arXiv:2602.02999v1 [cs.DB] 3 Feb 2026

ResQ：真实性能感知查询生成

Zhengle Wang Yanfei Zhang Chunwei Liu†普渡大学 Databend Cloud 普渡大学 wangzhengle@cau.edu.cn zhangyanfei@databend.com chunwei@purdue.edu

摘要

数据库研究与开发高度依赖现实的用户工作负载，用于基准测试、实例优化、迁移测试和数据库调优。然而，由于严格的隐私法规，获取真实世界的 SQL 查询一直极具挑战性。尽管云数据库供应商已开始向研究社区发布匿名化性能跟踪数据，但这些数据通常只提供高层次的执行统计信息，而不包含原始查询文本或数据，这对于需要实际执行的场景而言并不足够。现有工具无法捕捉细粒度的性能模式，也无法生成能够高保真、高效率地复现这些公开跟踪数据的可运行工作负载。为弥补这一不足，我们提出 ResQ——一种细粒度工作负载合成系统，旨在生成可执行的 SQL 工作负载，忠实匹配生产环境跟踪数据中每个查询的执行目标和算子分布。ResQ 构建执行感知查询图，通过贝叶斯优化驱动的谓词搜索将其实例化为 SQL，并在精确查询级别和参数化模板级别上通过复用显式建模工作负载重复。为确保实际应用中的可扩展性，ResQ 将搜索空间界定与轻量级局部成本模型相结合，以加速优化。在公开云端跟踪数据（Snowset、Redset）和新发布的工业跟踪数据（Bendset）上进行的实验表明，ResQ 显著优于最先进的基线方法，令牌节省率达到 96.71%，运行时间减少 86.97%，同时 CPU 时间的最大 Q- 误差降低 14.8×，扫描字节数的最大 Q- 误差降低 997.7×，且算子组合高度匹配。

PVLDB 引用格式：
ResQ：真实性能感知查询生成。PVLDB，18(1)：XXX-XXX，2025。doi:XX.XX/XXX.XX

PVLDB 工件可用性：
源代码、数据和/或其他工件已在 https://github.com/WASSER2545/ResQ 上提供。

1 引言

现代云数据库系统高度依赖现实工作负载来评估性能、指导系统调优并支持实例优化（例如容量规划和自动扩缩容）[5, 13, 15, 16,19, 29]。然而，由于隐私、安全和数据治理约束，获取具有代表性的生产环境 SQL 工作负载正变得越来越困难。传统的 TPC 基准测试（例如 TPC-H/TPC-DS）仍被广泛使用，但对于现代云部署而言已日渐过时：其固定模板和简化假设无法反映生产环境的关键特征，例如倾斜且长尾的查询分布、高重复率以及多样化的算子组合 [23]。

为更好地支持实证数据库研究，近期有多家云数据库供应商发布了匿名化的生产跟踪数据，包括 Snowset 和 Redset [23, 24]。这些跟踪数据包含从实际部署中收集的查询级执行日志，其中原始 SQL 文本和用户数据均已删除。相反，每条跟踪记录会披露部分执行目标（例如 CPU 时间、扫描字节数及其他资源信号）以及有限的结构信号（例如算子组合）。尽管这些跟踪数据很有价值，但它们无法执行，无法直接用于受控环境中的端到端基准测试、回归测试或调优。

这促使人们日益需要现实的性能感知工作负载合成：生成可执行的 SQL 工作负载，使其查询执行行为与生产环境中观察到的真实用户查询高度一致。缺乏忠实的执行特征时，合成工作负载可能导致基准测试结果产生误导、配置不理想，以及性能诊断不准确 [18]。

粗粒度工作负载合成。近期研究探索了粗粒度的工作负载合成，即按时间窗口而非单个查询聚合并建模执行统计数据。如图 1 所示，PBench [31]将工作负载跟踪划分为连续的时间窗口，并在每个窗口内合成查询，以匹配聚合后的执行指标。PBench 通过组合可复用组件来构建工作负载，这些组件来自大语言模型生成的查询和预先分析的基准测试片段，用以近似窗口级执行统计数据。相比之下，SQLBarber [14]将查询成本离散化为一组固定的成本范围，并通过贝叶斯优化扩展查询模板，以覆盖目标成本分布。为支持模板生成，SQLBarber 还采用基于大语言模型的框架来生成查询模板。

然而，粗粒度合成不可避免地以牺牲保真度为代价来实现聚合。由于它仅在窗口或分布层面匹配目标，因此无法准确捕捉单个查询的性能特征。因此，它提供的逐查询控制有限，也无法再现云工作负载中普遍存在的强重复模式 [23]。

细粒度工作负载合成。为克服粗粒度合成的局限性，本文提出了一种基于现实执行统计数据的细粒度工作负载合成方法。其目标是生成合成工作负载，使每个单独的查询 在执行指标（例如 CPU 时间、扫描字节数）和结构特征（例如算子组合）方面，都与

本作品采用知识共享署名-非商业性使用-禁止演绎 4.0 国际许可协议授权。请访问 https://creativecommons.org/licenses/by-nc-nd/4.0/ 查看本许可协议副本。如需进行本许可协议未涵盖的任何使用，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。Proceedings of the VLDB Endowment，第 18 卷，第 1 期，ISSN 2150-8097。
doi:XX.XX/XXX.XX

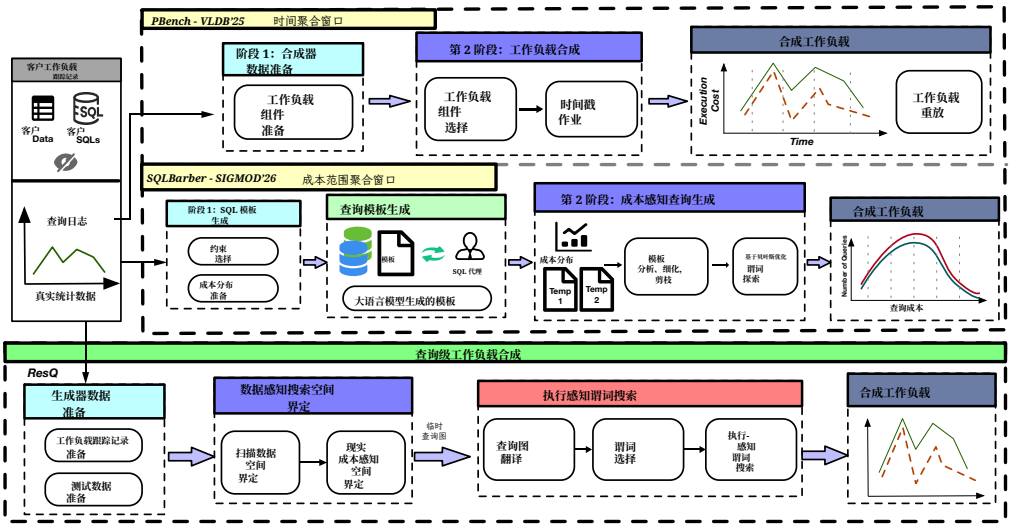


图 1：工作负载合成。

realcloud 工作负载跟踪中观察到的情况高度吻合。由于出于隐私原因，现实工作负载跟踪不会公开原始 SQL 查询或底层用户数据，因此查询级工作负载合成需要在信息不完整的情况下构建高保真可执行查询。具体而言，本研究聚焦于通过以下方式生成合成查询：（1）重建满足从工作负载跟踪中推导出的逐查询目标和约束的、具备性能感知能力的查询图；（2）将这些查询图实例化为可在公开基准数据库（例如 TPC-H 和 TPC-DS）上执行的 SQL 查询。

下游任务。细粒度、性能感知的合成能够在不访问用户 SQL 或数据的情况下，为多项下游任务提供实用的工作负载：(1) 基准测试。通过生成与真实执行特征相匹配的查询级工作负载，该方法能够对云数据库系统进行比传统基于模板的基准测试更忠实的基准测试。(2) 实例优化。具有目标执行特征的合成查询可用于在受控且现实的工作负载下评估和调优实例配置，例如资源分配和扩缩容策略 [4, 27]。(3) 迁移测试。在版本升级或数据迁移期间，性能感知的查询生成使从业者能够在不访问敏感用户查询或数据的情况下重放具有代表性的用户工作负载，从而帮助检测性能退化和兼容性问题 [7, 17]。(4) 数据库调优。通过系统地改变执行目标和数据相关约束，生成的查询支持对查询优化器和执行引擎进行压力测试，从而促进细粒度诊断和性能调优 [15, 28]。

关键挑战。要有效解决具备现实执行统计数据的查询级工作负载合成问题，面临三项关键技术挑战。

(C1) 信息不完整时的细粒度可控性。查询级合成需要生成单条查询，使其执行指标（例如 CPU 时间）和算子组合与真实工作负载跟踪记录中观察到的情况高度匹配。这要求在信息不完整时，对查询结构和执行行为进行细粒度控制因为原始 SQL 查询和数据库均不可用。

（*C2*）捕捉工作负载重复性。生产环境工作负载通常包含大量重复内容：相同或模板相似的查询会使用不同参数反复出现。朴素的合成方法往往会使查询过度多样化，从而丢失这种结构和时间局部性。

（*C3*）大规模效率。查询级合成需要在查询结构、谓词和执行计划构成的庞大而复杂的搜索空间中进行搜索。穷举探索或细粒度优化的成本可能高得难以承受，而过度简化的启发式方法则可能导致执行统计数据的保真度较差。实用的合成方法必须在准确性和生成成本之间取得平衡。

我们ResQ 方法。 为应对这些挑战，我们提出 ResQ（**现实合成查询**）。本研究通过明确揭示并控制驱动查询执行行为的关键因素，解决具备现实执行统计数据的细粒度工作负载合成问题。ResQ 不将查询生成视为纯粹的黑箱过程，而是采用结构化且关注性能的生成流程，直接针对上述挑战进行设计。

