule (Large-Scan-Rule)**：实现 Snowflake 的启发式规则 [20]，当查询符合以下条件时将其分类为正例：

查询计划包含聚合或过滤算子，且表扫描大小超过日均值。(6) **判别规则**：直接应用 SafeLoad 中的判别式规则过滤模块（参见第 4.2 节）。(7) **向量相似度**：根据训练集中的 MO 查询特征向量构建向量索引。如果查询的余弦相似度超过 0.9999，则将其分类为正例，与 SafeLoad 使用的阈值一致。

在消融研究（第6.3节）中，将 SafeLoad 用于内存过载查询分类的 XGBoost模型替换为多层感知机、LightGBM 和随机森林，分别称为 SafeLoad-MLP、SafeLoad-GBM 和 SafeLoad-RF。进一步的实验验证了 SafeLoad 各组件的有效性。为确保公平性，Plan-based-XGB 和 MART 在训练和推理中使用的查询计划统计信息均继承自 SafeLoad 特征向量中包含的相同查询级特征（见表2）。

实现细节。为确保公平，所有使用 XG-Boost 的基线方法均采用相同的超参数配置：500 轮提升、学习率为 0.05，最大树深度为 5。对于其他分类器：(a) 多层感知机模型由 4 个全连接层组成，隐藏维度分别为 256、128 和 64，使用 ReLU 激活函数和丢弃率为 0.3 和 0.2 的 Dropout，随后通过 Sigmoid 输出层进行二分类；(b) 随机森林配置了 100 棵决策树；(c) LightGBM 训练 100 轮提升，学习率为 0.05，并采用类别平衡权重处理数据不平衡。根据第 6.5 节的敏感性分析，配额管理（公式2）中的参数默认设为 $\gamma = 1$ 和 $\beta = 0.5$。最低配额成本 $C_{\min}$ 固定为 0.1，以确保取值非负。

评估协议。如第 2 节所述，MO 查询检测被建模为一项二分类任务，并使用准确率、精确率、召回率和 F1分数衡量性能，其中精确率和 F1分数是关键指标。我们还评估了 MO 查询的 CPU 时间节省，以及在线预测和离线构建的时间开销。我们在 SafeBench 基准测试的 G1 和 G2 上评估 SafeLoad 及所有基线方法（见表 3）；该基准测试支持确定性重放。每条工作负载轨迹都会在配置完全相同的集群上重放，确保每次查询到达时集群条件均可比较。MO 标签仅根据基线轨迹生成一次（需求超过可用内存，并通过 OOM 日志交叉核验），并一致地应用于所有运行。在评估期间，如果调度决策不同（例如 SafeLoad 与某个基线方法之间的差异），则会分叉重放，使工作负载和集群状态保持不变，仅改变决策。此外，还会以 1 分钟为间隔记录集群级特征快照，以支持真正例/假正例识别。

6.2 预测性能比较（EQ1）

我们从两个关键角度评估 SafeLoad 相较于基线方法的预测性能：（1）使用精确率和 F1分数等指标衡量分类有效性；（2）提高所有查询的效率，减少浪费的CPU时间。

分类有效性。表4总结了 SafeLoad 和基线方法在 G1 和 G2 测试数据集上的分类结果，包括精确率、召回率和 F1分数等标准指标。

表4：SafeLoad 与基线方法在 SafeBench G1 和 G2 数据集上的分类性能对比。

SafeBench G1	TN	TP	FN	假i				F1
基于 SQL 的 XGB	50,852,750	174	2,334	810	0.999938	0.1768	0.0694	0
Plan-based-XGB	50,852,138	1 ,350	1 ,158	1,422	0.999949	0.4870	0.5383	0
MART	50,851,233	143	2 ,365	2,327	0.999908	0.0579	0.0570	0
LearnedWMP	50,849,616	941	1 ,567	3,944	0.999892	0.1926	0.3751	0
判别规则 49	,742,527	2 ,393	115	1,111,033	0.978151	0.0021	**0.9541**	0.0043
LS-Rule	50,823,588	795	1 ,713	29,972	0.999377	0.0258	0.3170	0
向量相似度	39,237,558	1,692	816	11,616,002	0.771575	0.0001	0.6746	0.0003
SafeLoad	50,853,048	2 ,203	305	512	**0.999984**	**0.8114**	0.8784	**0.**
SafeBench G2	**TN**	**TP**	**FN**	**假i**				**F1**
基于 SQL 的 XGB	48,816,248	7	2,324	3,098	0.999889	0.0023	0.0030	0
Plan-based-XGB	48,818,272	1 ,175	1 ,156	1,074	0.999954	0.5225	0.5041	0
MART	48,816,258	1 ,259	1 ,072	3,088	0.999915	0.2896	0.5401	0
LearnedWMP	48,813,028	867	1 ,464	6,318	0.999841	0.1206	0.3719	0
判别规则 47	,843,863	2 ,258	73	975,483	0.980018	0.0023	**0.9687**	0.0046
LS-Rule	48,788,289	641	1 ,690	31,057	0.999329	0.0202	0.2750	0
向量相似度	39,337,115	1,241	1,090	9,482,231	0.805756	0.0001	0.5324	0.0003
SafeLoad	48,818,865	2 ,076	255	481	**0.999985**	**0.8119**	0.8906	**0.**

