COOOL：一种用于 SQL 提示推荐的学习排序方法

徐向红 字节跳动 清华大学
中国北京
x
xh20
@mails.tsinghua.edu.cn

Zhibing Zhao 字节跳动 美国
西雅图
zhibing.zhao@bytedance.co
m

张铁英
字节跳动
美国圣何塞
tieying.zhang@bytedance.com

康荣 字节跳动 北京，中国
k
angrong.cn
@bytedance.com

孙鲁明
字节跳动
北京，中国
sunluming@bytedance.com

陈建军
字节跳动
圣何塞，美国
jianjun.chen@bytedance.com

arXiv:2304.04407v1 [cs.DB] 10 Apr 2023

摘要

查询优化是每个数据库管理系统（DBMS）的关键组成部分，因为它决定了查询执行的效率。许多研究已将机器学习（ML）技术引入成本建模、基数估计和端到端学习型优化器，但由于训练时间过长、缺乏可解释性以及集成代价高昂，其中很少有方法被证明具有实用性。近期一项研究提出了一种通过针对每个查询推荐提示来优化查询的实用方法，但它存在两个固有问题。首先，该方法遵循回归框架来预测每个查询计划的绝对延迟，这极具挑战性，因为某个查询的查询计划延迟可能跨越多个数量级。其次，该方法要求为每个数据集训练一个模型，这限制了训练所得模型在实践中的应用。本文提出 COOOL，通过学习排序预测查询计划的成本顺序，以与数据库管理系统协同工作。COOOL 不估计绝对代价，而是采用基于排名的方法计算查询计划代价的相对排名分数。我们证明，从理论上讲，COOOL 能够区分具有不同延迟的查询计划。我们在 PostgreSQL 上实现了 COOOL，并在 join-order-benchmark 和 TPC-H 数据集上进行了广泛实验。结果表明，在单数据集任务以及面向多个数据集的统一模型任务中，COOOL 均优于 PostgreSQL 和最先进的方法。我们的实验还从表征学习的角度揭示了 COOOL 优于回归方法的部分原因，这可能为未来研究提供指导。

PVLDB 引用格式：
徐向红、Zhibing Zhao、张铁英、康荣、孙鲁明和陈建军。COOOL：一种用于 SQL 提示推荐的学习排序方法。PVLDB，14(1)：XXX-XXX，2020。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/xianghongxu/COOOL 上提供。

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。请访问 https://creativecommons.org/licenses/by-nc-nd/4.0/ 查看该许可协议副本。如需进行许可协议未涵盖的任何用途，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。出版权授权给 VLDB 基金会。Proceedings of the VLDB Endowment，第 14 卷，第 1 期，ISSN 2150-8097。doi:XX.XX/XXX.XX

1 引言

查询优化对于每个数据库管理系统（DBMS）的性能都至关重要。一个 SQL 查询通常有许多候选计划，它们的最终输出相同，但执行延迟却可能相差很大。查询优化的目标是从庞大的搜索空间中，为每个查询以足够高的准确性选出延迟最低的最佳候选计划。

几十年来，人们一直在研究查询优化 [31]，如今它仍是一个活跃的研究领域 [35]。目前已经提出了各种基于机器学习的研究方向：成本建模、基数估计、端到端查询优化等，其中最具实用性的方法是 Bao [23]。Bao 是一种查询优化系统，利用树卷积神经网络 [27]和汤普森采样 [33] 来推荐 SQL 提示。Bao 显著提升了端到端查询优化的实用性，但它继承了以往模型的两个问题 [19, 25, 32, 41, 42]。首先，Bao 遵循与先前工作相同的基本假设，即模型需要预测每个查询计划的精确代价，从而选择代价最小的计划。它首先在回归框架下估计每个候选计划的绝对代价，然后选择估计值最小的计划。虽然准确估计代价有时是理想的，但对于现有模型而言，这极具挑战性 [1, 36, 37]。因此，模型可能做出不准确的预测，并选择延迟较高的查询计划。其次，Bao 针对每个数据集分别训练一个模型，并在相应的数据集上评估该模型。不同数据库管理系统实例中的数据集各不相同，为每个数据集训练和维护独立模型的代价很高。因此，在现实场景中，模型的泛化能力是十分理想的。

为解决这些问题，我们提出 COOOL，利用学习排序（LTR）技术估计查询计划的成本顺序，从而与数据库管理系统协作。COOOL 构建于现有数据库管理系统之上，旨在推荐查询专用提示，以促进实际使用。为了获得一个能够优化来自不同数据集的查询的统一模型，我们采用数据/模式无关的树卷积神经网络（TCNN）作为查询计划成本排序评分器，以预测候选计划的相对成本顺序。我们作出了与 Bao 类似的假设，以继承其优势：

我们假设预先定义了有限个提示集，并且所有提示集均以语义等价的查询计划为结果。

与用于预测代价的回归方法相比，学习排序方法会学习不同提示集的相对得分。由于以下原因，学习排序方法相较于此前端到端查询优化工作中的回归方法具有优势 [23, 25, 40] 。计划执行延迟的范围从几毫秒到数千秒不等。而且，对于一个查询，两个语义等价的计划在树结构上的细微差异，可能导致执行延迟出现巨大差异。这些事实给回归模型的训练和推理带来了困难，因为回归方法的目标函数是最小化 L_2 误差。平方误差公式对异常的大延迟或小延迟十分敏感，这可能导致模型性能不稳定 [40]。尽管归一化方法可以通过将原始延迟分布映射到固定区间（例如， [0,1]）来缓解这一问题，但由于查询计划之间的相对延迟差距无法得到充分体现，这些方法可能会导致延迟分布失真。对于端到端查询优化而言，准确估计每个查询计划的延迟并非必要，因为优化器只需选择其中一个计划执行即可。准确预测查询计划延迟的排序就足以完成查询优化，而这正是学习排序的目标。

LTR 是一种用于训练模型执行排序任务的监督学习框架。LTR 方法主要分为三类：逐点式、成对式和列表式。大多数逐点式方法与回归相同。成对式方法关注两个项目之间的相对顺序，而列表式方法则关注整个列表中项目的顺序。这三种方法都可以应用于大多数现有模型，例如神经网络，唯一的区别在于损失函数。在本工作中，我们采用广泛使用的树卷积神经网络（TCNN） [23, 27, 40] 作为底层模型。要训练 LTR 模型，可以使用延迟的顺序作为标签；与回归模型相比，这种方法有望更好地应对查询计划延迟跨越多个数量级这一情况。通过将绝对代价估计问题转换为相对代价顺序预测问题，COOOL 可以利用 LTR 技术训练 TCNN，使其输出能够指示哪个计划最优。具体而言，我们分别研究了成对式和列表式策略在面向查询优化的 SQL 提示推荐中的表现，因此 COOOL 有两种实现：COOOL-pair 和 COOOL-list。通过这种方式，我们还可以训练一个统一模型，以改进来自不同数据集的查询计划。

我们的实验表明，COOOL在各种设置下都能持续改进查询计划，相较于 PostgreSQL，最高可实现6.73× 倍加速。此外，我们从表征学习 [44]的角度研究了回归框架-和排序策略。我们表明，通过回归方- 法训练的模型在计划嵌入空间中存在维度坍缩 [9] 。而 COOOL使用相同的嵌入方法时不存在维度坍缩。维度坍缩将限制 ML 方法构建统一模型，因为不同数据集中的坍缩维度可能不同。

此外，由于 Bao 完全集成在 PostgreSQL 中，因此实现起来需要投入更多精力；而我们的方案则构建于 PostgreSQL 之上，这使得 COOOL 能够轻松迁移到其他数据库管理系统。

总而言之，我们做出了以下贡献：

- 我们提出 COOOL，这是一种学习模型，通过 LTR 技术预测查询计划的 **C**成本**O**顺序，与数据库管理系统**OO**协作，以为查询优化推荐更优的 SQL 提示。据我们所知，COOOL 是首个端到端查询优化方法，它维护一个统一模型，以优化来自不同数据集的查询。
- 我们从理论上证明，COOOL 在优化损失函数时能够区分延迟不同的查询计划，并从表征学习的角度验证 COOOL 优于回归方法。
- 在 join-order-benchmark 和TPC-H 上进行的综合实验表明，COOOL 在多个评估标准维度上能够超越 PostgreSQL 和最先进的方法。

2 预备知识

2.1 任务定义与形式化

令 Q表示查询集合，H $=\{HS_1, HS_2, \ldots, HS_n\}$表示 n提示集的集合。每个提示集 $HS_i \in$H 仅包含布尔标志查询提示（例如，启用哈希连接、禁用索引扫描）。对于任意查询 $q \in Q$以及 $i \in \{1, 2, \ldots, n\}$，传统优化器 Opt可以使用提示集 HS_i生成相应的计划树 t_i^q。

$$t_i^q = Opt(q, HS_i). \tag{1}$$

设 $T^q = \{t_1^q, t_2^q, \ldots, t_n^q\}$ 为查询 q 的候选计划集合。T^q 中的查询计划在语义上等价，但执行延迟可能不同。