首先，为了实现（C1），ResQ 采用**两阶段查询图生成**框架。在第一阶段，ResQ 构建抽象查询图，定义连接图、算子类型和高层数据流，而不限定具体的 SQL 语法。在第二阶段，通过贝叶斯优化（BO）谓词搜索细化这些查询图，并在性能约束下将其实例化为可执行查询。其次，为应对挑战（C2），ResQ 部署了**查询池**，支持在两个层级检索和复用先前生成的查询：（i）查询哈希层级，用于捕捉完全相同的结构重复；（ii）参数化查询哈希层级，通过规范化谓词值和参数来抽象查询模板。最后，为应对挑战（C3），ResQ 将**搜索空间界定**与**局部预测模型**相结合，加速贝叶斯优化（BO）过程。搜索空间界定算法用于剪枝那些经证明无法达到目标执行指标的查询图，

而轻量级局部预测模型则可快速估算成本，引导 BO 搜索有希望的候选项，而无需执行完整的查询。

与现有方法的区别。以往方法要么以粗粒度合成工作负载 [14, 31]，要么通过大语言模型提示或逐词元学习生成可控性有限的 SQL [21, 30]。这些方法通常间接约束执行行为（例如，通过提示或奖励），因此难以系统地使每个查询匹配多个执行目标，也难以复现重复模式。相比之下，ResQ 明确将结构构建、可行性界定和谓词实例化分离，从而为轨迹驱动的查询级合成提供细粒度控制和实用的执行效率。

贡献。我们的贡献总结如下。

（1）我们形式化定义了从匿名化云端工作负载跟踪记录中生成具有现实执行统计数据的查询这一问题（第 2 节）。

（2）我们提出了 ResQ，这是一种兼具准确性与效率的查询级合成框架，结合了重复感知复用（精确哈希和参数化哈希查询池）、性能感知搜索空间界定以及轻量级局部成本模型，以加速基于贝叶斯优化的实例化（第 3 至 5 节）。

（3）我们与一家工业界云数据库厂商合作，发布了一个新的细粒度工作负载跟踪记录 Bendset，涵盖多种真实世界用例（第 6 节）。

（4）我们实现了 ResQ，并使用真实的云端工作负载跟踪记录对其进行评估，包括 Snowset、Redset 和 Bendset（第 7 节）。

2 问题定义

定义 2.1（**测试环境D**）。我们将测试环境定义为元组 $\mathbf{D} = (W, \Gamma, \Delta)$，其中 W 表示待测试的目标云数据库系统（包括其优化器和执行引擎），Γ 是系统级配置参数的集合（例如，并行度、内存限制），Δ 是由 *DBA* 预先加载的测试数据集。环境 D提供了一个可控的沙箱，用于执行合成查询，而无需访问任何用户 *SQL* 文本或用户数据。

定义 2.2（**参数化查询模板**）。参数化查询模板是一个 *SQL* 查询骨架 τ，其中参数向量 $\boldsymbol{\theta} \in \Theta$ 出现在谓词（例如范围边界、等值、IN列表）和查询选项中。使用 $\boldsymbol{\theta}$ 实例化 τ 后，可得到一个可执行 *SQL* 查询，记作 $q = \tau(\boldsymbol{\theta})$。

定义 2.3（**约束函数与约束**）。设 Q 表示测试环境 D中可执行 *SQL* 查询的空间。约束函数是任意函数 $f : Q \times \mathbf{D} \to \mathbb{R}$，其值可通过在 D 中运行 q 和/或检查生成的执行计划/性能画像（例如运行时间、扫描字节数、算子数量）获得。关于 f 的约束可指定为范围 $[l, u]$，要求 $f(q, \mathbf{D}) \in [l, u]$；也可指定为带容差 ϵ 的目标值 t，要求 $|f(q, \mathbf{D}) - t| \le \epsilon$。

定义 2.4（**定向查询生成**）。给定一个测试环境 D、一个参数化查询模板 τ 及其参数域 Θ，以及一组约束 $\mathcal{R} = \{(f_i, \rho_i)\}_{i=1}^{n}$（其中每个 ρ_i 都是一个范围或带容差的目标值），定向查询生成问题是寻找参数 $\boldsymbol{\theta} \in \Theta$，使得实例化查询 $q = \tau(\boldsymbol{\theta})$ 在 D上满足 $\mathcal{R}$ 中的所有约束。

定义 *2.5*（结构感知约束 *C*）。设 $H = \{h_k : Q \times D \to R_{\ge 0} \mid k = 1, \ldots, K\}$ 是一组依赖结构的函数，其中每个 $h_k(q, D)$ 都刻画 q 在 D 中或其执行计划中的某种结构属性（例如访问的表数量、连接/聚合/排序运算符数量、查询类别）。目标结构特征为 $C = (c_k)_{k=1}^{K}$，并带有容差 ϵ^H。若 $\forall k \in \{1, \ldots, K\}$ 中的 q 满足 $|h_k(q, D) - c_k| \le \epsilon_k^H$ 且 $\epsilon_k^H = 0$，则该查询满足结构感知约束（精确匹配是其特例）。

示例 2.1(**结构感知约束**)。 在 *Snowset*、*Redset* 和 *Bendset* 等跟踪记录中，原始 *SQL* 不可用，但记录可能会指定结构特征，例如“访问的表 = 3”、“#连接 = 2”和“#排序 = 1”。给定一个针对代理数据集 Δ 的测试环境D，结构感知约束要求合成查询在一定容差范围内匹配这些结构信号。

定义 *2.6*（执行性能目标 *Y*）。设 $G = \{g_m : Q \times D \to R_{\ge 0} \mid m = 1, \ldots, M\}$ 为一组可测量的执行性能函数，其中每个 $g_m(q, D)$ 都是在 D 中观测到的物理执行指标（例如 *CPU* 时间、*I/O* 时间、扫描字节数、峰值内存、运行时间）。目标执行特征是一个带有容差 $\boldsymbol{\epsilon}^G$ 的向量 $Y = (y_m)_{m=1}^{M}$。如果 $\forall m, |g_m(q, D) - y_m| \le \epsilon_m^G$，则查询 q 匹配 Y。

示例 2.2（**执行目标**）。考虑与特定云端数据库版本和配置相对应的测试环境 D 。跟踪记录可能会报告以下目标：*CPU* 时间 = 2.4 秒、峰值内存 = 512 *MB*，以及扫描字节数 = 800 *MB*。这些值构成目标执行特征 Y。

定义 2.7（**真实性能感知查询生成（RPQG）**）。给定 *(i)* 测试环境 D、*(ii)* 目标结构特征 C、*(iii)* 目标执行特征 Y，以及 *(iv)* 参数化模板的搜索空间 $\mathcal{T}$，真实性能感知查询生成问题是合成一个可执行查询$q^* = \tau(\boldsymbol{\theta})$，其中某个 $\tau \in \mathcal{T}$和 $\boldsymbol{\theta} \in \Theta_\tau$满足以下条件：*(a)* q^*满足结构感知约束 C，且 *(b)* q^* 在D中执行时，尽可能贴合执行目标 Y 。

当无法精确满足目标时，*RPQG* 会针对执行指标最小化差异目标，例如：

$$\min_{q=\tau(\boldsymbol{\theta})} \sum_{m=1}^{M} w_m \cdot \frac{|g_m(q, \mathbf{D}) - y_m|}{\max(y_m, \eta)} \quad s.t. \quad |h_k(q, \mathbf{D}) - c_k| \le \epsilon_k^H, \ \forall k,$$

其中 w_m 为用户指定权重，$\eta > 0$ 用于避免除以 *0*。

3 ResQ 系统架构

本节介绍 ResQ 的架构，用于解决第 2 节定义的真实性能感知查询生成问题。给定匿名工作负载轨迹和保护隐私的测试环境 $\mathbf{D} = (W, \Gamma, \Delta)$，ResQ 会合成可执行 SQL 查询，使其在容差范围内匹配轨迹提供的结构感知约束C 和执行性能目标Y，且无需访问原始用户 SQL 或数据。

如图2所示，ResQ以一系列匿名查询级轨迹记录作为输入，每条记录指定（i）一组执行目标（例如 CPU 时间和扫描字节数）以及（ii）一组

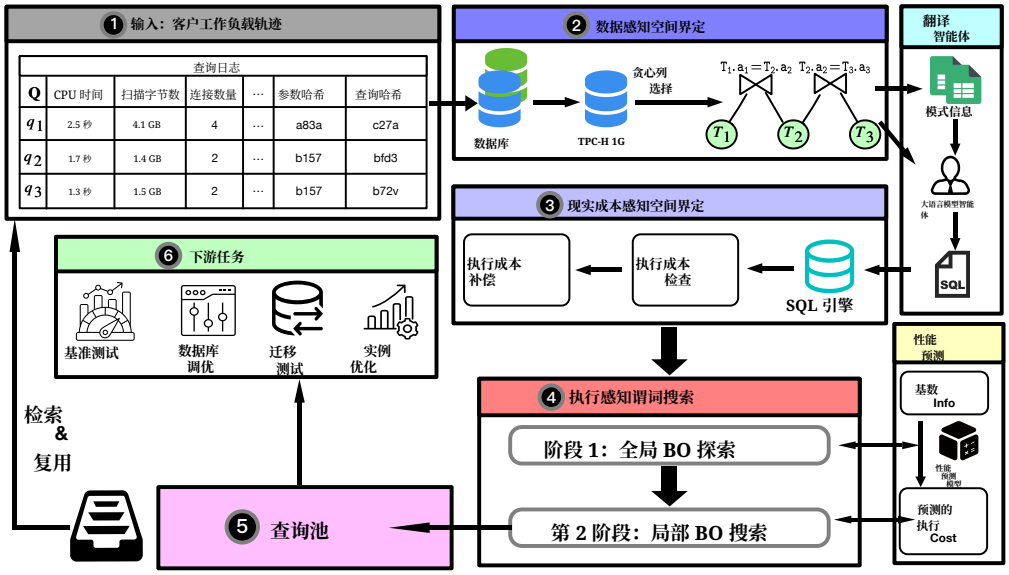


图2：ResQ 系统架构。

结构感知约束（例如连接/聚合/排序组合）（第 2 节）。对于每条记录，ResQ 都会在测试环境D 中合成一个可执行SQL查询，使其满足结构约束，并尽可能贴合执行目标。