基于 *SQL* 的 *XGB* 使用词频-逆文档频率从 SQL 语句中提取关键词特征向量，并应用 XGBoost 进行二分类。然而，它在 SafeBench G1 和 G2 上表现不佳，F1 分数分别为 0.0997 和 0.0026，这表明仅使用 SQL 特征不足以检测内存过载查询，尤其是在未纳入集群资源指标的情况下。*Plan-based-XGB*利用 SafeBench 数据集中的 148 个查询计划特征，在 G1 和 G2 上分别取得了 0.5114 和 0.5131 的 F1 分数，表现显著更佳。虽然这证明了查询计划特征的重要性，但由于无法纳入集群级特征，近一半的内存过载查询仍被错误分类。

MART 和 *LearnedWMP* 使用执行计划特征训练提升回归树和多层感知机，以预测查询内存消耗。在我们的实验设置中，这两种方法都通过比较预测的峰值消耗与可用内存来推断内存过载。MART 在 SafeBench G1 和 G2 上的 F1 分数分别为 0.0575 和 0.3771，而 LearnedWMP 的得分分别为 0.2545 和 0.1822。这种间接的替代决策规则并非针对检测目标设计，因此在 MO 查询检测中精确率较低。

LS-Rule 和 判别规则 是轻量级启发式过滤器，能够有效过滤非 MO 查询，从而达到较高的准确率（>0.9）。然而，它们的 F1 分数极低（<0.05），因此不适用于生产环境。

值得注意的是，判别规则 在 G1 和 G2 上保留了超过 93% 的 MO 查询，并有效过滤了非 MO 查询。该规则通过组合单特征规则生成，旨在最大限度保留 MO 查询，同时尽量减少非 MO 查询的保留。尽管分别在 G1 和 G2 上训练，该规则仍保持一致，展现出良好的鲁棒性。最终规则 (((特征_18 > 1 或特征_63 > 0) 且特征_52 > 1048576 且特征_161 > 0)) 将特征映射到直观的指标：Join 节点、表扫描输出大小、Window 节点以及前一日集群内存溢出。其可解释性使用户能够优化 SQL 查询，并主动避免 MO 查询。在 G1 和 G2 验证集上，正例保留率分别达到 93.77% 和 95.41%，负例保留率分别为 2.07% 和 2.18%，彰显了该规则在保留 MO 查询的同时过滤非 MO 查询的有效性。