模型 M 是一个函数，以候选计划树作为输入，并为该计划树生成一个分数。

$$s_i^q = M(q, t_i^q; \vec{\theta}), \tag{2}$$

其中， $\vec{\theta}$ 是待训练模型的参数。在学习排序场景中，查询执行引擎会选择预测排名分数最高的计划。

$$\hat{HS}^q = HS_{\arg\max_i s_i^q}, \tag{3}$$

其中，$\hat{}HS^q$ 是得分最高、对应估计代价最低的提示集。

2.2 学习排序（LTR）

在学习排序（LTR）的语境下，H 是项目集合，目标是为任何 $q \in Q$推荐 H 中的最佳项目。更具体地说，学习排序是在 s_i^q 上定义损失函数，以便底层模型 M能够经过适当训练，从而预测查询计划的排序。在本文中，“$t_{i_1}^q \succ t_{i_2}^q$”表示计划 $t_{i_1}^q$ 优于（延迟低于） $t_{i_2}^q$。给定查询 $q \in Q$，令 $\sigma_q = t_{i_1}^q \succ t_{i_2}^q \succ \ldots \succ t_{i_n}^q$ 表示按照延迟对查询计划进行的全序，其中 $t_{i_1}^q$ 的延迟最低， $t_{i_2}^q$ 的延迟次低，而 $t_{i_n}^q$ 的延迟最高。

在介绍我们的成对损失函数之前，我们先介绍一种经典模型——Plackett-Luce 模型（PL）[22, 28],，我们的成对方法正是基于该模型选定的。PL 是离散选择领域最受欢迎的模型之一，后来被用作信息检索中的列表式损失函数 [20] 以及分类任务中的 Softmax 函数 [8]。在 SQL 提示推荐的语境下，我们给出 PL 的定义如下。给定任意 $q \in Q$，$\sigma_q = t^q_{i_1} \succ t^q_{i_2} \succ \ldots \succ t^q_{i_n}$ 的概率为

$$\Pr_{\mathrm{PL}}(t^q_{i_1} \succ t^q_{i_2} \succ \ldots \succ t^q_{i_n}; \vec{\theta}) = \prod_{j=1}^{n} \frac{\exp(s^q_{i_j})}{\sum_{m=j}^{n} \exp(s^q_{i_m})}, \quad (4)$$

其中，s^q_i 是参数 $\vec{\theta}$ 的函数，该参数定义于公式 (2) 中。假设多个查询的排序彼此独立。因此，多个排序的概率simply 是每个单独排序概率的乘积。任意成对比较 $t^q_i \succ t^q_j$ 的边际概率是

$$\Pr_{\mathrm{PL}}(t^q_i \succ t^q_j; \vec{\theta}) = \frac{\exp(s^q_i)}{\exp(s^q_i) + \exp(s^q_j)}. \quad (5)$$

2.2.1 列表式损失函数。 给定训练数据 $\{\sigma_q | \forall q \in Q\}$，列表式损失函数就是负对数似然函数：

$$\mathcal{L}_{\mathrm{list}}(\vec{\theta}) = -\sum_{q \in Q} \ln \Pr_{\mathrm{PL}}(\sigma_q; \vec{\theta}), \quad (6)$$

其中，PrPL($\sigma_q; \vec{\theta}$) 的定义见方程（4）。这种列表式损失函数也与 [38] 提出的 listMLE 损失一致。 [38] 证明了该损失函数具有一致性，这意味着随着数据集规模趋于无穷，学习得到的排序将收敛于最优排序。

2.2.2 成对损失函数。 要应用成对损失函数，需要将σ_q 的完整排序 $q \in Q$ 转换为成对比较。这个过程称为排序拆分 [2]。排序拆分方法规定应如何将完整排序转换为成对比较。基本的拆分方法包括完全拆分、相邻拆分等 [2]。完全拆分意味着从一个排序中提取所有成对比较，而相邻拆分意味着仅提取相邻的成对比较。例如，给定一个排序 $t_1 \succ t_2 \succ t_3$，完全拆分会将其转换为（$t_1 \succ t_2$、 $t_1 \succ t_3$、 $t_2 \succ t_3$ ），而相邻拆分会将其转换为（ $t_1 \succ t_2$ $t_2 \succ t_3$ ）。尽管相邻拆分简单且合理，但 [2] 证明，相邻拆分可能导致不一致的参数估计，这意味着即使使用无限量的数据训练模型，也可能无法做出无偏预测。其他拆分方法更加复杂，超出了本文的讨论范围。

设 $P = \{\pi_1, \pi_2, \ldots, \pi_m\}$ 为成对比较的数据集，其中，对于任意 $j \in \{1, 2, \ldots, m\}$，$\pi_j$ 具有 $t^{q_j}_{i_1} \succ t^{q_j}_{i_2}$ 的形式。这里，q^j 表示与 π_j 对应的查询。对于不同的 j 值，q_j可能指向同一个查询，因为可以从同一个查询中提取出不同的成对比较。目标函数为

$$\mathcal{L}_{\mathrm{pair}}(\vec{\theta}) = -\sum_{j=1}^{m} \ln \Pr_{\mathrm{PL}}(\pi_j; \vec{\theta}), \quad (7)$$

其中，PrPL(p_j; $\vec{\theta}$) 的定义见公式 (5)。模型参数可以通过最大化 $L(\vec{\theta})$ 来估计。

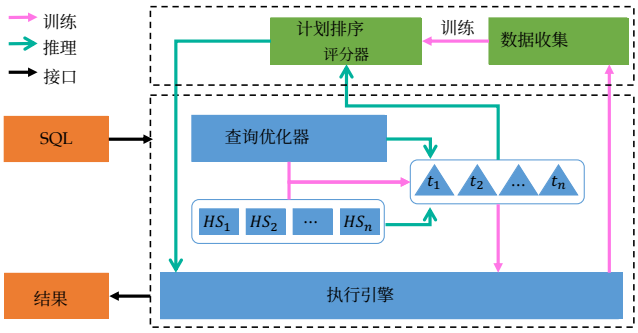


图1：COOOL 流水线的简要示意。

这等价于基于公式 (5) 最大化复合边际对数似然。 [14] 和 [47] 证明，在正确的排名拆分方法下，可以高效地估计 $\vec{\theta}$。完全拆分会从完整排名中提取所有成对比较，是一种正确的拆分方法。

3 COOOL 架构

COOOL 的核心组件是神经计划排名评分器，用于估计候选计划的相对排名分数。在训练阶段，我们执行查询计划并收集其性能指标作为训练数据。然后，我们利用成对式方法或列表式方法来训练排名评分器。在推理阶段，当用户提交查询时，传统优化器将利用相应的提示集生成 n 查询计划。接下来，评分器将计算每个计划的相对排名分数，并向执行引擎推荐最优计划。COOOL 的数据流流水线如图 1 所示。

数据收集。 我们的方法采用标准的监督学习范式进行训练。对于给定的查询，我们通过底层传统优化器为每个查询生成与提示集 H 对应的 n 查询计划。然后，我们将它们发送到执行引擎，并记录每个查询计划的实际执行性能。收集的数据用于训练神经网络排名评分器，训练阶段独立于数据库管理系统。

成本顺序估计。 为了以最小的估计延迟执行最优计划，我们需要向数据库管理系统推荐其对应的提示集。评分器以计划树作为输入，并输出该计划的排名分数；通过对所有候选计划的分数进行排序，即可获得估计延迟的顺序。具体而言，我们首先将输入计划树的节点转换为向量，然后将向量树输入由 TCNN 构建的计划嵌入模型。最后，我们将计划嵌入输入多层感知机(MLP)，以计算该计划的估计分数。在训练阶段，我们使用收集到的计划和延迟数据来训练模型。因此，在推理阶段，它可以根据延迟估计计划的相对顺序，并与数据库管理系统协作改进查询计划。

假设和比较。 我们假设，将每个提示集应用于给定查询都会生成语义等价的计划。此外，这些提示应用于整个查询，而不是

部分计划。尽管可以使用细粒度提示（例如，允许特定表之间使用嵌套循环连接，而其他表之间不允许），但这会带来指数级的候选计划搜索空间，从而显著增加训练和推理开销。出于实用性考虑，我们作出了与 Bao 相同的假设。

Bao 的实现需要付出更多努力，因为它完全集成在 PostgreSQL 中，而我们是在 PostgreSQL 之上构建的，这使得 COOOL 易于迁移到其他数据库管理系统。此外，我们进一步尝试维护一个统一模型，以优化来自不同数据集的查询；此前的端到端查询优化研究尚未对此进行探讨。