为使其能够大规模实现，ResQ 结合了：(i) 生成器数据准备；(ii) 两阶段生成流水线，先限定数据与性能搜索空间，再实例化谓词（第 4 节）；(iii) 查询图转换器，将面向执行的表示转换为可执行的 SQL；(iv) 局部性能模型，以减少搜索过程中耗费高昂的查询执行（第 5 节）；以及 (v) 查询池，通过支持检索和复用来恢复重复项并提高生成效率。

3.1 生成器数据准备

此阶段包含两个关键流程：工作负载跟踪准备和测试数据准备。

3.1.1 工作负载跟踪准备。 云数据库供应商发布的热门跟踪记录提供了低敏感度的性能信号，这些信号可以在省略原始 SQL 文本和用户数据的情况下共享用于研究。遵循第 2 节的定义，我们将每条跟踪记录视为包含：(a) 执行目标向量Y 和 (b) 结构感知分析C，以及其时间戳。不同于窗口级生成（例如 PBench [31]）将查询级记录聚合到粗粒度时间窗口中，ResQ 保留查询级记录，以支持细粒度的目标匹配和重复模式恢复。

执行目标（Y）。 在当前实现中，我们从 Bendset、Snowset [24], 和 Redset [23] 中提取每个查询的*CPU* 时间 和扫描字节数，将其作为主要执行目标。

结构感知约束 (C)。 我们使用 连接、聚合和排序 作为关键算子类型，以此约束算子构成。Bendset 和 Redset 报告明确的算子数量，而最早的公开跟踪数据集 Snowset 仅提供粗粒度的算子级分析时间；因此，对于 Snowset，我们将算子出现情况视为二元指示变量（存在/不存在）。关于 Bendset 的更多详情见第 6 节。

3.1.2 测试数据准备。 为了在不访问用户数据的情况下生成可执行查询，我们使用隐私保护代理数据集 Δ，通过实例化测试环境 $D = (\mathcal{W}, \Delta)$（第 2 节）来完成这项工作。我们部署了多个具有不同规模因子的 TPC-H 和 TPC-DS 实例，作为候选数据集。数据集规模主要影响扫描字节数，而数据分布（例如倾斜）会影响连接扇出和中间结果大小，进而影响 CPU 时间。

为了支持后续的生成阶段，我们在模式和数据分布层面对每个代理数据集 Δ 进行预处理。对于模式元数据，我们记录表/列名、数据类型和连接键。对于数据分布元数据，我们收集行数、每列数据大小和基本值统计信息（例如最小值/最大值）。这些元数据随后用于构建可行的查询结构、限制扫描字节数，并在谓词搜索期间定义谓词域。

3.2 两阶段查询图生成

如图2所示，ResQ 使用两阶段流水线生成每个查询（详见第4节）。核心思路是，首先利用主要由结构 和数据访问决定的约束界定搜索空间，然后通过谓词实例化 细化剩余的自由度，以匹配执行目标。

阶段 I：空间界定与基础图构建。ResQ 首先构建一个满足结构感知约束C 且在代理数据集 Δ 上可行的基础查询图。此阶段执行数据感知空间界定，以剪枝那些无法达到Y中扫描字节目标的候选项，并生成一小组可供细化的可行基础图。

阶段 II：代价感知细化与谓词实例化。从有界基础图出发，ResQ 将查询细化至 Y 中的目标 CPU 时间。它结合现实的代价感知检查（并在必要时进行轻量级代价补偿）与一种性能感知的谓词搜索流程，以高效调整选择率。

3.3 查询图翻译器

为弥合算子级规划与可执行 SQL 之间的差距，ResQ 部署了专用的查询图转换器。该组件不参与代价感知决策，而是处理已完全指定的查询图，并负责生成符合W方言的可运行 SQL 查询。

翻译器以确定性方式将关系算子（例如扫描、连接、聚合、排序、标量求值）映射到对应的 SQL 形式，同时保留算子依赖关系。基于大语言模型的模块仅用于解决难以用确定性规则表达的语法及方言相关细节，例如子查询嵌套、别名管理和表达式放置。

3.4 性能预测

性能感知生成需要频繁获取关于执行指标的反馈，但在W中运行每个候选查询可能代价过高。因此，ResQ 引入轻量级局部性能模型，根据查询图和可用统计信息估计关键指标——尤其是 CPU 时间。这些估计用于指导成本感知界定，并减少谓词搜索期间的高代价执行次数。详细的模型设计见第 5 节。

3.5 查询池

真实世界的工作负载表现出强烈的重复性。为恢复重复模式并降低生成延迟，ResQ 维护一个查询池，支持两个层级的检索与复用：(i)查询哈希复用，用于精确的结构重复；以及 (ii) 参数化哈希复用，用于模板级重复，即复用结构但谓词值不同。当跟踪记录与池中的条目匹配时，ResQ 复用相应的结构/模板，并将计算集中于轻量级细化和谓词实例化，从而在保持工作负载局部性的同时提高效率。

在实践中，出于隐私约束，某些工作负载跟踪（例如 Snowset）不会公开显式的查询哈希。在这种情况下，Res Q 会将目标执行分析Y 与结构约束C 相结合，构造一个定长特征向量作为代理签名 。该向量用作索引，用于从候选池中检索与给定跟踪记录的执行行为和结构最匹配的候选查询或查询图。如果找到合适的匹配项，ResQ 会直接复用检索到的查询（或查询图）；否则，该记录会被转交给生成流程，以合成一个新查询，然后将其插入候选池，供日后复用。

4 两阶段查询图生成

本节介绍 ResQ 的核心生成流程。给定一条指定结构感知约束 C 和执行目标Y （第 2 节）的跟踪记录，我们的目标是合成一个可执行查询，使其（i）满足C；并且（ii）在测试环境D = (W Γ， Δ)中执行时，尽可能贴合Y。

主要难点在于，Y 包含执行层级的信号（例如 CPU 时间和扫描字节数），而直接在 SQL 语法空间中搜索既成本高昂，又难以控制。ResQ 通过先生成面向执行的表示来解决这一问题，只有在结构和成本可行性大致确定后，才将其转换为 SQL。

4.1 从现实查询生成到查询图生成

4.1.1 为什么需要查询图。 CPU 时间和扫描字节数等执行目标主要由算子组合（例如连接、聚合、排序的数量）、访问的数据（数据量和宽度）以及谓词选择率驱动，而不是由表面的 SQL 语法驱动。因此，ResQ 不直接生成 SQL 文本，而是引入查询图 G这一抽象，用于捕获算子依赖关系和数据流，同时将特定方言的语法及物理实现细节留到后续阶段处理。

4.1.2 查询图。

定义 4.1（**查询图** G）。查询图G = （V E）是一个有向无环图（*DAG*），其节点V 表示关系算子（例如扫描、连接、聚合、排序、标量求值），其边 E 表示算子之间的数据依赖关系。每个节点都带有 *SQL* 实例化所需的属性，例如所引用的表、列和算子参数。

4.1.3 问题重述。 第 2 节中的 RPQG 目标是在可执行查询上定义的，通过结构函数 $h_k(q, \mathrm{D})$ 和执行函数 $g_m(q, \mathrm{D})$ 来刻画。ResQ 使用 G 作为中间表示，并将生成拆分为：(i) 构造满足结构特征 C 且能够达到数据访问目标（例如扫描字节数）的可行G；(ii) 将 G 实例化为参数化 SQL 模板，并调优谓词以匹配计算目标。

从概念上讲，这将 RPQG 化简为针对查询图的约束搜索，以及在每个图内进行的参数搜索。

定义 4.2（**多目标查询图生 成**）。 设 $\mathbb{G}$ 为在 Δ 的模式上所有有效查询图构成的空间。给定目标结构特征 C 和目标执行特征 Y，多目标查询图生成问题是寻找一个查询图 $\mathrm{G}^* \in \mathbb{G}$，使得 G^* 满足 C，并且存在一种实例化方式，使其在 D 上的执行与 Y匹配。

命题 4.1（**查询到图的归约（非正式）**）。对于本文所考虑的执行目标（例如扫描字节数和 *CPU* 时间），可行性和优化主要取决于：（i）算子组合及其依赖关系；（ii）访问数据量和谓词选择率。这些因素都在查询图中得到了明确表示。因此，*ResQ* 首先在查询图空间中进行搜索，以控制这些因素，然后将选定的图翻译为 *SQL*，以完成最终实例化。

4.2 阶段 I：数据感知的搜索空间界定

阶段 I 使用代理数据集 Δ，根据数据访问可行性界定搜索空间。具体而言，我们以扫描字节数为目标，因为它会显著受到所扫描表和列的选择影响，并且可以在谓词调优前进行界定。

*4.2.1*扫描字节空间界定。 令 R ={R_1、.....、 R_n} 表示 Δ 中的表。给定连接的结构约束（例如，C 中的连接数），候选查询图必须选择一组基本扫描

算子，从而确定一组访问的表 $R_\prime \subseteq R$。为确保查询有效性，$R_\prime$ 必须在模式连接图中构成一个连通子图（例如，该图由主键–外键关系诱导而成）。

对于每个选定的表 $R_i \in R_\prime$，设 $A(i)$ 为其列。每一列 $A(i)_j$ 都关联一个估算扫描权重 $w(i)_j$（字节数），该权重根据第 3.1 节收集的存储统计信息得出。给定目标扫描字节数（来自 Y），目标是为每个表选取一个列子集，使估算扫描字节数尽可能接近目标，同时确保包含连接键列，以保证可行性。