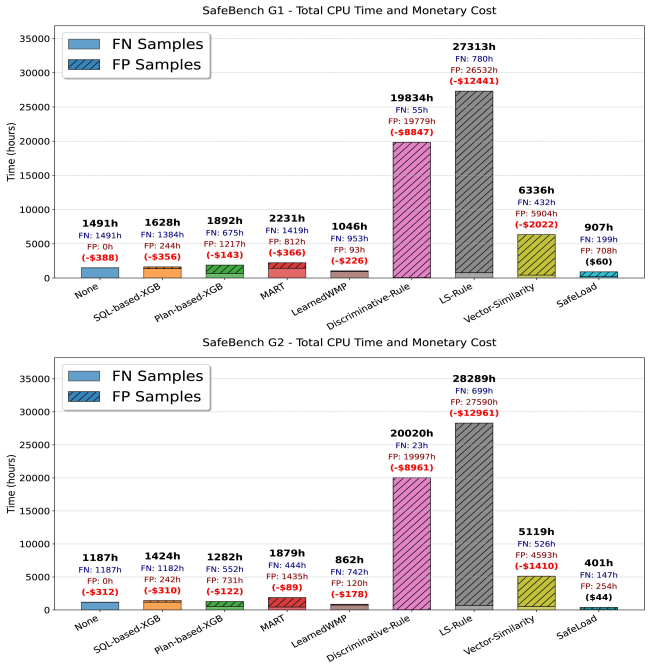


图 8：所有基线方法浪费的 CPU 时间和货币成本。各基线方法相对于 SafeLoad 的货币成本差额（G1 中为 60 美元，G2 中为 44 美元）。

向量相似度 的精确率较低，原因是假阳性过多：它将余弦相似度高于 0.9999 的查询判定为正例，因此在查询高度相似的环境中，常会将非 MO 查询误分类。这凸显了基于阈值的方法的局限性，也表明需要借助学习模型更好地捕捉复杂的特征分布并提高精确率。

总体而言，SafeLoad 达到了最先进水平的预测性能，与最佳基线方法相比，精确率最高提升了 66%，并将 MO 查询浪费的 CPU 时间减少了多达 8.09× 。

浪费的 CPU 时间和货币成本的减少。最大限度地减少与预测错误相关的 CPU 时间和货币成本，对于查询准入控制至关重要。图 8 展示了 G1 和 G2 上假阴性和假阳性造成的 CPU 时间与成本浪费。假阴性会浪费预配置集群上的 CPU 资源，而假阳性则会在无服务器集群上产生开销。我们采用直观的定价模型：预配置集群每 CPU 小时收费 0.3 美元，无服务器集群每 CPU 小时收费 0.5 美元；此外，MO 查询可享有 2,000 CPU 小时的免费配额。在我们的实验中，所有实例生成的 OOM 查询每天浪费的 CPU 时间略低于 2,000 CPU 小时。因此，假阳性样本使用的 CPU 小时数低于这一阈值，不会向用户收取额外费用。

部署前，G1 和 G2 上的中央处理器浪费时间总计分别为 1491.1 小时和 1187.1 小时。基于 SQL 的 XGB 将其略微降至 1384.1 小时和 1181.6 小时；Plan-based-XGB 的降幅更大，但在无服务器集群上仍分别消耗 1217.2 小时和 730.7 小时，限制了其实用性。判别规则、LS-Rule 和向量相似度会将许多非 MO 查询错误分类，导致无服务器中央处理器开销超过实际 MO 查询的中央处理器使用量，因此不适合部署，这使得

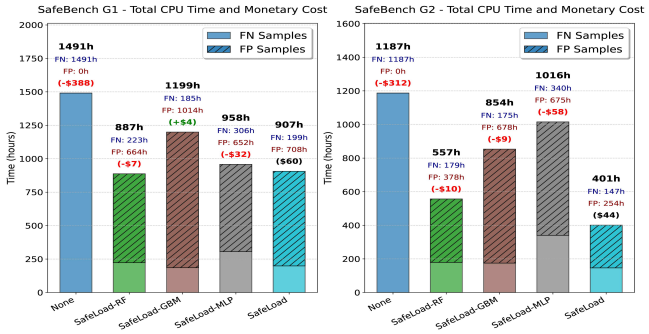


图 9：不同 SafeLoad 变体的浪费的CPU时间和货币成本。各变体相对于 SafeLoad 的货币成本差额（G1 为 60 美元，G2 为 44 美元）。

表 5：对 SafeLoad 各组件的消融实验。

SafeBench G1	TN	TP	FN	假I			F1
SafeLoad	50,853,048	2,203	305	512	**0.999984**	**0.8114**	0.8784	**0.**
不使用判别规则过滤	50,851,847	2,005	503	1,713	0.999956	0.5393	0.7994	
不含全局和局部模型	50,849,660	912,417	3,900		0.999876	0.0228	0.0363	
不含局部模型	50,852,415	2,244	264	1,145	0.999972	0.6621	0.8947	
无误预测校正	50,853,048	2,089	419	512	0.999982	0.8032	0.8329	
无自调优配额管理	50,851,702	2,286	222	1,858	0.999959	0.5516	**0.9115**	0
SafeBench G2	**TN**	**TP**	**FN**	**假I**				**F1**
SafeLoad	48,818,865	2,076	255	481	**0.999985**	**0.8119**	0.8906	**0.8494**
无判别规则过滤	48,818,187	1,960	371	1,159	0.999969	0.6284	0.8408	
不使用全局和局部模型	48,816,940	612,270	2,406		0.999904	0.0247	0.0262	
不使用本地模型	48,818,003	2,123	208	1,343	0.999968	0.6125	**0.9108**	0
无误预测校正	48,818,865	1,903	428	481	0.999981	0.7982	0.8164	
无自调节配额管理	48,818,559	2,097	234	787	0.999979	0.7271	0.8996	