4 COOOL 用于提示推荐

首先，我们介绍以查询计划为输入并计算相对排名分数的成本顺序估计模型。接下来，我们利用成对式和列表式方法来构建训练循环。最后，我们详细阐述排名损失如何用于选择最佳计划的理论分析。

4.1 代价顺序估计

每个提示集都对应一个查询计划树，因此，为给定查询推荐最优提示集，就是选择代价排名分数最高的计划树，如公式（3）所示。与[23, 25, 40], 类似，我们使用 TCNN 获取计划嵌入，并利用 MLP 计算排名分数。

计划树向量化。 我们可以使用底层优化器提供的 EXPLAIN 命令来获取计划树文本，如公式（1）所示。首先，我们需要将计划树中的每个节点转换为一个向量。与 [23], 相同，我们使用一种与数据/模式无关的编码方案，该方案仅包含算子类型的独热编码，以及由底层传统优化器提供的基数和代价。然后，我们将原始树转换为二叉树，以便进行树卷积操作。

独热节点操作类型编码。 我们总结计划树中所有操作的类型（嵌套循环、哈希连接、归并连接、顺序扫描、索引扫描、仅索引扫描和位图索引扫描），并为这七种操作编号。然后，我们为每个节点创建一个由相应类型数量的位组成的向量，并将每个节点对应类型的位设置为高位。例如，$E_o(v)$ 是节点 v 的独热编码，而 $E_o(v)[i] = 1$ 表示节点 v 是第 i种操作类型，$E_o(v)$ 中的其余元素均为0。尽管独热节点类型编码很简单，但它能够提取计划树中的结构信息。

树节点向量化。 除了操作信息外，每个节点还可以包含代价和基数。可以从多个传统优化器和学习模型中获取代价和基数，但为简洁起见，我们只使用底层传统优化器得出的两个值。因此，节点编码由操作类型编码、基数和代价拼接而成，即 $E(v) = \text{Concat}(E_o(v), \text{Cost}(v), \text{Card}(v))$，其中 $\text{Cost}(v)$ 和 $\text{Card}(v)$ 分别是传统优化器估算的代价和基数。我们将节点编码方法应用于计划树中的节点，以获得向量化的计划树。

计划树二叉化。计划树中的一些节点可能只有一个子节点，例如聚合和排序操作对应的节点。为了便于执行树卷积操作，我们通过为每个只有一个子节点的节点添加一个伪子节点 Null，将非二叉树转换为二叉树，并将伪子节点的代价和基数设为 0。随后，原始计划树 t 可以转换为向量化树 p。

4.1.1 TCNN 计划嵌入。TCNN 在 [27] 中被提出，用于处理编程语言处理中的树结构数据。TCNN 首次在 [25], 中用于计划表示，并在 [23, 40] 中得到确立。本节将简要介绍如何使用 TCNN 表示计划，更多技术细节请参阅 [25, 27] 。

在执行引擎执行原始计划树时，一个节点的计算依赖于其子节点的结果。基于这一事实，计划嵌入方法应体现递归特性，以获得恰当的归纳偏置 [26]。为与计划执行保持一致，模型自然需要同时捕获节点及其子节点的特征。具体而言，设 $l(v)$ 和 $r(v)$ 分别是节点 v 的左、右子节点。向量 $E(v)$ 中的统计代价/基数信息与 $E(l(v))$ 和 $E(r(v))$ 密切相关。树卷积可以自然地满足这一需求。

树卷积类似于图像卷积，它具有二叉树结构的滤波器，用于捕获局部特征。我们以树卷积滤波器为例，滤波器中有三个权重向量，即 w， w_l， w_r。将树卷积应用于当前节点$E(v)$可以获得新的表示：

$$E(v)' = \sigma(E(v) \odot w + E(l(v)) \odot w_l + E(r(v)) \odot w_r), \quad (8)$$

其中， σ 是一个非线性激活函数， $\odot$ 是点积运算。节点 v 的新表示包含其子节点的信息。通过堆叠树卷积操作，潜在表示将拥有比仅与子节点交互更大的感受野。通过这种方式，模型能够捕获长链计划执行的高级特征，以表示一个节点。树卷积操作的输出是一个与输入具有相同结构的树，因此我们采用动态池化方法来聚合所有节点的潜在表示，以表示查询计划。总之，我们可以通过 $\vec{p} = \text{TCNN}(p)$获得计划嵌入，其中 $\vec{p} \in \mathrm{R}^h$ 是计划表示的向量， h 是计划树嵌入空间的大小。

4.1.2 排名分数计算。 最后，我们可以利用计划嵌入来计算相对排名分数。我们使用一个简单的 MLP，以计划嵌入向量作为输入，并输出一个标量作为排名分数 s，即 $s = \text{MLP}(\vec{p})$。在 MLP 中堆叠全连接层和非线性激活函数可以增强表示能力，这种做法已广泛应用于查询优化中 [25, 40]。为简便起见，我们添加一个隐藏层和一个激活函数。通过对所有候选计划的排名分数进行排序，即可得到成本顺序。

4.2 学习排序训练循环

训练循环由三部分组成：数据收集与去重、标签映射与成对数据提取，以及模型训练与评估。

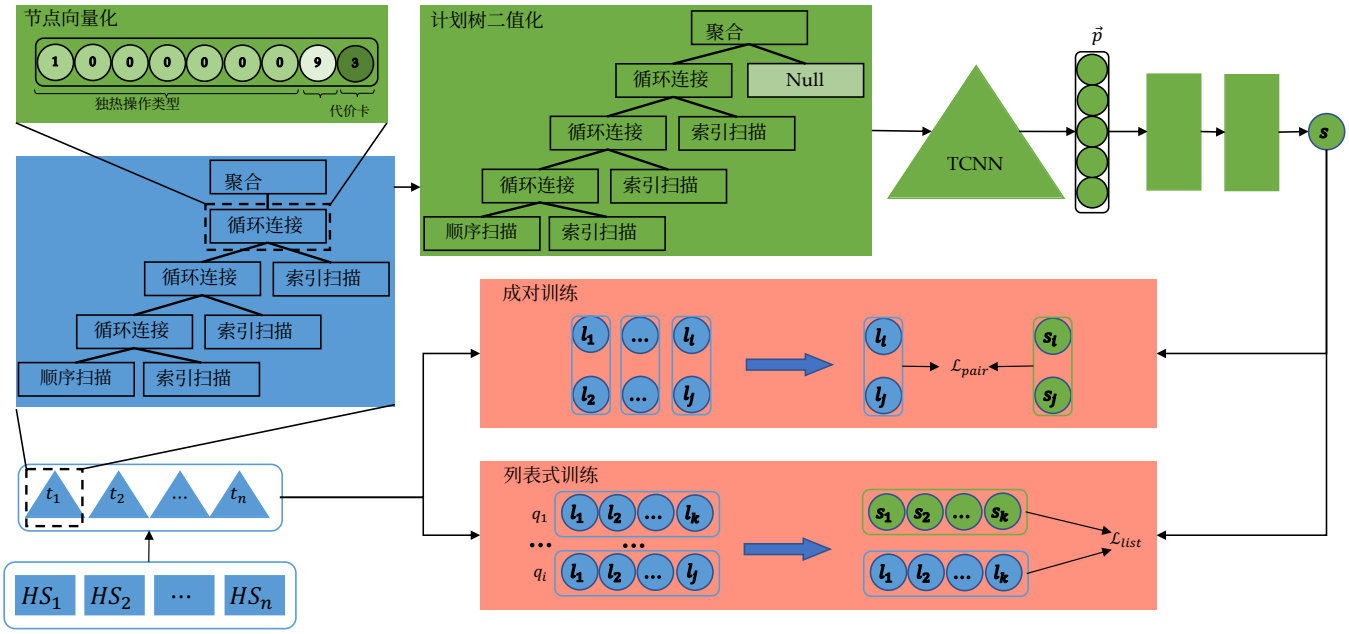


图2：使用成对式和列表式学习排序技术进行树卷积神经网络的成本顺序估计。

4.2.1 数据收集与去重。 令 $Q_{训练}$ 表示训练查询集合。我们使用传统优化器为每个查询$q \in Q_{训练}$ 生成 n 个计划，如公式（1）所示。然后，我们将这些计划发送给执行引擎，并记录每个数据点（查询 $= q$、计划 $= t$、延迟 $= l$）。由于对于给定查询，不同提示可能会产生相同的查询计划，因此存在重复的查询计划。我们会在成对训练和列表式训练循环中移除重复的查询计划。

*4.2.2*标签映射与成对数据提取。 由于较低的延迟表示更优的查询计划，我们使用每个查询计划的延迟的倒数作为标签，以反转查询计划的顺序。任何其他能够反转顺序的映射函数都同样适用，因为重要的只是顺序。