这构成了一个受约束的多子集选择问题，我们将其用作界定步骤：在进行任何开销高昂的谓词优化之前，提前剪除无法达到扫描目标的图。

4.2.2 贪心数据感知列选择。 由于列数增加会带来指数级复杂度，对于现实模式而言，精确的动态规划方案并不切实际。因此，ResQ 采用一种可扩展至迭代生成的贪心算法（算法 1）。该算法分为三个阶段：（i）连接图落地，用于选择一组相连的表和强制连接键列；（ii）逐表初始化，以避免退化扫描（每个表至少贡献一列）；（iii）全局贪心补全，持续添加更多列，直至达到扫描目标。

复杂度与复用。 对于固定的连接图，贪心列选择的运行时间为 $O(N \log N)$，其中 N 是所选表中候选列的数量。尽管在最坏情况下，连接图的枚举呈指数级增长，但对于结构化基准模式（例如 TPC-H/TPC-DS），可行的连通子图数量较少。此外，ResQ 会缓存中间结果——经过落地的连接图及其受扫描界定的列选择——并在具有相似结构特征的跟踪记录之间复用这些结果，从而大幅减少重复工作。

4.3 阶段I：现实成本感知的空间界定

在界定扫描字节数之后，ResQ 进一步利用计算相关的可行性细化候选查询图，重点关注 CPU 时间。

步骤I：强制实施结构感知约束。 从受扫描界定的基础图开始，ResQ 注入额外算子以满足目标结构特征 C（例如，在需要时添加排序和聚合算子），从而生成完整的候选图。

步骤 II：执行成本可行性检查。我们对候选图进行初步 CPU 时间估算，并将其与 Y 中的目标 CPU 时间进行比较。只要条件允许，这项检查就会使用局部性能模型（第 5 节）。这种设计在准确性与成本之间实现了实用的权衡，避免对大多数候选项进行完整执行。

步骤 III：执行成本补偿。如果估计的 CPU 时间低于目标，ResQ 会引入轻量级标量求值算子，对选定列应用计算表达式，以增加 CPU 成本，同时保持整体查询结构不变。候选标量运算从支持的函数库（算术运算、字符串、日期函数）中选取。其增量 CPU 成本由局部模型（第 5.2 节）估计，ResQ 贪心地选择运算类型及应用次数，以使查询

算法 1： 贪心数据感知列选择

输入： 目标扫描字节数 $y$scan；模式级连接图 Gschema；候选表集合 R，每张表对应的列集合 $\{A(i)\}$

输出： 列选择 $\{S(i)\}_{i=1}^{K}$

//步骤 I：连接图落地

选择子集 $R_\prime \subseteq R$，使得 $|R_\prime| = K$ 且 $R_\prime$ 构成一个 Gschema 中的连通子图；

从 $R_\prime$ 推导出表组 $\{A(i)\}_{i=1}^{K}$；

识别强制连接键列 $\{M(i)\}_{i=1}^{K}$ 来自连接谓词；

初始化 $S(i) \leftarrow M(i)$ 对所有 i；

$S \leftarrow \sum_i \sum_{A_j^{(i)} \in M(i)} w_j^{(i)}$；

//步骤 II：分组初始化

设置平均目标 $\bar{y} \leftarrow y$扫描$/K$；

对于 $i \leftarrow 1$ 到 K **执行**

  如果 $\mathcal{S}(i) = \emptyset$ **则**

    选择 $A_{j^*}^{(i)} = \arg\min_{A_j^{(i)} \in A(i)} \left| w_j^{(i)} - \bar{y} \right|$；

    $\mathcal{S}^{(i)} \leftarrow \{A(i)_{j^*}\}$；

    $S \leftarrow S + w_{j^*}^{(i)}$；

//步骤 III：全局贪心补全

构建候选集 $\mathcal{R}_{\text{cand}} = \{A_j^{(i)} \mid A_j^{(i)} \in \mathcal{A}^{(i)} \setminus \mathcal{S}^{(i)}\}$；

按 w_j 升序对 Rcand 排序$^{(i)}$；

foreach $A_j^{(i)} \in \mathcal{R}_{cand}$ **do**

  if $S \geq y$扫描 **then**

    中断

  $\mathcal{S}^{(i)} \leftarrow S(i) \cup \{A(i)_j\}$；

  $S \leftarrow S + w_j^{(i)}$；

返回 $\{S(i)\}_{i=1}^{K}$；

图进入一个可行范围。这确保后续谓词搜索有足够的“空间”，可以朝最终目标进行细粒度调整。

4.4 阶段 II：性能感知谓词搜索

阶段 II 实例化谓词以匹配执行目标，重点关注 CPU 时间。在给定固定查询结构（查询图）和一组候选谓词列的情况下，ResQ 搜索谓词参数，使 CPU 时间尽可能接近目标，同时尽量减少高代价执行。

4.4.1 谓词列选择。 允许所有可能的谓词列会导致搜索空间维度过高且不规则，从而显著降低贝叶斯优化（BO）的效果。因此，ResQ 将调优限制在少量对执行敏感的 列上。直观而言，CPU 时间受中间结果大小的显著影响；因此，(i) 能够较早进行过滤，且 (ii) 具有宽广、平滑的选择率范围的谓词，是最有用的调优参数。

对于多连接查询，ResQ 不要求每个表贡献相同数量的谓词维度。相反，对于大型表或取值范围更广的表，会分配更多维度，因为它们通常主导 CPU时间方差。我们将谓词形式限制为单侧范围谓词（例如，$R.a \le p$），这样谓词取值与中间结果大小之间就会呈现弱单调关系，从而提高优化稳定性。

4.4.2 两阶段谓词调优。 谓词调优被表述为在由选定谓词列构成的有界定义域上进行的黑盒优化。ResQ 采用两阶段贝叶斯优化策略：阶段 1 进行全局探索，以找到有希望的区域；第 2 阶段则围绕目标附近的代表性种子进行局部细化。

4.4.3 性能感知评分函数。 对每个候选谓词配置都直接执行查询，代价高昂。因此，ResQ 使用混合评分策略：首先利用局部模型预测 CPU 时间，只有当预测的 CPU 时间落在目标值附近的容差窗口内时，才执行查询。这样既能在目标区域附近获得准确反馈，又能大幅减少远离目标时的执行次数。

4.4.4 阶段 ***1***：全局贝叶斯优化探索。 在阶段 1 中，贝叶斯优化通过少量随机评估进行初始化，随后开展引导探索。所有已评估的谓词配置及其得分都会被记录。阶段 1 结束时，ResQ 会选出两个种子：x^+（最接近目标但不低于目标）和 x^-（最接近目标但低于目标），它们从两侧界定了目标区域。

4.4.5 第 ***2*** 阶段：局部贝叶斯优化细化。 在第 2 阶段，ResQ 通过围绕 x^+ 和 x^- 缩小各谓词维度，构造更小的搜索空间，并可针对非常大的值域应用值分桶。随后在每个缩减后的空间中运行贝叶斯优化，重点进行利用。从两端进行细化，可以增强对预测误差和不规则代价曲面的鲁棒性。

算法 2 总结了两阶段调优过程。

算法 2： 两阶段 BO 谓词调优**输入：** SQL 模板 q^{temp}，谓词列 C，列类型 T，目标 CPU 时间 y_{cpu}^{tar}输出：优化后的 SQL 谓词 // 辅助函数：谓词评分 1 函数 ScorePredicate(x):2 $q' \leftarrow$ BuildSql(x, q^{temp})$y_{cpu}^{pred} \leftarrow$ localmodelpredict(q'); 3 if $y_{cpu}^{pred} \in [y_{cpu}^{tar} \cdot a, y_{cpu}^{tar} \cdot b]$ then 4 $y_{cpu}^{obs} \leftarrow$ RunQuery(q') 5 else6 $y_{cpu}^{obs} \leftarrow y_{cpu}^{pred}$ 7 $score \leftarrow -$均方误差 $\left(y_{cpu}^{obs}, y_{cpu}^{tar}\right)$; 8 返回 score; // 阶段 1：全局 BO 探索 9 $S_1 \leftarrow$ BuildGlobalSearchSpace(C, T); 10 在 S1 上初始化 BO; 11 for $i = 1$ 到 $n_{\text{rand}} + n$次调用 do 12 $x \leftarrow$ BO.ask(); 13 评分 $\leftarrow$ ScorePredicate(x); 14 BO.tell(x,score); 15 记录 x, score; 16 为第 2 阶段选择种子：x^+ 和 x^-; // 第 2 阶段：局部 BO 搜索 17 **对**每个种子 $x^* \in \{x^+, x^-\}$ **执行**18 $S_2 \leftarrow$ ShrinkSearchSpace(S_1, x^*);19 $S_2 \leftarrow$ ApplyBucketing(S_2);20 在 S_2 上初始化贝叶斯优化；21 **对** $i = 1$ 到 $n_{\text{随机}} + n_{\text{次调用}}$ **执行** 22 $x \leftarrow$ BO.ask();23 得分 $\leftarrow$ ScorePredicate(x);24 BO.tell(x, 得分); 25 **返回**$\{x_{best}^+, x_{best}^-\}$

5 本地模型支持

ResQ 中的约束感知生成需要频繁获取执行目标 Y （例如 CPU 时间）的反馈，这些目标在第 2 节中定义。虽然在测试环境 D $=(W, \Gamma, \Delta)$ 中执行每个候选查询可以获得准确的测量结果，但在搜索循环中这样做的成本极高。因此，ResQ 使用局部性能模型，根据候选查询图的算子级特征及其中间统计量，对关键执行指标（尤其是 CPU 时间）进行近似预测。这些模型之所以称为“局部”，是因为它们是在固定环境 D （即特定的引擎W、配置 Γ 和数据集 Δ）下训练和使用的，这显著提高了预测稳定性。

本节介绍我们的 CPU 时间建模与预测框架。我们以物理算子为粒度对执行过程建模，并汇总各算子的预测结果，以估算查询级 CPU 时间。我们首先介绍连接算子，因为它们通常是分析型工作负载中 CPU 消耗的主要来源，而且仅用简单的解析公式很难准确建模。随后，我们将介绍其他算子的轻量级模型。