它们不适合部署。相比之下，SafeLoad 能够高效识别 MO 查询，将预配置集群上的中央处理器浪费时间分别降至 198.9 小时和 146.7 小时。图 8 显示，部署 SafeLoad 后，客户在 G1 和 G2 上分别节省了 388 美元和 312 美元。

6.3 消融实验（EQ2）

6.3.1 关键组件的影响。我们进行了消融实验，以评估 SafeLoad 中各组件的影响。表 5 总结了实验结果。消融实验表明，移除**全局和局部模型**会使 F1 分数从 0.8 以上骤降至接近零（G1 上为 0.0280，G2 上为 0.0254），这说明它们在 MO 查询检测中不可或缺，是性能提升的最大来源。**判别规则过滤**模块也带来了显著收益；移除该模块后，F1 值大幅降至 0.6441（G1）和 0.7193（G2），同时计算开销也有所增加。**自调节配额管理**模块主要提升精确率；在 G1 上移除该模块后，精确率从 0.8114 急剧降至 0.5516，表明该模块对减少假阳性至关重要。仅使用**全局模型**（不含局部模型）时，F1 值（G1 上降至 0.7611）和精确率（G1 上降至 0.6621）均有所下降，说明局部模型有助于捕捉集群特定模式。最后，**误预测纠正**模块带来的提升最小，但仍有意义；省略该模块后，F1 值在 G1 上降至 0.8178。

总体而言，结果表明各个组件在提升检测性能方面都发挥着关键作用，验证了我们框架设计选择的合理性。

表 6：采用其他学习模型的 SafeLoad 变体。

SafeBench G1	TN	真正例	假I			F1
SafeLoad-RF	50,853,037	2 ,133	375	523	0.999982	0.8031	0.8505
SafeLoad-GBM	50,852,497	2 ,227	281	1 ,063	0.999974	0.6769	**0.8880**	0.7682
SafeLoad-MLP	50,852,761	1 ,975	533	799	0.999974	0.7120	0.7875	0.7478
SafeLoad	50,853,048	2 ,203	305	512	**0.999984**	**0.8114**	0.8784	**0.**
SafeBench G2	**TN**	**真正例**	**假I**			**F1**
SafeLoad-RF	48,818,828	1 ,987	344	518	0.999982	0.7932	0.8524	
SafeLoad-GBM	48,818,486	2 ,057	274	860	0.999977	0.7052	0.8825	
SafeLoad-MLP	48,818,855	1 ,704	627	491	0.999977	0.7763	0.7310	0.7530
SafeLoad	48,818,865	2 ,076	255	481	**0.999985**	**0.8119**	**0.8906**	

6.3.2 机器学习模型变体。 从**模型预测性能**来看，如表 6 所示，所有采用 SafeLoad架构的模型变体的 F1 分数均高于 0.7，显著优于最佳基线方法（其 F1 分数为 0.5）。这凸显了 SafeLoad架构在实现准确预测方面的优势。在这些变体中，SafeLoad中的 XGBoost模型取得了最高的 F1 分数和精确率，彰显了基于树的模型在预测内存过载方面的有效性。

从**减少 CPU 时间和货币成本**的角度来看，如图 9 所示，SafeLoad-RF 和 SafeLoad-GBM 在减少 CPU 时间浪费方面的表现与 SafeLoad 相近。然而，它们在时间开销和预测精确率方面有所不足。例如，SafeLoad-RF 每条 SQL 查询的平均预测开销为 4.2 毫秒，而在使用学习模型的场景中，SafeLoad 的开销明显更低，仅为 0.48 毫秒。另一方面，SafeLoad-GBM 的平均预测精确率比 SafeLoad 低 0.1，导致假阳性更多，并增加了用户开销。此外，图 9 表明，在 SafeLoad 的所有变体中，采用 XGBoost 模型的配置总体收益最高。

6.4 时间开销（EQ3）

本节分析所有评估方法的时间开销，并将其分为在线预测开销和离线构建开销。

6.4.1 在线预测开销。基于 SQL 的 XGB 和 Plan-based-XGB 的预测开销分别为每个查询 0.42ms 和 0.44ms，约占平均查询延迟的 0.005%。MART 和 LearnedWMP 的预测开销分别为每个查询 0.11ms 和 0.6ms。启发式规则基线、Discriminative-Rule 和 LS-Rule 的开销可忽略不计，分别为 401ns 和 386ns。向量相似度的开销较低，为 0.03ms。SafeLoad 使用其规则过滤模块处理 97.8% 的查询，开销为 401ns。其余 2.2% 的查询需要模型预测和向量搜索，开销为 0.48ms（~0.006% 的查询延迟）。因此，SafeLoad 的在线预测开销极低，适合在生产环境中使用。