经过标签映射后，我们针对每个查询得到一个按延迟倒数排序的查询计划列表。对于每个包含 n^q 个查询计划的列表（$n^q \le n$ 去重后），我们提取所有 $\binom{n^q}{2}$ 个成对比较，以获得成对数据 $P = \{\pi_1, \pi_2, \ldots, \pi_m\}$，其中，对于任意 $j \in \{1, 2, \ldots, m\}$，$\pi_j = t_{i_1}^{q_j} > t_{i_2}^{q_j}$。

*4.2.3*模型训练与评估。 我们通过最大化成对或列表式对数似然函数来计算模型参-⃗数 θ。具体而言，对于列表式方法，通过最小化公式（6）中定义的列表式损失来计算模型型参-⃗数 θ。对于成对方法，通过最小化公式（7）中定义的成对损失来计算 $\vec{\theta}$。更多详情请参阅第 2.2 节。

在推理阶段，我们使用训练好的模型为每个候选计划计算分数，并对分数排序以获得相应的顺序，然后为每个查询选择估计的最佳计划来执行。

4.3 理论分析

本节简要分析 COOOL 如何根据不同查询计划的延迟顺序进行学习。为此，对于给定的查询 $q \in Q$，我们考虑查询计划 $\{t_1, t_2, \ldots\ldots, t_n\}$，以及它们对应的延迟 $\{l_1, l_2, \ldots\ldots, l_n\}$。我们的目标是使用模型选出最佳计划。在训练阶段，它们有不同的初始分数 $\{s_1, s_2, \ldots\ldots, s_n\}$。不失一般性，我们假设 $l_1 > l_2 > \ldots > l_n$.

4.3.1 成对方法。我们首先介绍成对方法，因为它比列表式方法更简单直观。在成对方法中，每行数据由某个查询 $q \in Q$对应的两个查询计划组成，分别记为 $t_{i_1}^q$ 和 $t_{i_2}^q$。不失一般性，我们假设它们对应的延迟$l_{i_1} > l_{i_2}$。令 s_{i_1} 和 s_{i_2} 分别表示模型对这两个查询计划的输出，并定义 $\delta = s_{i_2} - s_{i_1}$。为简单起见，我们聚焦于这对查询计划，并令 $\mathrm{L}_{pair}^{q\,i_1\,i_2}$表示这对查询计划上的损失。我们有

$$\begin{aligned}\mathcal{L}_{pair}^{q,i_1,i_2} &= -\ln \frac{\exp(s_{i_2})}{\exp(s_{i_1}) + \exp(s_{i_2})} \\ &= -\ln \frac{\exp(s_{i_1}+\delta)}{\exp(s_{i_1}) + \exp(s_{i_1}+\delta)} \\ &= -\ln \frac{\exp(\delta)}{1+\exp(\delta)} \\ &= -\delta + \ln(1+\exp(\delta))\end{aligned}$$

对于 $s_{i_1} = 0$ 因为损失只取决于 δ。

$\mathrm{L}_{pair}^{q\,i_1\,i_2}$关于 δas 的偏导数

$$\frac{\partial \mathcal{L}_{pair}^{q,i_1,i_2}}{\partial \delta_i} = -1 + \frac{\exp(\delta)}{1+\exp(\delta)} < 0. \tag{9}$$

这意味着， δ的增加会导致损失函- 数 $L_{pair}^{q\,i_1\,i_2}$„减小。 δ趋于上升，同时在训练过程中，损失函数 $L_{pair}^{q\,i_1\,i_2}$„不断减小，这是我们所期望的。

4.3.2 列表式方法。 我们关注相邻查询计划的模型输出之间的差异，即 $\delta_i = s_{i+1} - s_i$对所有 $i \in \{1, 2, \ldots, n-1\}$。为方便起见，我们定义 $\delta_0 = 0$。那么对于所有 $i = \{1, 2, \ldots, n\}$，有

$$s_i = s_1 + \sum_{j=0}^{i-1} \delta_j.$$

列表式方法的损失函数可以写为：

$$\begin{aligned}
\mathcal{L}_{list} &= -\ln \prod_{j=1}^{n} \frac{\exp(s_{n-j+1})}{\sum_{m=1}^{n-j+1} \exp(s_m)} \\
&= -\sum_{j=1}^{n} \ln \frac{\exp(s_1 + \sum_{k=0}^{n-j} \delta_k)}{\sum_{m=1}^{n-j+1} \exp(s_1 + \sum_{k=0}^{m-1} \delta_k)} \\
&= -\sum_{j=1}^{n} \ln \frac{\exp(\sum_{k=0}^{n-j} \delta_k)}{\sum_{m=1}^{n-j+1} \exp(\sum_{k=0}^{m-1} \delta_k)} \\
&= -\sum_{j=1}^{n} \left(\sum_{k=0}^{n-j} \delta_k - \ln(\sum_{m=1}^{n-j+1} \exp(\sum_{k=0}^{m-1} \delta_k)) \right).
\end{aligned}$$

在这个等式中，第一个等号是将方程 (4) 代入方程 (6) 得到的，第二个等号是将 s_i替换为 $s_1 + \sum_{j=0}^{i-1} \delta_i$ 得到的。由于 $\exp(s_1) > 0$，将分子和分母同时除以 $\exp(s_1)$，即可得到第三个等号。最后一个等号则源于 ln() 函数的性质。

现在，我们计算任意 $i \in \{1, 2, \ldots, n-1\}$ 下 L_{list}关于 δ_i的偏导数：

$$\frac{\partial \mathcal{L}_{list}}{\partial \delta_i} = -\sum_{j=1}^{n-i} \left(1 - \frac{\sum_{m=i+1}^{n-j+1} \exp(\sum_{k=0}^{m-1} \delta_k)}{\sum_{m=1}^{n-j+1} \exp(\sum_{k=0}^{m-1} \delta_k)} \right) < 0. \tag{10}$$

该不等式成立，因为当 $1 \le i \le n-j$时，我们有

$$\sum_{m=i+1}^{n-j+1} \exp(\sum_{k=0}^{m-1} \delta_k) < \sum_{m=1}^{n-j+1} \exp(\sum_{k=0}^{m-1} \delta_k).$$

这意味着，对于每个 $i \in \{1, 2、\ldots\ldots、 n\}$， δ_i的增加会导致损失函数L_{list}的减少。在训练过程中，当损失函数 L_{list} 被最小化时， δ_i往往会增大，这是我们所期望的。

总而言之，通过最小化 L_{list} 或 L^{pair}，不同查询计划的排名分数差异会逐渐增大，这意味着我们的方法能够将最佳计划与其他计划区分开来。

5 实验

为了全面评估 COOOL 的性能，我们针对三种场景开展实验：单实例、工作负载迁移和维护一个模型 （统一模型）。第一种场景在查询优化机器学习领域很常见，指的是针对每个数据集学习一个模型。相关研究很少关注其他场景，但它们对于机器学习模型在数据库管理系统中的部署至关重要，尤其是最后一种场景，它意味着训练一个模型，以改进来自不同数据集的查询计划。

因此，我们开展了广泛的实验，主要回答以下研究问题（RQs）：

- *RQ1*：与基线方法相比，*canCOOOL* 能否在总查询执行延迟和改进慢查询方面实现最佳性能？
- *RQ2*：能否将与模式无关的模型直接迁移到另一个数据集？
- *RQ3*：所提出的方法能否通过维护一个统一模型，改进来自不同数据集的查询计划？
- *RQ4*：实验能否揭示 *COOOL* 方法优于 *Bao* 的原因？

5.1 实验设置

数据集和工作负载。 我们使用广泛采用的开源数据集（IMDB、TPC-H）及其对应的工作负载（JOB、TPC-H）。为确保比较公平并便于复现，我们不修改原始查询。

- **连接顺序基准测试 (JOB)**。JOB [17] 包含 113 个分析型查询，旨在使用从真实世界收集的互联网电影数据库 (IMDB) 对查询优化器进行压力测试。这些查询来自 33 个模板，涉及复杂连接（每个查询的连接数为 3 至 16 个，平均为 8 个）。• **TPC-H**。TPC-H [29]是一种标准分析型基准测试，其数据和查询均由均匀分布生成。我们使用 10 的规模因子。TPC-H 中有 22 个模板，我们省略了模板 #2 和 #19，因为它们的计划树中有些节点的子节点超过两个，导致树卷积操作无法处理。对于每个模板，我们使用官方 TPC-H 查询生成程序[1]生成 10 个查询。

基线方法。 我们将提出的模型与传统方法和最先进的方法进行比较，具体如下。

- PostgreSQL：我们使用 PostgreSQL（版本 12.5）自身的优化器，并采用默认设置。
- Bao：我们对 Bao 源代码进行了大幅优化[2]，具体如下。首先，我们使用了 Bao 论文中的全部 48 个提示集，而不是开源代码中的 5 个提示集。其次，由于我们使用的是未经修改查询的标准基准测试，因此我们使用训练集中过去所有经过充分探索的执行经验来训练 Bao。