5.1 连接算子的建模

在不同的连接算法中，CPU 消耗主要由两个常见阶段决定：(i) 匹配阶段，该阶段处理连接键并识别匹配的元组对；(ii) 物化阶段，该阶段构造输出元组并物化选定列。因此，我们将连接算子的 CPU 时间建模为

$$CPU_{\text{join}} = CPU_{\text{match}}^{(\cdot)} + CPU_{\text{material}}, \quad (1)$$

其中，$CPU^{(\cdot)}$ match 表示算法特定的匹配逻辑，而CPU material 表示各种连接算法共用的输出构造。

5.1.1 **哈希连接。** 现代向量化 OLAP 引擎通常将哈希连接实现为匹配阶段，后接一个可选的物化阶段。在匹配过程中，算子生成轻量级行引用（例如行索引），而非物化后的元组。在物化过程中，根据这些引用获取/解码选定列。

匹配阶段。 匹配阶段会计算连接键的哈希值并探测哈希表。其 CPU 成本主要由输入基数和每个元组的哈希计算开销决定：

$$CPU_{\text{match}} = \alpha \cdot |R_B| \cdot cost_{\text{hash}}(K_B) + \beta \cdot |R_P| \cdot cost_{\text{hash}}(K_P) + \delta \cdot \#\text{tasks}, \tag{2}$$

其中，$|R_B|$ 和 $|R_P|$ 是构建/探测基数，K_B 和 K_P 是连接键，而 #tasks 则用于捕获执行环境下的编排开销（例如并行分片）。

物化阶段。 物化阶段根据行引用获取并解码输出列，其开销取决于输出基数和投影宽度：

$$CPU_{\text{material}} \approx \epsilon \cdot |R_{\text{out}}| \cdot cost_{\text{fetch}}(S_{\text{out}}), \tag{3}$$

其中，$|R\text{out}|$ 是输出基数，$S\text{out}$ 是输出模式，而$cost\text{fetch}(\cdot)$则表示每行的获取/解码开销，该开销取决于列数和数据类型。

哈希连接 CPU 时间。 结合这两个阶段，

$$CPU_{\text{HJ}} = CPU_{\text{match}} + CPU_{\text{material}}. \tag{4}$$

这种分解解释了为什么 CPU 时间在仅计数连接（物化开销极小）和投影宽元组的连接（以物化开销为主）之间可能存在显著差异。

5.1.2 **其他连接算法。** 对于嵌套循环连接和归并连接，我们采用公式 (1) 中相同的两阶段抽象。它们的匹配阶段在实现上有所不同（例如，嵌套探测与顺序比较），但其物化成本均使用相同的宽度感知建模策略来刻画。由于哈希连接在我们的目标系统的分析型工作负载中占主导地位，我们将学习重点放在哈希连接上，并将其他算法视为次要情况。

5.2 对其他算子建模

除连接之外，现代 OLAP 引擎中的许多算子呈现出更简单的 CPU 时间模式，可以用轻量级参数化模型近似表示。

扫描。 扫描的 CPU 时间取决于输入元组的数量，以及对已访问列段进行解码和处理的成本：

$$CPU_{\text{scan}} = \alpha_s \cdot |R| \cdot \sum_{c \in \text{cols}} S_{\text{segment}}(c), \tag{5}$$

其中，$|R|$ 表示扫描元组的数量，cols 是已访问列的集合，而$S\text{segment}(c)$是平均每个元组的段处理开销（例如，解压缩/解码），该开销通过本地性能分析得出。

投影和标量函数。 对于投影和标量计算（例如字符串/日期函数），CPU 时间与输入基数近似呈线性关系：

$$CPU_{\text{func}} = \alpha_g \cdot |R| \cdot cost_{\text{func}}. \tag{6}$$

排序。 对于基于比较的排序，CPU 时间通常呈超线性趋势：

$$CPU_{\text{sort}} = \alpha_o \cdot |R| \cdot \log |R|. \tag{7}$$

这些抽象并非旨在成为完美的分析模型；相反，它们提供简洁的特征/先验，从而支持在D这一固定环境中实现高效学习和稳定外推。

5.3 CPU 时间预测

ResQ 通过聚合算子级预测来估计查询级 CPU 时间。给定包含算子节点 V 的候选查询图 G，我们计算：

$$\widehat{CPU}(\mathbf{G}) = \sum_{v \in \mathcal{V}} \widehat{CPU}(v), \tag{8}$$

其中 $\widehat{CPU}(v)$ 是算子 v 的预测CPU时间。

我们区分**复杂算子**（主要是连接）与**简单算子**（扫描、过滤算子、标量计算、排序）：前者的 CPU 时间取决于多个相互作用的因素（输入大小、选择率、输出宽度、并行度），后者的 CPU 时间主要由输入基数和算子类型决定。

5.3.1 **连接 *CPU* 时间预测.** 为预测连接的 CPU 时间，我们推导与第 5.1 节中匹配阶段和物化阶段相对应的特征。典型特征包括：构建/探测基数（$|R_B|, |R_P|$）、其比值（$|R_B|/|R_P|$）作为选择率的代理、用于刻画非线性效应的 $\log |R_B|$，以及诸如 $|R\text{输出}| \cdot \bar{w}\text{输出}$ 之类的宽度感知项，用以近似物化成本。

我们不固定单一的参数形式，而是训练一个梯度提升决策树（GBDT）回归器 [10]，以捕捉连接执行中常见的非线性交互和阈值效应（例如倾斜、突发的缓存效应以及内存局部性变化）。训练数据通过在D中执行多种分析查询来收集，并从引擎分析信息中提取算子级运行时统计信息；目标变量是每个连接算子的实测 CPU 时间。

获取基数。 在推理时，连接预测需要基数相关特征。这些特征可以通过以下方式获得：(i) 优化器估计值，(ii) 学习型基数模型[16, 8, 11, 12, 25, 26]，或 (iii) 轻量级辅助执行。在本研究中，我们采用一种明确的工程权衡：发起一个辅助 COUNT(*) 查询，该查询与原查询共享相同的连接图和谓词，但避免物化完整结果，从而以适度开销获得准确的中间基数。这提升了 CPU 时间匹配的预测质量，同时避免了在整个搜索过程中运行完整候选项所需的高昂成本。

5.3.2 **其他算子的 *CPU* 时间预测.** 对于非连接算子，我们使用基于本地分析信息训练的轻量级参数化模型。对于过滤算子和标量求值等算子，我们将 CPU 时间建模为输入基数的线性函数：

$$CPU_{\text{op}} = \alpha \cdot |R_{\text{in}}| + \beta, \tag{9}$$

其系数从D中的剖析数据学习得到。对于排序，我们使用超线性模型，该模型纳入理论 $\Theta(n \log n)$项以及可选的宽度项：

$$CPU_{\text{sort}} = \alpha \cdot |R_{\text{in}}| \cdot \log |R_{\text{in}}| + \beta \cdot |R_{\text{in}}| \cdot \bar{w}_{\text{out}} + \gamma. \tag{10}$$

表 1：公有云工作负载跟踪数据对比

	Bendset	Redset	Snowset
来源	Databend Cloud	Amazon Redshift	Snowflake
查询	40M	~50M	~70M
周期	8 天	3 个月	14 天
特征	30	24	91
重复	85.0%	42.3%	–
算子信号	计数	数量	剖析时间

如何ResQ使用局部模型。在现实成本感知空间界定（第4.3节）期间，$\widehat{CPU}$(G)会尽早筛除不可行候选，并指导CPU代价补偿决策。在谓词搜索（第4.4节）期间，本地预测为BO探索提供低成本反馈，系统仅对预测CPU时间接近目标区域的候选项在D中执行查询，从而在保持最优值附近精度的同时减少高代价执行的次数。

6 Bendset 简介

Snowset [24] 和Redset [23]等现有公开跟踪记录，极大推动了基于轨迹的工作负载研究。然而，对于本文探讨的细粒度查询生成，它们仍不完全足够。具体而言，查询级合成需要：(i) 在不同系统间保持稳定的执行目标（例如 CPU 时间和扫描字节数）；(ii) 可在生成过程中加以约束的结构信号（例如算子组合）；以及 (iii) 重复相关信号，以复现生产工作负载中观察到的强局部性。为更好地满足这些需求并支持可复现研究，我们公开发布了一种新的跟踪记录：Bendset。

6.1 Bendset 概述。

Bendset 是 Databend Cloud [3] 发布的公开可用工作负载跟踪记录。它采集了 8 天内来自 100 个无服务器实例的真实生产查询。与 Snowset 和 Redset 中的通用分析型工作负载不同，Bendset 反映了一条游戏分析流水线：持续摄取数据，执行增量流式转换，并提供低延迟分析查询，为下游人工智能应用生成新鲜特征。完整跟踪记录包含 4000 万条查询，每条查询有 30 项特征，涵盖查询计时、I/O 统计数据、内存消耗和算子级指标。

6.2 与Snowset和Redset的比较。

表1总结了我们评估中所用三个公开跟踪记录的关键特征。Bendset 的重复率（85.0%）显著高于 Redset（42.3%），因此特别适合研究重复感知合成与重放。重要的是，Bendset 同时提供查询哈希 和参数化查询哈希，从而能够分别在精确查询和模板层面准确度量并复现工作负载重复。此外，Bendset 和 Redset 提供了显式的算子统计信息（例如连接/聚合/排序的计数），而 Snowset 主要报告粗粒度的算子级分析时间；这一差异对查询生成很重要，因为算子计数 可以直接作为结构感知约束加以强制执行。