6.4.2 离线构建开销。基于 SQL 的 XGB 和 Plan-based-XGB 的训练开销分别为 485 秒和 735 秒。MART 和 LearnedWMP 的训练开销分别为 390 秒和 15 小时。LS-Rule 没有离线开销，而 Discriminative-Rule 需要 34 分钟进行规则验证。向量相似度方法的 MO 查询较少，向量索引构建开销极低，仅为 0.73 毫秒。SafeLoad 的离线流程包括 XGBoost

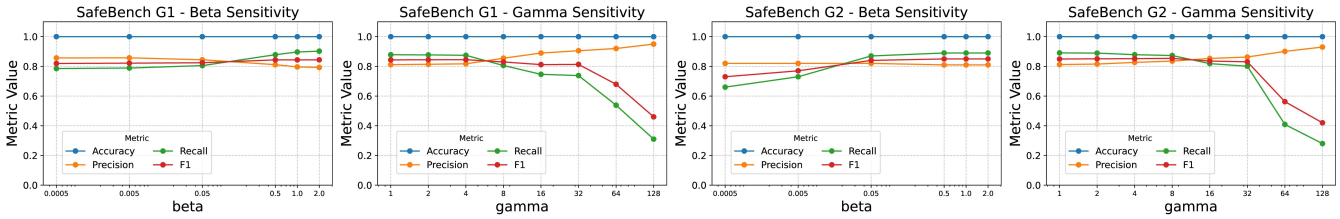


图10：四项分类指标的敏感性分析：β在 G1 上、γ在 G1 上、β在 G2 上，以及γ在 G2 上（从左到右）。

训练、向量索引构建和规则生成。未被规则过滤筛除的样本中，只有 2.2% 用于模型训练。在 G1 数据集上，向量索引构建和模型训练耗时 11 秒，离线总开销为 34 分钟，主要来自规则生成。值得注意的是，为 G1 和 G2 生成的规则完全相同，这表明可以复用前一天的规则，无需重新计算。对于生产环境中的每日模型更新而言，这一开销是合理的。

6.5 参数敏感性分析（EQ4）

通过敏感性分析，我们确定了 SafeLoad 自调优配额管理的最优参数：$\gamma = 1$ 和 $\beta = 0.5$。图10展示了 G1 和 G2 的分析结果。

对于 β（其中 $\gamma = 1$），测试值为 0.0005、0.005、0.05、0.5、1 和 2。当 β 介于 0.5 和 2 之间时，性能保持稳定；在 $\beta = 0.5$时，F1值和精确率达到最优。超过 $\beta = 2$后，配额成本会按照公式2降至最低并维持在该水平。

对于 γ（其中 $\beta = 0.5$），测试值为 1、2、4、8、16、32、64 和 128。当 γ 介于 1 和 4 之间时，性能保持稳定；但增大 γ 会减少预测出的正样本数量，从而降低 F1值和精确率。这是因为 γ 会调节模型置信度，并对低置信度正样本施加更高的配额成本。

总体而言，SafeLoad 的分类指标在广泛的参数取值范围内保持稳定，其中 $\gamma = 1$ 和 $\beta = 0.5$ 实现了性能与资源效率之间的最佳平衡。

7 相关工作

查询准入控制。现有的查询准入机制主要关注查询延迟 [19, 22, 27], SLA 合规性 [26], 资源隔离 [24], 以及吞吐量优化 [19]。Q-Cop [22]提出了“查询混合”的概念，通过线性回归预测执行时间，并拒绝可能超出时间限制的查询。Auto-WLM [19] 通过预测查询长度动态调整并发限制，而 Bouncer [27]则估算百分位响应时间以满足 SLO目标。ActiveSLA [26]通过评估满足 SLA 的经济价值来优化查询准入，而 DAC [24] 则通过事务分类和动态阈值调整实现策略感知资源隔离。然而，这些方法忽略了内存过载这一关键问题。SafeLoad 在执行前高效识别 MO 查询，弥补了这一不足；与现有侧重延迟和吞吐量的方法相比，这是一种新颖的方案。