模型实现。 我们使用三层 TCNN，通道数分别设置为 {256, 128, 64}，计划嵌入空间的大小 h 为 64，MLP 的隐藏层大小设为 32。激活函数为 Leaky ReLU [39], ，优化器为 Adam [15] ，初始学习率为 0.001。我们基于训练损失，在 10 个训练轮次后采用提前停止机制，保存验证集上表现最佳的模型，并报告测试集上的结果。

[1]https://www.tpc.org/tpc_documents_current_versions/current_specifications5.asp [2]https://github.com/learnedsystems/baoforpostgresql

设备与配置。 我们使用一台配备 16 GB RAM 的虚拟机，搭载 8 核 Xeon(R) Platinum 8260 2.4GHz CPU。为了获得出色的数据库管理系统性能，我们按照 PGTune[3]的建议配置 PostgreSQL 参数：4 GB 共享缓冲区、12 GB 有效缓存大小等。我们使用 Pytorch1.12.0 实现模型，并使用 NVIDIA V100 GPU。

评估协议。 每项实验重复 10 次，除统一模型场景外，剔除测试结果中最好和最差的结果，然后报告其余运行的平均性能。测试集的总执行延迟定义为各查询执行延迟之和。验证集占训练集的 10%，但 TPC-H 的“重复”设置除外，这些设置使用训练集的 20%。

5.2 单实例实验（RQ1）

本节重点关注单实例场景下不同优化方法的性能。根据先前工作的评估细节 [23, 40],，单实例优化场景主要有两种评估设置：1）将标准基准随机划分为训练集和测试集；2）将标准基准作为测试集，并通过随机替换谓词扩充查询，使训练数据覆盖测试集中的所有模板。这两个评估方面并不全面，因此我们在不添加额外查询的情况下使用标准基准，并对其进行如下调整。

模型性能的评估标准。 我们不采用完全随机的方式选择测试数据，而是从多个评估维度对模型进行评估，以便通过多种方式测试数据选择。首先，我们考虑两种真实场景：临时场景和重复场景。

在 **临时**场景中，查询来自未包含在训练数据中的模板。更具体地说，我们首先从数据集中选择若干模板，并将所选模板中的所有查询用作测试数据。这对应于这样一种真实世界场景：模型需要为一个与先前查询截然不同的查询推荐一个提示集。我们分别使用 JOB数据集中的七个模板和 TPC-H 数据集中的四个模板作为测试集。

在重复场景中，测试数据中的查询与训练数据中的查询“相似”，但并不相同。这里的“相似”是指这些查询与训练数据中的查询来自相同的模板。实践中，我们从每个模板中选取一个或多个查询作为测试集，并将其余查询保留在训练集中。对于 JOB 数据集，我们从每个模板中选取一个查询；对于 TPC-H 数据集，则从每个模板中选取两个查询作为测试数据。此场景记为 **重复**。对于来自同一模板的查询，我们取其执行延迟的平均值来表示相应模板的延迟。

对于这两种场景，我们分别在随机和尾延迟评估协议下全面评估模型的性能，因此分别进行随机模板/查询选择和慢速模板/查询选择，简称为**随机** 和 **慢速** 。

[3]https://pgtune.leopard.in.ua/#/

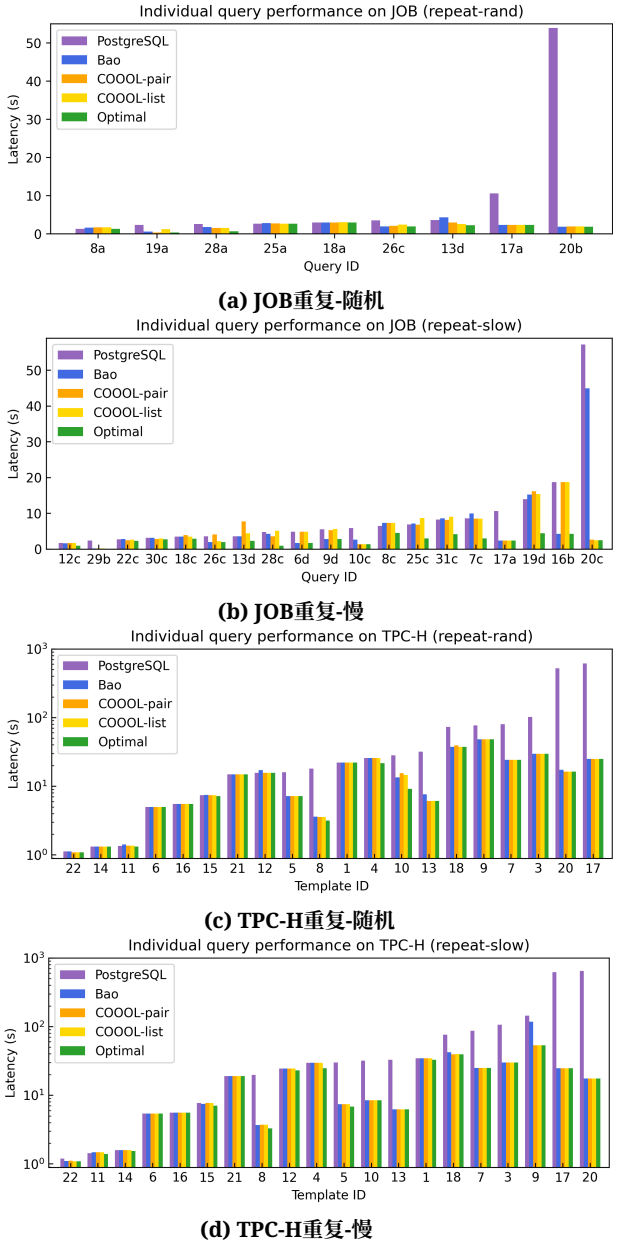


图 3：单实例场景下各模型的单个查询性能。

单实例实验的总体结果汇总于表1中。“repeat”设置下的单个查询性能如图3所示，其中我们展示了在PostgreSQL 上执行延迟大于 1s的查询，以便于观察；“Optimal”表示给定 H 下的最低延迟。

观察结果。 表1中，两个工作负载各自分别有四种设置，我们观察到：

- 列表式方法（COOOL-list）在大多数设置中取得了最佳性能，并以较大优势击败 Bao

表1：单个数据集上的查询执行总延迟加速比（相对于 PostgreSQL）。每种工作负载下的最佳性能以粗体标示。

	JOB				TPC-H			
	adhoc-rand	adhoc-slow	repeat-rand	repeat-slow	adhoc-rand	adhoc-slow	repeat-rand	repeat-slow
Bao	1.07	0.91	3.02	1.37	5.36	1.17	5.28	4.73
COOOL-list	**1.35**	**1.46**	3.01	**1.57**	**6.09**	3.63	**5.33**	5.55
COOOL-pair	1.30	1.36	**3.47**	1.56	3.86	**3.86**	5.28	**5.56**

表 2：Bao 和 COOOL 与 PostgreSQL 相比的回归数量（单实例）

设置	Bao	COOOL-list	COOOL-pair
JOB重复-随机	24	17	**13**
JOB重复-慢	17	11	**8**
TPC-H重复-随机	3	**1**	**1**
TPC-H重复-慢	**1**	**1**	**1**

在几乎所有设置中，除了JOB 上的“repeat-rand”之外，COOOL 都略快于 Bao；在该设置下，COOOL 比 Bao 稍慢。

- 成对方法（COOOL-pair）在除TPC-H 上的“adhoc-rand”之外的几乎所有设置中也都优于 Bao。它在三个设置中取得了最佳性能（JOB 上的“repeat- rand”，以及 TPC-H上的“adhoc-slow”和“repeat-slow”）。在大多数设置中，COOOL-pair 和 COOOL-list 的性能相近。
- 在这些设置下，Bao 的性能都不是最好的。对于 JOB 上的“adhoc-slow”，Bao 的总执行延迟甚至出现了回退，这意味着其运行速度低于 PostgreSQL 优化器本身。

结合图 3，我们得出以下观察结果：

- Bao 在某些“slow”设置下显著差于 COOOL（在 JOB 上，“repeat-slow”中的20c；在 TPC-H上，“repeat-slow”中的模板 9）。“slow”设置显然比“rand”设置对 ML 模型更具挑战性，而 COOOL 明显优于 Bao。
- 在 JOB 和 TPC-H 数据的“repeat-rand”场景中，COOOL 和 Bao 都接近最优。
- 有时，Bao 或 COOOL 在少量查询上的表现不如 PostgreSQL，这对于机器学习模型而言是正常的。不过，在大多数设置下，COOOL 模型发生性能退化的频率低于 Bao 模型，而在这些设置下，COOOL-pair 的查询性能退化次数最少。详情见表 2。