7 实验

我们在真实的云数据库服务以及三个公开工作负载跟踪上评估 ResQ。除非另有说明，所有报告的数值均为端到端测量，包括生成、辅助执行和指标收集。

7.1 实验设置

平台与实现。我们在 Databend Cloud [3]，（一种类似 Snowflake 的云数据仓库服务）上运行所有实验。除非另有说明，每次运行均使用*Small* 仓库配置。我们从 system_history.query_history 收集查询执行指标。我们使用 skopt [9] 作为贝叶斯优化（BO）求解器。我们使用最新的 qwen-plus 模型作为查询图转换器（第 3.3 节）。**工作负载跟踪数据**。我们在三种公有云工作负载跟踪数据上进行评估：Bendset、Redset [23],和 Snowset [24]。所有跟踪数据仅包含 SELECT 查询，并且我们保留每条采样跟踪数据中的原始时间顺序，以反映现实的工作负载动态。*Bendset*：我们从不同的两天中随机采样两条时间上连续的 2 小时跟踪数据。每条跟踪数据包含按顺序执行的 200 条查询。这两条跟踪数据的重复率不同（分别为 40.5% 和 8.5%）。*Redset*：我们随机选取一条包含 104 条连续查询的 1 小时跟踪数据；其重复率为 42.3%。*Snowset*：我们随机提取一条包含 54 条查询、跨度约 30 分钟的跟踪数据。

初始工作负载。 我们使用具有多个规模因子的基准数据库构建测试数据集。具体而言，我们部署了规模为 1GB、2GB 和 5GB 的 TPC-H，以及规模为 1GB 和 2GB 的 TPC-DS。

评估指标。 *(1)* 性能准确度。对于Y 中的执行目标（CPU 时间和扫描字节数），我们报告*Q*-误差 ，并使用中位数、第90百分位和第99百分位数汇总结果，以反映典型表现和尾部表现。*(2)* 结构遵循度。 对于C 中的每项结构感知约束（例如连接/聚合/排序计数），我们报告每查询绝对误差，以及评估工作负载上的平均绝对误差。*(3)* 大语言模型成本。 我们报告基于大语言模型的组件消耗的提示令牌和补全令牌。*(4)* 生成延迟。 我们报告端到端实际运行时间，包括贝叶斯优化迭代、本地模型推理和辅助查询执行。

基线。 我们评估以下三种基线方法：**(1)SQLSmith-Q [22]:** 它通过随机扩展解析树来生成语法有效的 SQL 查询，主要用于发现错误。对于查询级评估，我们将其调整为*SQLSmith-Q*，针对每种目标和约束设置随机生成三个查询，以评估随机生成能否很好地匹配执行目标。 **(2) LLMGen：** LLMGen 改编自 PBench [31], 中基于大语言模型的增强模块，该模块利用专家知识引导多轮对话式改写，生成近似给定执行目标和结构约束的查询。遵循 PBench 的做法，我们针对每个输入目标进行三轮迭代精炼。 **(3) SQLBarber [14]：** SQLBarber [14] 通过将工作负载的逻辑成本分布划分为互不相交的成本范围，使用大语言模型生成 SQL 模板，并应用贝叶斯优化（BO）搜索谓词，使每个模板都能覆盖其指定范围，从而合成工作负载。为使 SQLBarber 适用于查询级生成，我们将其调整为针对每个目标查询生成一个模板，并应用基于贝叶斯优化的谓词搜索，为每个执行目标实例化该模板。

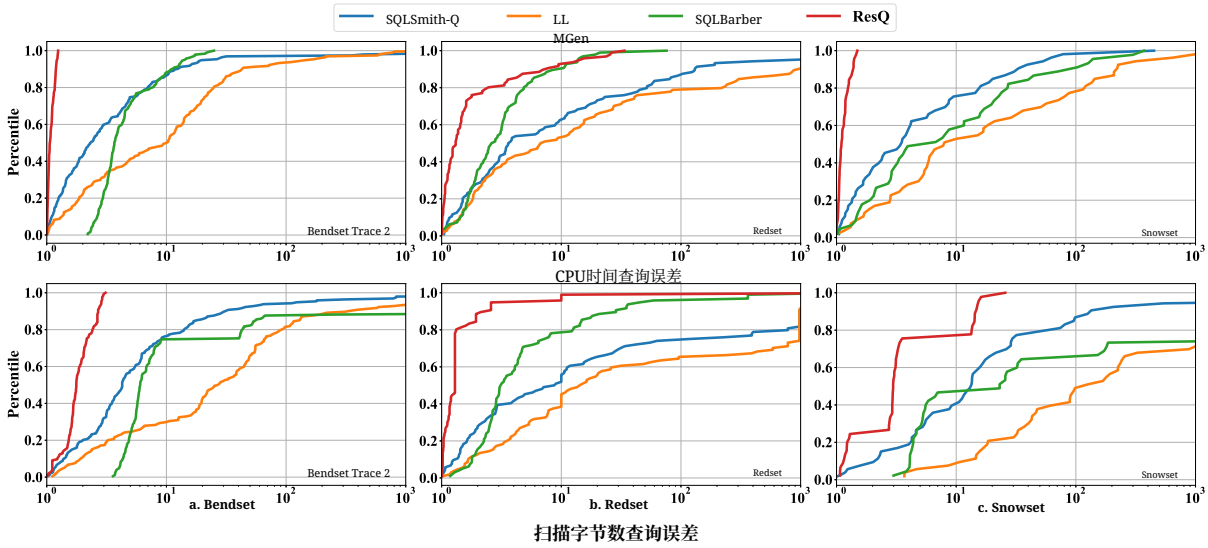


图 3：CPU 时间的 Q-误差累积分布函数 时间（上）和扫描字节数（下）：(a) Bendset Trace 2，(b) Redset 跟踪，(c) Snowset 跟踪 ce.

7.2 Bendset 上的评估

我们首先在两个 Bendset 跟踪记录上评估 ResQ，这些记录按第 7.1 节所述进行采样。我们报告每查询目标准确率、对结构感知约束的遵循情况（算子数量的平均绝对误差）、大语言模型令牌消耗以及端到端生成延迟。图 3 还使用累积分布函数图总结了误差分布。

性能准确度。表 2 和表 3 显示，在 CPU 时间和扫描字节数方面，ResQ 始终实现了最高的目标拟合准确率。在 Trace 1 中，我们的方法实现了 1.13 的 CPU 时间查询误差中位数，表现优于 SQLBarber（2.31）和 SQLSmith-Q（2.19），领先近 2× 倍。在 Trace 2 中，考察扫描字节数指标时，这种优势更加显著：ResQ 的 Q-误差中位数为 1.76，而 SQLBarber 和 LLMGen 则表现不佳，误差分别高达 6.00 和 24.17。这些结果表明，我们的方法能够细致地理解查询结构与物理资源消耗之间的关系，而基线方法生成的查询往往与预期的性能特征相差甚远。

尾部稳健性。 ResQ的尾部表现也显著更优。在 Trace 2 上，SQLSmith-Q 的第 99 百分位 CPU时间q误差高达 522.56，而 ResQ 仅为 1.21。同样，在 Trace 1 的扫描字节数方面，SQLSmith-Q（9247.89）和 SQLBarber（9.93）的第 99 百分位误差都出现了巨大偏差，而 ResQ 则稳定在 2.56。从中位数到第 99 百分位数几乎平坦的误差曲线，凸显了我们生成方法的稳健性和可预测性。与基于大语言模型的方法（如 LLMGen、SQLBarber）不同，后者受其固有的黑箱特性所限，缺乏对底层物理执行引擎和数据分布的原理性理解。因此，它们只能对查询性能进行粗粒度近似，在拟合细粒度的真实世界目标时稳健性较差。

结构约束遵循情况 为验证生成的查询在满足性能目标的同时是否保持逻辑一致性，我们报告了关键算子的平均绝对误差（MAE）。如表 2 和表 3 所示，ResQ 在连接、聚合和排序算子上的 MAE 接近于零（例如，两条跟踪记录中的连接和排序均为 0.00）。这些结果表明，我们的方法成功遵循了预定义的结构约束，确保合成查询不仅符合性能要求，而且在结构上也与任务规范保持一致。

大语言模型成本。 ResQ主要使用大语言模型执行查询图到SQL转换，因此与基于大语言模型的基线方法相比，词元消耗大幅降低。在 Trace 2 上，ResQ消耗了 18.97K 个提示词令牌，相较于 LLMGen（4487.67K）和 SQLBarber（2054.13K），减少了两个数量级以上。这种差异源于基线方法必须在提示中纳入大量少样本示例、复杂的表模式以及复杂的数据分布统计信息，以引导生成过程。相比之下，ResQ将大语言模型的职责简化为专注的翻译任务——将预先优化的算子树映射为 SQL 语法——从而大幅缩减输入上下文。此外，SQLBarber 和 LLMGen 依赖迭代式多轮对话进行查询重写和错误纠正，导致补全词元显著增加（例如，SQLBarber 为 1665.9K，而我们仅为 2.28K）；而 ResQ 则能通过单次处理实现高保真合成。这种“单次翻译”范式不仅消除了耗费高昂的精炼循环，还确保了大规模查询生成任务具备更强的可扩展性，在这类任务中，API成本和延迟都是关键约束。

效率。 为了评估 ResQ 在实际场景中的可扩展性，我们分析了端到端延迟以及资源消耗。表2和表2报告了不同跟踪记录中查询生成所需的总实际运行时间。我们的方法显著提升了效率，尤其是与涉及迭代精炼的方法（例如 SQLBarber 中的 BO 谓词搜索）相比。在 Trace 2 中，ResQ 仅用 111.73 分钟便完成生成任务，相较于 SQLBarber（1454.27 分钟）实现了 13× 倍加速。SQLSmith-Q 等较简单的方法由于采用基于模板的方式，在某些场景下可能具有相近的延迟，但无法达到性能目标。值得注意的是，ResQ 在 Trace 1 中仍比 SQLBarber 快 5.5× 倍，同时保持了更高的优化精度。显著的延迟差距源于生成范式的根本差异；具体而言，