内存消耗预测。以往研究虽取得进展，但仍存在关键不足。Ganapathi 等 [10] 使用核典型相关分析，根据查询计划预测性能指标，主要关注延迟，对内存的关注有限使用量。Li 等 [13] 提出了 MART，该方法将执行计划特征与提升回归树相结合，以估算中央处理器和 I/O 时间；这一查询资源估计框架可调整用于内存预测。Twitter [21] 和 CASA [29] 利用 SQL 文本特征和 XGBoost 预测内存需求，但未纳入查询计划和集群级洞察，而这些对于在动态环境中确保准确率至关重要 [11]。LearnedWMP [18]使用多层感知机和 XGBoost 对查询模板和内存直方图建模，重点关注并发查询。这些方法难以检测内存过载，往往忽视动态集群状况和峰值需求场景。SafeLoad 提出了一种全面且新颖的框架，整合了全局和局部学习模型、判别规则生成以及自调节配额管理，从而能够主动识别 MO 查询。

工业系统中的内存诊断。Snowflake 已采用启发式规则 [20] 来检测复杂查询，但无法捕捉复杂的模式。Oracle AWR [3], 、MySQL Performance Schema [15]、BigQuery [4], 和 AWS Redshift [1] 侧重于执行后分析或运行时分析，并依赖动态内存分配。相比之下，SafeLoad 会在执行前主动预测内存过载，将查询特征与集群指标相结合，因此有别于现有的执行后分析方法。

8 结论

本文介绍了 SafeLoad，这是一种创新的准入控制框架，能够主动防止云数据仓库发生内存过载。SafeLoad 将判别规则生成、学习模型和自调节配额管理机制相结合，为内存超载查询检测提供了全面的解决方案。实验结果表明，精确率提升了 66%，CPU 时间浪费最多减少 8.09×，凸显了 SafeLoad 在增强云环境中的资源管理和用户体验方面的巨大潜力。

致谢

本研究由 CCF-阿里云数据库研究基金（项目编号：CCF-Aliyun2024008）、浙江省先锋研发计划（编号：2024C01021）、国家自然科学基金委员会项目（编号：62402420）、国家自然科学基金委员会项目（编号：U24A20254）以及浙江省“科技创新领军人才计划”（编号：2023R5214）资助。Bo Tang 部分获得中国国家自然科学基金委员会（NSFC 编号：62422206）资助。我们感谢匿名审稿人和 AnalyticDB 团队提供的宝贵反馈与建议。