总而言之，我们观察到，相较于Bao，COOOL 在加速查询执行总时长、缓解单个查询的性能退化以及优化慢查询方面具有优势和竞争力。此外，尽管 COOOL-list 在查询执行总延迟加速比方面优于COOOL- pair，但其在避免单个查询性能退化方面表现不佳。

5.3 工作负载迁移研究（RQ2）

众所周知，针对某个实例优化的模型在另一个工作负载上的表现可能很差，因为它无法从未见过的工作负载中学习模式。尽管大多数机器学习模型都依赖特定模式，这使得在另一工作负载上建立已训练的模型不可行，但针对与模式无关的模型，鲜有实验研究。本文直观地展示了如何将针对某个实例优化的模型直接迁移到另一个工作负载。具体而言，我们在源工作负载上训练一个模型，然后测试其在另一个工作负载（即目标工作负载）上的表现（源→目标）。

我们开展了在 JOB 上进行训练并将模型迁移到 TPC-H（JOB→TPC-H）的实验。同时，我们在 TPC-H 上训练模型，并研究其在 JOB（TPC-H→JOB）上的性能。在这一直接迁移研究中，我们还与实例优化实验一样，探究了“adhoc”和“repeat”场景，以及“rand”和“slow”设置。为了进行直观比较，我们在源工作负载的训练集上训练模型，并展示其在目标工作负载测试集上的性能。

总体查询执行速度提升如表 4 所示，我们有以下观察。

- 与相应的实例优化模型相比，各模型之间的性能差异并不稳定，而且大多数设置下性能都有所下降。我们的实验表明，即使模型与模式无关，将在一个工作负载上训练的模型直接应用于另一个工作负载，也无法获得良好的性能。

- 与针对实例优化的模型相比，大多数在 TPC-H 上训练的模型在 JOB 上的表现更差。相比之下，在“TPC-H adhoc”设置下性能有所提升，尤其是在“TPC-H adhoc-slow”上。我们可以得出结论，JOB 中的数据可能有助于提升 TPC-H 的表现，尤其是慢查询的性能。我们将在第 5.5.2 节从表征学习的角度进行更深入的分析。

- 我们在表3中展示了两个工作负载的计划树节点统计和深度。一方面，这表明 JOB 比 TPC-H 更复杂，因此，直接迁移在 JOB 上学习的模型可能有助于优化 TPC-H 中的查询。另一方面，这说明不同数据集中的数据分布会影响机器学习模型在工作负载迁移中的性能。因为“TPCH repeat”设置下的实验结果表明，在 JOB 的模板上学习的模型无法达到在 TPC-H 模板上学习的模型的相同性能。

表3：两个工作负载的总体计划树统计。

工作负载	最大节点数	平均节点数	最大深度	平均深度
JOB	72	23.6	36	12.0
TPC-H	35	14.3	20	9.6

根据这些观察，引入 JOB 数据的性能优于在 TPC-H“adhoc”设置下训练的模型。我们可以得出结论：对于给定的工作负载，使用来自同一工作负载的训练数据未必能获得令人满意的性能。因此，如何利用其他工作负载的数据，已成为查询优化模型面临的一个新兴问题。

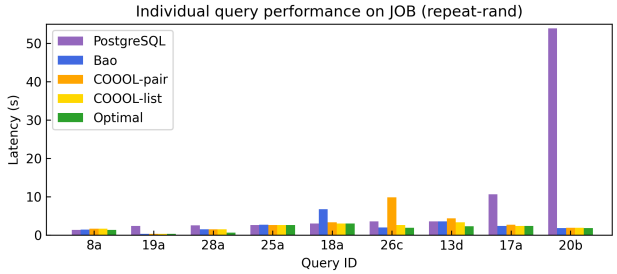
5.4 统一模型性能（RQ3）

本节重点关注使用 JOB和 TPC-H数据集训练的模型的性能。与上一节类似，我们针对每个数据集考虑四种场景：“adhoc-rand”、“adhoc-slow”、“repeat-rand”和“repeat-slow”。对于每种场景，将 JOB 和 TPC-H 的训练数据合并为新的训练集，并分别在 JOB 和 TPC-H 测试集上评估模型。

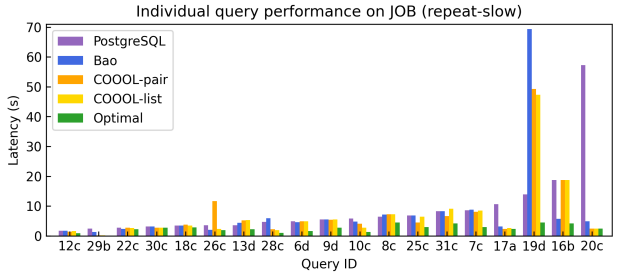
总体结果汇总于表5。我们观察到：

- COOOL-pair 在所有“临时”设置下表现最佳，而 COOOL-list 在所有“重复”设置下表现最佳。它们的整体表现相近，并且在几乎所有场景中都优于 Bao。
- COOOL-pair在使用JOB和TPC-H数据集训练模型（统一模型）时，相比表1获得了最大的性能提升，尤其是在 TPC-H上，统一模型在四种场景中的三种场景中优于单实例模型。
- 不同的训练数据集是否有帮助？我们观察到，JOB 训练数据有助于提升模型在 TPC-H 测试数据上的性能，尤其是在“adhoc”场景下。而在大多数场景下，TPC-H 训练数据并不能帮助提升模型在 JOB 测试数据上的性能。我们认为，这是因为 JOB 查询比 TPC-H 查询更复杂（节点更多、深度更大，如表3所示）。

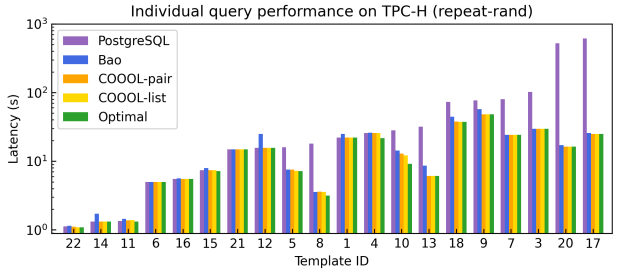
统一模型在“repeat”设置下的单个查询性能如图4所示，其中我们还展示了在 PostgreSQL 上执行延迟超过 1s 的查询。结合图4中的观察，我们可以得出以下结论：



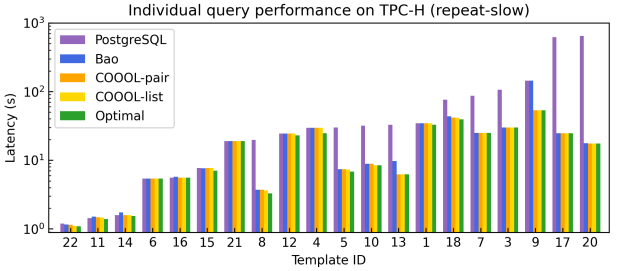
(a) JOB重复-随机



(b) JOB重复-慢



(c) TPC-H重复-随机



(d) TPC-H重复-慢

图4：统一模型的单个查询性能。

- 两种 COOOL方法的性能都接近最优。这印证了我们的观察：JOB训练数据有助于提升模型在 TPC-H 上的性能。
- 另一方面，TPC-H 训练数据损害了模型在 JOB 测试数据上的性能。例如，在图4b中，Bao 和 COOOL 方法在查询19d上的性能都很差。
- 在执行时间回归次数方面，我们将结果列于表6中。COOOL-pair和 COOOL-list 的表现均优于 Bao，且在这些设置下，COOOL-list 的查询回归次数最低。

模型性能总结。 不同数据集中的不同数据分布给统一查询优化模型带来了挑战，但 COOOL 模型能够缓解这一问题，从而更好地训练统一模型；与最先进的回归方法相比，该模型在提升总查询执行速度、缓解单个查询性能退化以及优化慢查询方面表现更佳。我们的实验显示，所提出的模型具有显著优势。在单实例场景下，COOOL-list 在总查询执行速度加速方面表现最佳，而 COOOL-pair 在减少单个查询回归数量方面表现最佳。当一个统一

表 4：直接迁移相对于 PostgreSQL 在目标工作负载上的查询执行总延迟加速比，其中 ↑表示与相应的实例优化模型相比性能有所提升。

	TPC-H → JOB				JOB →TPC-H			
	adhoc-rand	adhoc-slow	repeat-rand	repeat-slow	adhoc-rand	adhoc-slow	repeat-rand	repeat-slow
Bao	**1.07**	**1.19**↑	**1.07**	0.69	**6.35**↑	1.56↑	**1.64**	1.42
COOOL-list	0.97	0.93	0.92	0.82	0.85	**4.70**↑	**1.64**	1.70
COOOL-pair	0.96	1.18	0.89	**0.86**	5.90↑	4.60↑	1.48	**1.82**