表2：ResQ 在 Bendset Trace 1 上的评估。

方法	CPU时间查询误差			扫描字节数查询误差			Join	聚合	Sort	提示词令牌	输出令牌	延迟
	中位数	第90百分位	99th	中位数	90th	99th	MAE	MAE	MAE	(K)	(K)	（最小值）
SQLSmith-Q	2.19	58.52	425.59	3.74	137.51	9247.89	**0.00**	0.55	**0.00**	**0.00**	**0.00**	**202.07**
LLMGen	5.53	8.69	19.73	129.74	5367.18	8550.57	0.03	0.90	0.47	2244.19	265.52	249.25
SQLBarber	2.31	4.29	11.029	2.41	4.08	9.93	0.07	**0.02**	0.12	870.51	510.67	1560.85
ResQ	**1.13**	**1.42**	**1.66**	**1.14**	**1.69**	**2.56**	**0.00**	0.07	**0.00**	144.32	2.83	281.26

表3：ResQ 在 Bendset Trace 2 上的评估。

方法	CPU时间查询误差			扫描字节数查询误差			Join	聚合	Sort	提示词令牌	输出令牌	延迟
	中位数	第90百分位	99th	中位数	90th	99th	MAE	MAE	MAE	(K)	(K)	（最小值）
SQLSmith-Q	2.30	11.44	522.56	4.27	28.95	5319.80	**0.00**	0.27	**0.00**	**0.00**	**0.00**	175.51
LLMGen	8.80	32.92	148.41	24.17	133.41	1346.78	**0.00**	0.71	0.75	4487.67	383.65	313.51
SQLBarber	3.66	10.63	17.92	6.00	4628.11	5716.56	**0.00**	**0.00**	0.02	2054.13	1665.9	1454.27
ResQ	**1.05**	**1.18**	**1.21**	**1.76**	**2.64**	**2.87**	**0.00**	0.08	**0.00**	18.97	2.28	**111.73**

SQLBarber 缺乏搜索空间界定机制，并且在贝叶斯优化过程中利用实际执行反馈，因此产生了显著开销。

最后，ResQ 还能从重复查询中进一步受益。在 Trace 2（重复率为 40.5%）中，查询池复用能够将结构构建和谓词搜索的成本分摊到重复的查询/模板上，从而提高吞吐量；对于较长的跟踪记录，这一优势更加显著。

7.3 Redset 上的评估

为进一步评估 ResQ 的泛化能力，我们在图 3 和表 4 中报告了其在 Redset 跟踪上的表现。结果与先前跟踪记录中的观察高度一致，进一步验证了我们方法的鲁棒性。

目标准确性与尾部稳健性。 如表 4 所示，ResQ 持续展现出卓越的目标拟合准确率。值得注意的是，对于 扫描字节数，我们的方法实现了仅为 1.93 的第 99 百分位数 Q-误差，与 SQLSmith-Q（4666.01）和 LLMGen（985.87）形成鲜明对比。即使在波动性更大的 *CPU* 时间 指标上，ResQ 仍将第 99 百分位数控制在 12.61 的较低水平。这证实了我们的框架能够可靠地处理多样的数据分布和复杂的查询模式。在生成效率方面，ResQ 用时 126.31 分钟完成任务，较 SQLBarber 实现了 6.1× 倍加速。

结构约束遵循情况。 ResQ 严格遵循结构感知约束，在表 4 中实现了 0.00 的连接 MAE 和 0.02 的聚合 MAE。这表明，改进目标匹配并未以违反基于跟踪数据推导出的结构特征为代价。

效率。 ResQ 在126.31分钟内完成 Redset 跟踪，相较于 SQLBarber（773.49分钟）实现了 6.1× 的加速，同时取得了更高的目标准确性。这种效率源于 ResQ 的有界搜索，以及在贝叶斯优化过程中减少了对昂贵查询执行的依赖。

7.4 Snowset 上的评估

最后，我们在 Snowset 上评估 ResQ。与 Bendset 和 Redset 相比，Snowset 提供的结构信号较为粗略（算子特征是剖析时间，而非明确的算子数量），因此细粒度查询生成更具挑战性。

目标准确性与稳定性。 如表5所示，在结构信息减少的情况下，基线方法的稳定性显著下降，尤其是在扫描字节数方面：LLMGen 和 SQLBarber 的99%分位数q误差分别高达7769.17和4807.46。相比之下，ResQ 仍保持稳定，其 CPU时间q误差的99%分位数为1.25，扫描字节数q误差的99%分位数为14.24。这些结果凸显了 ResQ 搜索空间界定的优势：它依据数据访问可行性约束优化空间，而非仅仅依赖精确的算子数量约束。

无显式哈希的复用。与 Bendset 不同，Snowset 不提供查询哈希或参数化查询哈希字段。为了仍能受益于复用，ResQ 会利用可用的目标/约束信号（例如算子存在指示器和执行目标范围）进行检索，以识别具有相似分析结果的已生成查询/模板，并将其作为生成的热启动。即使无法获取精确的重复标识符，这种复用也有助于提升稳定性和效率。

效率。ResQ 在 Snowset 上实现了最低的端到端延迟（53.47 分钟），优于 SQLBarber（453.86 分钟），并且在提供高得多的目标准确率的同时，性能仍可与较轻量的基线方法相媲美。

7.5 消融实验

局部性能模型的影响。如图4所示，我们在 Bendset Trace 2 上评估局部模型的效率。纳入局部模型显著提高了生成效率。具体而言，ResQ 的端到端延迟为 111.73 分钟，而消融版本 ResQ-R 完成相同任务需要 143.51 分钟。这表明局部模型显著减少了贝叶斯优化过程中实际执行的次数。

效率提升源于局部模型在贝叶斯优化阶段充当高速代理的能力。通过快速估计候选查询结构的性能，局部模型使系统能够剪除搜索空间中缺乏前景的区域，而无需调用成本更高的评估

表 4：在 Redset 跟踪上的 ResQ 评估。

方法	CPU时间查询误差			扫描字节数查询误差			Join MAE	聚合 MAE	提示词令牌 (K)	输出令牌 (K)	延迟 （分钟）
	中位数	90th	99th	中位数	90th	99th					
SQLSmith-Q	3.59	59.11	1599.00	7.90	976.56	4666.01	0.00	0.43	**0.00**	**0.00**	132.42
LLMGen	5.89	279.41	1068.62	10.97	976.56	985.87	0.03	0.19	1026.92	84.13	128.85
SQLBarber	2.45	5.82	13.59	3.71	16.17	36.23	0.14	0.04	768.36	72.52	773.49
ResQ	**1.23**	**4.35**	**12.61**	**1.18**	**1.30**	**1.93**	**0.00**	**0.02**	123.86	2.97	**126.31**

表5：ResQ 在 Snowset 跟踪上的评估。

方法	CPU时间查询误差			扫描字节数查询误差			Join MAE	聚合 MAE	Sort MAE	提示词令牌 (K)	输出令牌 (K)	延迟 （分钟）
	中位数	第90百分位	99th	中位数	90th	99th						
SQLSmith-Q	2.42	25.48	57.63	13.19	98.27	24078.12	**0.00**	0.46	**0.00**	**0.00**	**0.00**	60.58
LLMGen	5.97	84.88	140.27	89.19	3865.10	7769.17	0.11	0.07	**0.00**	894.79	79.76	63.92
SQLBarber	3.12	17.64	27.66	25.72	172.33	4807.46	0.21	0.35	**0.00**	183.10	127.40	453.86
ResQ	**1.06**	**1.18**	**1.25**	**2.95**	**9.52**	**14.24**	**0.00**	**0.05**	**0.00**	9.17	0.87	**53.47**

步骤。因此，在交付时间方面，ResQ 的吞吐量远高于 ResQ-R，并持续优于所有基线方法，包括 SQLSmith-Q（175.51 分钟）、LLMGen（313.51 分钟）和 SQLBarber（1454.27 分钟）。这凸显了局部模型是使目标驱动的查询生成能够应用于大规模数据库测试环境的关键组件。

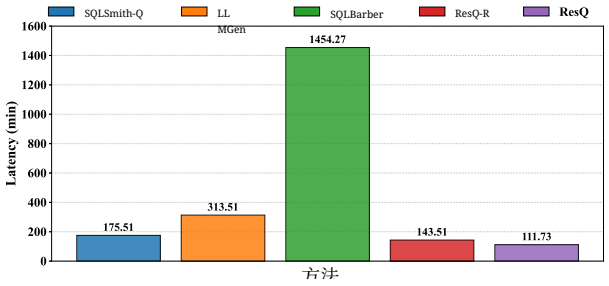


图4：本地模型消融研究

8 相关工作

现有的细粒度查询生成方法大致可分为两类。

（1）基于优化器的成本感知查询生成。这些方法生成 SQL 查询，以匹配单一基于优化器的成本目标，例如结果基数或计划成本，旨在生成能够对数据库管理系统施加压力的多样化测试查询。它们通常分为三类。（i）随机生成。SQLSmith [22] 是经典的随机查询生成器，能够针对给定数据库快速生成大量查询，但通常会产生无效查询，并且在满足特定目标方面的可控性有限。Bati等人 [1] 使用遗传算法提升随机测试生成的覆盖率和效率，但没有明确针对关键成本约束（例如基数）或结构要求。（ii）基于模板的生成。这类方法首先构造满足结构约束的查询模板，然后调整谓词以满足目标基数/成本范围。Bruno等人 [2] 表明，逻辑约束感知的查询生成是NP难问题，并提出通过对谓词值执行爬山法来实现基数目标。Mishra等人 [20] 将这一设置扩展到多个基数目标，并引入剪枝算法以降低优化成本。近年来，SQLBarber [14] 使用大语言模型根据用户指定的结构要求（例如算子数量）迭代提出并细化 SQL 模板，然后应用贝叶斯优化，将谓词调整至目标成本/基数范围。（iii）基于学习的生成。LearnedSQLGen [30] 训练强化学习智能体，以逐个 SQL词元生成 SQL，并使用逻辑成本估计作为奖励，同时利用有限状态机确保语法有效性。