,

参考文献

[1] Nikos Armenatzoglou, Sanuj Basu, Naga Bhanoori, Mengchu Cai, Naresh Chain- ani, Kiran Chinta, Venkatraman Govindaraju, Todd J. Green, Monish Gupta, Sebastian Hillig, Eric Hotinger, Yan Leshinksy, Jintian Liang, Michael McCreedy, Fabian Nagel, Ippokratis Pandis, Panos Parchas, Rahul Pathak, Orestis Poly- chroniou, Foyzur Rahman, Gaurav Saxena, Gokul Soundararajan, Sriram Sub- ramanian 和 Doug Terry。2022。Amazon Redshift 的革新。载于*2022* 年数据管理国际会议论文集 （美国宾夕法尼亚州费城） （*SIGMOD '22*）。美国计算机协会，纽约，美国，2205–2217。[2] SafeLoad 作者。2025。SafeBench 基准测试。https://github.com/SafeLoad-project/SafeBench。访问日期：2025-12-08。[3] AWS。2025。Oracle AWR。https://docs.oracle.com/en-us/iaas/performance-hub/ doc/awr-report-ui.html。访问日期：2025-12-08。[4] BigQuery。2025。BigQuery 查询执行图。https://cloud.google.com/blog/products/data-analytics/understanding-the-bigquery-query-execution- graph。访问日期：2025-12-08。[5] Chris J. C. Burges、Krysta M. Svore、Qiang Wu 和 Jianfeng Gao。2008。排序、提升与模型自适应。技术报告 MSR-TR-2008-109。18 页。[6] 阿里云。2025。无服务器数据仓库中的内存过载查询智能路由。https://www.alibabacloud.com/blog/intelligent-routing-of-memory---overloading-queries-in-serverless-data-warehouses_602559。访问日期：2025 12 08 - -。[7] 阿里云。2025。介绍 AnalyticDB 上的智能查询路由，实现始终在线分析——绝不崩溃！https://www.alibabacloud.com/blog/introducing-intelligent-query-routing-on-analyticdb-for-always-online-analytics---no-crash_602558。访问日期：2025-12-08。[8] Benoit Dageville、Thierry Cruanes、Marcin Zukowski、Vadim Antonov、Artin Avanes、Jon Bock、Jonathan Claybaugh、Daniel Engovatov、Martin Hentschel、Jiansheng Huang、Allison W. Lee、Ashish Motivala、Abdul Q. Munir、Steven Pelley、Peter Povinec、Greg Rahn、Spyridon Triantafyllis 和 Philipp Unterbrunner。2016。Snowflake 弹性数据仓库。载于*2016* 年数据管理国际会议论文集 （美国加利福尼亚州旧金山） （*SIGMOD '16*）。美国计算机协会，纽约，美国，215–226。[9] Matthijs Douze、Alexandr Guzhva、Chengqi Deng、Jeff Johnson、Gergely Szilvasy、Pierre-Emmanuel Mazaré、Maria Lomeli、Lucas Hosseini 和 Hervé Jégou。2024。faiss 库。*arXiv* 预印本 *arXiv:2401.08281* （2024）。[10] Archana Ganapathi、Harumi A. Kuno、Umeshwar Dayal、Janet L. Wiener、Armando Fox、Michael I. Jordan 和 David A. Patterson。2009。利用机器学习预测查询的多项指标：助力实现更优决策。载于第 *25* 届国际数据工程会议论文集（*ICDE 2009*），*2009* 年 *3* 月 *29* 日至 *4* 月 *2* 日，中国上海。592–603。[11] Roman Heinrich、Manisha Luthra、Johannes Wehrstein、Harald Kornmayer 和 Carsten Binnig。2025。学习型成本模型究竟有多好？来自查询优化任务的见解。arXiv:2502 01229 [ ]。cs.DB Zhaoxuan Ji、Zhongle Xie、[12] Yuncheng Wu 和 Meihui Zhang。2024。LBSC：面向云数据库的成本感知缓存框架。载于第 *40* 届电气电子工程师学会国际数据工程会议（*ICDE 2024*），*2024* 年 *16* 月 *13* 日4911 4924 10 1109 60146 2024荷兰乌得勒支，*5* 月 -。电气电子工程师学会，–。https://doi.org/。/ICDE。。00373 [13] Jiexing Li、Arnd Christian König、Vivek Narasayya 和 Surajit Chaudhuri。2012。利用统计技术稳健估算 SQL 查询的资源消耗。*VLDB* 学报 5，11 （ 2012 年 7 月），1555–1566。[14] Scott M Lundberg 和 Su-In Lee。2017。解释模型预测的统一方法。神经信息处理系统进展 30（2017）。[15] MySQL。2025。MySQL Performance Schema。https://dev.mysql.com/doc/refman/ 8.4/en/performance-schema.html。访问日期：2025-12-08。[16] Emanuel Parzen。1962。概率密度函数及众数的估计。数学统计年刊 33, 3（1962），1065–1076。[17] Yuchen Peng、Zhongle Xie、Ke Chen、Gang Chen 和 Lidan Shou。2025。迈向分析型数据库中自动且高效的预测查询处理。载于*2025* 年电气电子工程师学会第 *41* 届国际数据工程会议（*ICDE*）。2253–2266。