表5：统一模型（在 JOB 和 TPC-H 数据上训练）相对于 PostgreSQL 的查询执行总延迟加速比。每种工作负载上的最佳性能以粗体显示。"↑"表示统一模型的性能优于在相应单个数据集上训练的模型。

	JOB				TPC-H			
	adhoc-rand	adhoc-slow	repeat-rand	repeat-slow	adhoc-rand	adhoc-slow	repeat-rand	repeat-slow
Bao	0.80	0.92	2.79	1.21	5.77↑	1.93	4.83	4.31
COOOL-list	1.00	1.21	**3.24**↑	**1.26**	6.59↑	2.91	**5.37**	**5.53**
COOOL-pair	**1.06**	**1.71**↑	2.90	1.21	**6.73**↑	**3.92**↑	5.34↑	5.51

表6：与 PostgreSQL（统一模型）相比，Bao 和 COOOL 的回归数量

设置	Bao	COOOL-list	COOOL-pair
JOB重复-随机	23	**12**	17
JOB重复-慢	20	**9**	11
TPC-H重复-随机	8	**1**	**1**
TPC-H重复-慢	4	**1**	**1**

表7："adhoc-slow"设置下达到收敛所需训练时间的比较

	JOB	TPC-H	统一
Bao	119.5秒	34.5秒	265.9秒
COOOL-list	167.0秒	159.8秒	317.1s
COOOL-pair	493.6s	222.8s	894.6s

模型使用多个数据集进行训练，在"repeat"设置下，COOOL-list的表现最佳，而在"adhoc"设置下，COOOL-pair 表现最佳，且查询执行性能没有总体回归。就稳定性而言，COOOL-pair 是三个模型中表现最佳的方法。

5.5 模型比较与分析（RQ4）

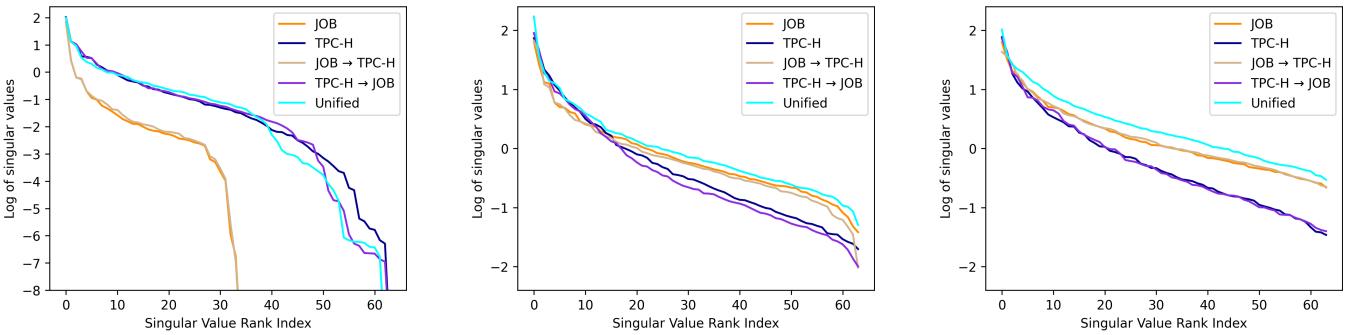
我们进行了大量实验并取得了令人鼓舞的结果。但排名策略如何影响模型训练尚不明确。本节旨在探讨为何两种 COOOL 模型优于 Bao，尤其是在统一模型训练方面。

在 COOOL 方法中，我们使用与 Bao 完全相同的计划表示模型，以确认这些改进是由...带来的。

我们的学习排序方法。我们在本节分析两个工作负载的"adhoc-slow"设置。

5.5.1 模型效率。我们比较了所提方法与最先进方法在空间和时间效率方面的表现。

• 空间复杂度。两个 COOOL 模型均采用与 Bao 结构和隐藏层大小相同的树卷积神经网络，它们的参数数量均为 132,353，即 529,412 字节。模型的存储开销约为 0.5MB，在实际应用中效率足够高。• 训练样本数量。由于本文涉及三种不同的策略，因此很难直接推导它们的时间复杂度。我们首先展示训练样本的数量。设 N 表示训练查询的数量。对于每个查询，H 中有 n 个候选提示集。Bao 的时间复杂度为 Nn。设 m_i表示给定 H 时每个查询对应的唯一查询计划数量，其中 $i \in \{1, 2, \ldots\ldots, N\}$，且 $m_i \leq n$。由于 COOOL-pair 使用完整的排序拆分策略，样本数量为 $\Theta(\sum_{i=1}^{N} m_i\ (m_i-1)\ 2) = O(Nn^2)$。由于 n 在实际应用中是给定的常数，因此复杂度可以接受。对于 COOOL-list，有 N个训练样本。• 训练时间开销。为了直观展示三种方法的时间效率，我们汇总了它们在"adhoc-slow"设置下达到收敛所需的平均训练时间，如表7所示。显然，COOOL 确实比 Bao 需要更多的训练时间才能收敛。此外，COOOL-pair 收敛所需的时间远多于 COOOL-list。• 结论。尽管 COOOL-pair 和 COOOL-list 的收敛时间比 Bao 更长，但它们的模型推理效率和参数数量均与 Bao 相同。因此，我们可以得出结论：COOOL 是一种实用的方法。



(a) Bao (b) COOOL-pair (c) COOOL-list

图 5：在单实例、工作负载迁移和统一模型场景下，adhoc-slow设置中两个工作负载的 Bao 和 COOOL 计划嵌入空间的奇异值谱。

5.5.2 从表征学习的角度看计划树嵌入。基于回归的方法通常会根据归一化方法将模型输出域限制在某个区间内。相比之下，基于排序的方法不会限制模型输出的绝对值，这可能有助于模型学习更好的计划表征。本节旨在从表征学习的角度，深入探讨 COOOL 优于 Bao 的原因。

令一个 h维向量 $\mathbf{z_i}$ 表示查询计划 i的嵌入。我们计算由模型获得的所有计划嵌入的协方差矩阵 $C \in \mathrm{R}^{h\times h}$。形式化地，

$$C = \frac{1}{M}\sum_{i=1}^{M}(\mathbf{z_i}-\bar{\mathbf{z}})(\mathbf{z_i}-\bar{\mathbf{z}})^{\top},$$

其中， M是计划的数量，$\mathbf{z}=\frac{1}{M}\sum_{i=1}^{M}\mathbf{z_i}$。然后，我们对 C $C = USV^{\top}$ S =进行奇异值分解，s.t.，其中( )。随后，我们可以按排序顺序和对数刻度 $diag\ \sigma^k$（$\lg(\sigma^k)$）获得奇异值谱。我们描绘了在“adhoc-slow”设置下、不同场景中学习得到的三个模型的计划嵌入空间的奇异值谱，如图 5 所示。我们有以下观察结果。

- 在图5a中，我们观察到所有任务的奇异值在对数尺度 $\lg(\sigma^k)$ 上都显著下降。谱曲线趋近于零（小于 1e-7），这表明在 Bao 的每种实验场景中，其嵌入空间都发生了维度坍缩 [9]；而在 JOB 上训练的模型甚至有一半的计划嵌入空间发生了坍缩。维度坍缩导致 Bao 的计划嵌入向量仅覆盖一个低维子空间，这可能会损害模型的表征能力，进而降低性能。此外，在不同数据集上的单实例场景中，模型发生坍缩的维度数量各不相同。根据工作负载迁移场景的曲线，我们观察到目标工作负载的计划嵌入空间与源工作负载覆盖相同数量的维度。因此，我们可以得出结论：嵌入空间由训练数据决定。

- 在图 5b 和图 5c 中，COOOL 方法的奇异值没有骤降，这意味着其计划嵌入空间没有发生坍缩。Bao 和 COOOL 的计划嵌入方法相同，这一比较表明，从回归框架和排序策略中学习到的嵌入空间存在显著差异。

- Bao 统一模型的谱更接近从 TPC-H 数据中学到的表示谱，而 COOOL-pair 和 COOOL-list 的谱则更接近从 JOB 数据中学到的表示谱，而非从 TPC-H 数据中学到的表示谱。事实上，TPC-H 的训练样本数量多于 JOB，而 JOB 比 TPC-H 更复杂（见表3）。在多个数据集上学习时，Bao 的性能受样本数量更多的数据集影响更大。相比之下，两种 COOOL 方法能够从复杂的查询计划中学习。

- 奇异值与协方差矩阵中的潜在信息有关。从这一角度来看，我们可以得出三个结论。首先，在不同设置下，Bao 的整体谱在绝对奇异值方面都低于 COOOL，这表明 Bao 的表征能力不如 COOOL。其次，在大多数维度上，Bao 的统一模型曲线并非最高，而两个 COOOL 模型的情况恰恰相反。这可能说明，排序策略能够帮助模型从两个工作负载中学习潜在模式，而回归框架则不能。第三，COOOL-pair 和 COOOL-list 曲线的取值范围和趋势有所不同，这可能导致这两个模型在不同场景下表现不同。