（2）真实性能感知查询生成。这一类方法关注云端生产环境工作负载中观测到的执行级信号，例如 CPU 时间、扫描字节数、内存/I/O 使用量和物理算子分布。由于出于隐私原因，生产环境 SQL 和用户数据通常无法获取，这些方法旨在基于匿名化性能跟踪数据，在受控环境中合成可执行查询，以复现类似的执行行为。Zhou等人 [31] 提出了一种基于大语言模型的多目标 SQL 生成工作流，并在 PBench 中通过多轮交互以及查询执行反馈，迭代细化候选项。然而，在 PBench 中，这一生成组件主要用于扩充查询池，以进行粗粒度工作负载合成，并未明确针对逐查询执行配置文件匹配准确率进行优化，也未对其进行系统评估。

定位。真实性能感知场景与定向查询生成 [2, 14, 20, 30]有根本区别：我们的目标并非匹配单一逻辑计划/优化器级成本目标，而是在隐私和代理数据库约束下，匹配每个查询的多种观测执行级指标（及其重复模式）。

9 结论

我们介绍了 ResQ，这是一种在隐私约束下进行细粒度、由跟踪数据驱动的工作负载合成方法，可生成符合逐查询执行目标和结构信号的可执行 SQL 查询。通过结合两阶段查询图生成、性能感知界定、轻量级局部模型和重复感知复用，ResQ 在 Snowset、Redset 以及新发布的 Bendset 上实现了高保真度和高效率。

参考文献

[1] Hardik Bati、Leo Giakoumakis、Steve Herbert 和 Aleksandras Surna。2007 年。一种用于数据库系统随机测试的遗传方法。*Very Large Data Bases,Very Large Data Bases*（2007 年 9 月）。[2] Nicolas Bruno、Surajit Chaudhuri 和 Dilys Thomas。2006 年。生成满足基数约束的查询以进行数据库管理系统测试。*IEEE Transactions on Knowledge and Data Engineering* 18 卷，12 期（2006 年），1721–1725。[3] databendlabs。2025 年。Databend：面向分析、搜索和人工智能的 Rust 数据仓库。https://github.com/databendlabs/databend。GitHub 仓库。[4] Jialin Ding、Ryan Marcus、Andreas Kipf、Vikram Nathan、Aniruddha Nrusimha、Kapil Vaidya、Alexander van Renen 和 Tim Kraska。2022 年。SageDB：面向实例优化的数据分析系统。*Proceedings of the VLDB Endowment* 15 卷，13 期（2022 年）。[5] Haowen Dong、Chao Zhang、Guoliang Li 和 Huanchen Zhang。2024 年。云原生数据库：综述。*IEEE Transactionson Knowledge and Data Engineering* 36 卷，12 期（2024 年），7772–7791。[6] Anshuman Dutt、Chi Wang、Vivek Narasayya 和 Surajit Chaudhuri。2020 年。使用低开销回归模型高效近似选择率函数。*Proceedings of the VLDB Endowment* 13 卷，12 期（2020 年），2215–2228。[7] Hua Fan、Dachao Fu、Xu Wang、Jiachi Zhang、Chaoji Zuo、Zhengyi Wu、Miao Zhang、Kang Yuan、Xizi Ni、Guocheng Huo 等。2024 年。迈向云端数百万数据库传输服务。*Proceedings ofthe VLDB Endowment* 17 卷，12 期（2024 年），4001–4013。[8] Benjamin Hilprecht、Andreas Schmidt、Moritz Kulessa、Alejandro Molina、Kris-tian Kersting 和 Carsten Binnig。2019 年。DeepDB：从数据而非查询中学习！*CoRR* abs/1909.00607（2019 年）。arXiv:1909.00607 http://arxiv.org/abs/1909.00607[9] holgern。2026 年。scikit-optimize：提供 scipy.optimize 接口的序贯模型优化。https://github.com/holgern/scikit-optimize。[10] Guolin Ke、Qi Meng、Thomas Finley、Taifeng Wang、Wei Chen、Weidong Ma、Qiwei Ye 和 Tie-Yan Liu。2017 年。LightGBM：一种高效的梯度提升决策树。*Advances in neural information processing systems* 30（2017 年）。[11] Martin Kiefer、Max Heimel、Sebastian Breß 和 Volker Markl。2017 年。使用带宽优化核密度模型估计连接选择率。*Proceedings of the VLDB Endowment* 10 卷，13 期（2017 年），2085–2096。[12] Andreas Kipf、Thomas Kipf、Bernhard Radke、Viktor Leis、Peter Boncz 和 Alfons Kemper。2018 年。学习式基数：使用深度学习估计相关连接。*arXiv preprint arXiv:1809.00677*（2018 年）。[13] Skander Krid、Mihail Stoian 和 Andreas Kipf。2025 年。RedBench：反映真实工作负载的基准。*arXiv preprint arXiv:2506.12488*（2025 年）。[14] Jiale Lao 和 Immanuel Trummer。2025 年。SQLBarber：利用大语言模型生成定制化、现实的 SQL 工作负载的系统。arXiv:2507.06192 [cs.DB] https://arxiv.org/abs/2507.06192[15] Jiale Lao、Yibo Wang、Yufei Li、Jianping Wang、Yunjia Zhang、Zhiyuan Cheng、Wanghu Chen、Mingjie Tang 和 Jianguo Wang。2025 年。GPTuner：基于大语言模型的数据库调优系统。*ACM SIGMOD Record* 54 卷，1 期（2025 年），101–110。[16] Viktor Leis 和 Maximilian Kuschewski。2021 年。迈向云端成本最优的查询处理。*Proceedings of the VLDB Endowment* 14 卷，9 期（2021 年 5 月），1606–1612。https://doi.org/10.14778/3461535.3461549

[17] 李飞飞。2019。阿里巴巴的云原生数据库系统：机遇与挑战。*Proceedings of the VLDB Endowment* 12, 12 (2019), 2263–2272. [18] Ryan Marcus、Jeffrey Tao、Peizhi Wu 和 Zijie Zhao。[n.d.]。工业数据库工作负载中的幸存者偏差。([n. d.])。[19] MichailGeorgoulakis Misegiannis、Daniel Ritter、Jana Giceva 和 Technische-Universität München。[n.d.]。CloudGlide：剖析云分析的全貌。([n. d.])。[20] Chaitanya Mishra、Nick Koudas 和 Calisto Zuzarte。2008。为数据库测试生成目标查询。载于*Proceedings of the 2008 ACM SIGMOD international conference on Management of data*。https://doi.org/

10 1145 1376616 1376668 。/。

[21] 2025 Tobias Schmidt、Viktor Leis、Peter Boncz 和 Thomas Neumann。。Sql-storm：将数据库基准测试带入大语言模型时代。*Proceedings of the VLDB Endowment* 18, 11 (2025), 4144–4157。[22] A. Seltenreich。2020。SQLsmith。https://github.com/anse1/sqlsmith。[23] Alexander van Renen、Dominik Horn、Pascal Pfeil、Kapil Vaidya、Wenjian Dong、Murali Narayanaswamy、Zhengchun Liu、Gaurav Saxena、Andreas Kipf 和 Tim Kraska。2024。为何TPC还不够：Amazon Redshift集群分析。*Proceedings of the VLDB Endowment* 17, 11 (2024), 3694–3706。[24] Midhul Vuppalapati、Justin Miron、Rachit Agarwal、Dan Truong、Ashish Motivala 和 Thierry Cruanes。2020。在解耦存储上构建弹性查询引擎。载于第*17*届*USENIX*网络系统设计与实现研讨会（*NSDI 20*）。449–462。[25] Lucas Woltmann、Claudio Hartmann、Maik Thiele、Dirk Habich 和 Wolfgang Lehner。2019。基于局部深度学习模型的基数估计。载于第二届利用人工智能技术进行数据管理国际研讨会论文集。1–8。[26] Zongheng Yang、Amog Kamsetty、Sifei Luan、Eric Liang、Yan Duan、Xi Chen 和 Ion Stoica。2020。NeuroCard：适用于所有表的基数估计器。*arXiv*预印本 *arXiv:2006.08109* (2020)。[27] Geoffrey X Yu、Ziniu Wu、Ferdi Kossmann、Tianyu Li、Markos Markakis、Amadou Ngom、Samuel Madden 和 Tim Kraska。2024。云蓝图设计：通过BRAD统一并自动优化云数据基础设施——扩展版。*arXiv*预印本 *arXiv:2407.15363* (2024)。[28] Zhongwei Yue、Shujian Peng、Peng Cai、Xuan Zhou、Huiqi Hu、Rong Zhang、Quanqing Xu 和 Chuanhui Yang。2024。通过多任务学习实现功能感知的数据库调优。载于*2024*年*IEEE*第*40*届国际数据工程会议（*ICDE*）。IEEE，83–95。[29] Chao Zhang、Guoliang Li、Leyao Liu、Tao Lv 和 Ju Fan。2025。CloudyBench：用于全面评估云原生数据库的测试平台。载于*2025*年*IEEE*第*41*届国际数据工程会议（*ICDE*）。IEEE计算机学会，2535–2547。[30] Lixi Zhang、Chengliang Chai、Xuanhe Zhou 和 Guoliang Li。2022。Learnedsql-gen：利用强化学习进行约束感知SQL生成。载于*2022*年数据管理国际会议论文集。945–958。[31] Yan Zhou、Chunwei Liu、Bhuvan Urgaonkar、Zhengle Wang、Magnus Mueller、Chao Zhang、Songyue Zhang、Pascal Pfeil、Dominik Horn、Zhengchun Liu、Davide Pagano、Tim Kraska、Samuel Madden 和 Ju Fan。2025。PBench：用于云分析基准测试的真实统计数据工作负载合成器。*Proc. VLDB Endow.* 18, 11 (2025年7月), 3883–3895。https://doi.org/10.14778/3749646.3749661