https://doi.org/ 10 1109 65448 2025 00171 . /ICDE . . [18] Shaikh Quader, Andres Jaramillo, Sumona Mukhopadhyay, Ghadeer Abuoda, Cal- isto Zuzarte, David Kalmuk, Marin Litoiu, and Manos Papagelis. 2024. Learned-WMP：使用查询模板分布进行工作负载内存预测。*CoRR* abs/2401.12103 (2024)。arXiv:2401.12103[19] Gaurav Saxena, Mohammad Rahman, Naresh Chainani, Chunbin Lin, George Caragea, Fahim Chowdhury, Ryan Marcus, Tim Kraska, Ippokratis Pandis, and Balakrishnan (Murali) Narayanaswamy. 2023. Auto-WLM：机器学习增强的工作负载管理，应用于 Amazon Redshift。载于*2023* 年数据管理国际会议（*SIGMOD/PODS 2023*）伴随会议，*2023* 年 *6* 月 *18—23* 日，美国华盛顿州西雅图，Sudipto Das, Ippokratis Pandis, K. Selçuk Candan, and Sihem Amer-Yahia（编）。225–237。[20] Snowflake. 2025. 识别可能受益于查询加速服务的查询和仓库。https://docs.snowflake.com/en/user-guide/query-acceleration-service。访问日期：2025-12-08。[21] Chunxu Tang, Beinan Wang, Zhenxiao Luo, Huijun Wu, Shajan Dasan, Maosong Fu, Yao Li, Mainak Ghosh, Ruchin Kabra, Nikhil Kantibhai Navadiya, Da Cheng, Fred Dai, Vrushali Channapattan, and Prachi Mishra. 2021. 在 Twitter 上预测 SQL 查询成本。载于*IEEE* 云工程国际会议（*IC2E 2021*），*2021* 年 *10* 月 *4—8* 日，美国加利福尼亚州旧金山。154–160。[22] Sean Tozer, Tim Brecht, and Ashraf Aboulnaga. 2010. Q-Cop：通过避免不良查询组合，尽量减少高负载下的客户端超时。载于第 *26* 届国际数据工程会议（*ICDE 2010*）论文集，*2010* 年 *3* 月 *1—6* 日，美国加利福尼亚州长滩，Feifei Li, Mirella M. Moro, Shahram Ghandeharizadeh, Jayant R. Haritsa, Gerhard Weikum, Michael J. Carey, Fabio Casati, Edward Y. Chang, Ioana Manolescu, Sharad Mehrotra, Umeshwar Dayal, and Vassilis J. Tsotras（编）。397–408。[23] Laurens van der Maaten and Geoffrey Hinton. 2008. 使用 t-SNE 可视化数据。机器学习研究期刊 9（2008 年 11 月），2579–2605。[24] Donghui Wang, Peng Cai, Weining Qian, and Aoying Zhou. 2021. 混合 OLTP 工作负载下共享一切数据库的判别式准入控制。载于第 *37* 届 *IEEE* 国际数据工程会议（*ICDE 2021*），*2021* 年 *4* 月 *19—22* 日，希腊于尼亚。780–791。[25] Ziniu Wu, Ryan Marcus, Zhengchun Liu, Parimarjan Negi, Vikram Nathan, Pascal Pfeil, Gaurav Saxena, Mohammad Rahman, Balakrishnan Narayanaswamy, and Tim Kraska. 2024. Stage：Amazon Redshift 中的查询执行时间预测。载于*2024* 年数据管理国际会议伴随会议（智利 Santiago AA）（*SIGMOD '24*）。美国计算机协会，纽约，纽约州，美国，280–294。[26] PengCheng Xiong, Yun Chi, Shenghuo Zhu, Jun'ichi Tatemura, Calton Pu, and Hakan Hacigümüs. 2011. ActiveSLA：面向数据库即服务提供商的面向利润的准入控制框架。载于与 *SOSP 2011* 联合举办的 *ACM* 云计算研讨会，*2011* 年 *10* 月 *28* 日，*11* 月 *26* 日 15*SOCC '*，葡萄牙卡斯凯什，*10* 月—Jeffrey S. Chase and Amr El Abbadi（编）。. [27] Hao Xu and Juan A. Colmenares. 2024. Bouncer：面向低延迟在线数据系统、具有响应时间目标的准入控制。载于*2024* 年数据管理国际会议伴随会议（*SIGMOD/PODS 2024*），*2024* 年 *6* 月 *9—15* 日，智利 *Santiago AA*，Pablo Barceló, Nayat Sánchez-Pi, Alexandra Meliou, and S. Sudarshan（编）。400–413。[28] Lijie Xu, Wensheng Dou, Feng Zhu, Chushu Gao, Jie Liu, Hua Zhong, and Jun Wei. 2015. 经验报告：分布式数据并行应用中的内存不足错误特征研究。载于*2015* 年 *IEEE* 第 *26* 届软件可靠性工程国际研讨会（*ISSRE*）。518–529。[29] Tim Zeyl, Harshwin Venugopal, Calvin Sun, and Paul Larson. 2024. CASA：面向查询处理引擎的基于分类的调整槽位准入控制。载于第 *40* 届国际数据工程会议（*ICDE 2024*）研讨会，*2024* 年 *5* 月 *13—16* 日，荷兰乌得勒支。278–284。[30] Chaoqun Zhan, Maomeng Su, Chuangxian Wei, Xiaoqiang Peng, Liang Lin, Sheng Wang, Zhe Chen, Feifei Li, Yue Pan, Fang Zheng, and Chengliang Chai. 2019. AnalyticDB：阿里云上的实时 OLAP 数据库系统。*VLDB* 学报 12, 12（2019 年 8 月），2059–2070。