- 维度坍缩问题可能不会对单实例优化造成太大影响，因为嵌入空间是根据单个工作负载学习得到的，而且模型收敛时，潜在的坍缩维度是固定的。由于不同数据集中的坍缩维度可能不同，因此可能出现正交情况，导致在一个数据集中学习到的子空间表示在另一个数据集中成为噪声，进而造成性能不稳定。因此，这可能会限制机器学习查询优化模型的可扩展性。

总而言之，计划树嵌入空间中的维度坍缩，是机器学习查询优化模型在维持统一模型、从不同工作负载中学习时面临的一项挑战。COOOL 的两个变体与 Bao 使用完全相同的模型，但在几乎所有场景和设置下都持续优于 Bao。我们从查询计划表示的角度，提供了一些关于 COOOL 方法为何优于 Bao 的见解。这或许能为研究人员进一步提升机器学习查询优化模型的性能提供指引。

6 相关工作

6.1 查询优化的机器学习

传统的基于代价的查询优化器旨在选择估计代价最低的候选计划，其中代价表示执行延迟或其他用户定义的资源消耗指标。基于代价的优化器中提出了各种技术 [4, 5, 12], , 例如草图、直方图、概率模型等。此外，它们基于不同的假设（例如，属性值独立性 [3], 、均匀性 [11], 、数据独立性 [6]），而当这些假设无法满足时，这些技术通常会退而采用有根据的猜测 [30]。传统优化器已经过数十年的研究 [31], , 重点采用人工设计的方法和启发式方法来预测代价、估计基数并生成计划 [10]。然而，启发式方法的效果取决于数据分布，基数估计和成本建模已被证明难以解决 [21]。因此，基于代价的优化器可能会生成性能较差的计划。

因此，数据库领域一直在尝试应用机器学习（ML）技术来解决这些问题。例如，一些研究引入强化学习（RL）和蒙特卡洛树搜索（MCTS）来优化连接顺序选择任务 [16, 24, 34, 45], , 另一些研究则使用神经网络来完成成本建模和基数估计 [19, 32, 41, 42]。这些工作表明，精心设计的机器学习模型可以提升查询优化中部分组件的性能。然而，它们可能无法提升优化器的性能，因为它们都没有证明，单个组件的性能提升确实能够带来更优的查询计划 [18]。

近年来，一些研究尝试构建用于查询优化的端到端ML 模型 [23, 25, 40]。它们都使用 TCNN预测查询计划的代价/延迟，并利用深度强化学习框架训练模型。尽管与以往使用ML模型替代优化器部分组件的研究相比，这些研究取得了改进，但对优化器的评估应从多个维度进行衡量 [35], , 其中最受关注的部分包括 实用性、与数据/模式无关、可解释性、性能、可扩展性等。Neo [25] 从表中学习行向量嵌入，因此需要为不同数据维护行向量嵌入。Balsa [40] 由于自身依赖，无法处理高级 SQL 特性（例如子查询），这限制了其可扩展性。此外，与 Bao [23]相比，Neo 和 Balsa的实用性较低。Bao引入了一种新颖的端到端查询优化方法，在现有优化器之上推荐针对每个查询的提示。SQL 提示可以限制现有优化器的搜索空间，并且每组提示都对应一个计划，这使其在实际场景中具有实用性。

6.2 学习排序

学习排序（LTR）是一个活跃的研究主题 [43] ，在信息检索 [20], 、元搜索引擎 [7], 、推荐系统 [13], 、偏好学习 [46], 等领域有许多应用。搜索引擎或推荐系统通常需要对数量庞大且可变的项目（例如网页、电影）进行排序。SQL 提示推荐问题更类似于偏好学习问题：对于每个用户（在我们的场景中为 SQL 查询），需要对数量相对较少且固定的项目（在我们的场景中为 SQL 提示）进行排序。Plackett-Luce 模型 [22, 28], 后来被用作信息检索中的列表式损失函数 [20], 以及分类任务中的 Softmax 函数 [8],, 是偏好学习中最流行的模型之一。它是一种列表式模型，但可以通过将完整排名拆分为成对比较，并最小化成对损失函数，利用成对方法 [14, 47], 高效地进行学习。与 [14] 和 [47], 类似，考虑到其简洁性，我们最大化边缘成对似然。

7 结论

本文提出了COOOL，该方法利用学习排序（LTR）技术预测查询计划的成本顺序，与数据库管理系统（DBMS）协作，为查询优化推荐 SQL 提示。COOOL 具有成对方法和列表式方法，二者均能提升整体查询执行速度，尤其是在慢查询上，并缓解单个查询的查询性能退化。此外，我们可以通过所提出的方法维护一个统一模型，以改进查询计划选择；同时，我们还从表征学习的角度阐释了 COOOL-list 和 COOOL-pair 方法为何都优于 Bao。

COOOL 在两个数据集上维护统一的端到端查询优化机器学习模型方面向前迈进了一步，其深入分析可为大规模预训练查询优化模型提供初步基础。尽管 COOOL 方法前景可观，但它们无法估计推荐查询计划的代价，也无法对两个查询计划的代价进行定量比较。

未来，我们计划研究用于候选计划排序的评估指标，这些候选计划在查询优化中的执行延迟可能相差多个数量级，从而促进最先进的学习排序技术的引入。近年来，学习排序仍是一个活跃的研究领域，因此查询优化中仍有大量未来工作值得探索。此外，开发大规模预训练查询优化模型，并准确地对不同查询计划的代价进行定量比较，也是具有挑战性的未来方向。

参考文献

[1] Mert Akdere、Ugur Çetintemel、Matteo Riondato、EliUpfal 和Stanley BZdonik。2012年。基于学习的查询性能建模与预测。载于 *2012*年 *IEEE* 第*28*届数据工程国际会议。IEEE，390–401。[2] Hossein Azari Soufiani、William Chen、David CParkes和 Lirong Xia。2013年。用于排序聚合的广义矩方法。神经信息处理系统进展 26（2013年）。[3] StavrosChristodoulakis。1984年。数据库性能评估中某些假设的影响。*ACM Transactions onDatabase Systems*（*TODS*）9，2（1984年），163–186。[4] GrahamCormode 和 MinosGarofalakis。2010年。概率数据上的直方图和小波。*IEEE* 知识与数据工程汇刊22，8（2010年），1142–1157。[5] Graham Cormode、Minos Garofalakis、PeterJ Haas、Chris Jermaine 等。2011年。海量数据概要：样本、直方图、小波、草图。*Foundations and Trends® in Databases* 4，1–3（2011年），1–294。

[6] Amol Deshpande、Minos Garofalakis 和 Rajeev Rastogi。2001年。独立性有益：用于高维数据的基于依赖关系的直方图概要。ACM SIGMOD Record 30，2（2001年），199–210。[7] Cynthia Dwork、Ravi Kumar、Moni Naor 和 Dandapani Sivakumar。2001 年。面向网络的排序聚合方法。载于《第 10 届万维网国际会议论文集》。613–622。[8] Bolin Gao 和 Lacra Pavel。2017年。Softmax 函数的性质及其在博弈论与强化学习中的应用。arXiv 预印本 arXiv:1704.00805（2017年）。[9] Tianyu Hua、Wenxiao Wang、Zihui Xue、Sucheng Ren、Yue Wang 和 Hang Zhao。2021年。自监督学习中的特征去相关。载于《2021年 IEEE/CVF 国际计算机视觉会议，ICCV 2021》，加拿大魁北克省蒙特利尔，2021年10月10日至17日。IEEE，9578–9588。https://doi.org/10.1109/ICCV48922.2021。00946[10] Yannis E Ioannidis。1996年。查询优化。ACM Computing Surveys（CSUR）28，1（1996年），121–123。[11] Yannis E Ioannidis 和 Stavros Christodoulakis。1993年。用于限制连接结果规模最坏情况误差传播的最优直方图。ACM Transactions on Database Systems（TODS）18，4（1993年），709–748。[12] Yannis E Ioannidis 和 Viswanath Poosala。1995年。平衡直方图最优性与实用性，以估计查询结果规模。ACM SIGMOD Record 24，2（1995年），233–244。[13] Alexandros Karatzoglou、Linas Baltrunas 和 Yue Shi。2013年。推荐系统的学习排序。载于《第7届 ACM 推荐系统会议论文集》。493–494。[14] Ashish Khetan 和 Sewoong Oh。2016年。用于高效排名聚合的数据驱动的排名拆解。载于《国际机器学习会议》。PMLR，89–98。[15] Diederik P. Kingma 和 Jimmy Ba。2015年。Adam：一种随机优化方法。载于《第3届国际学习表征会议，ICLR 2015》，美国加利福尼亚州圣迭戈，2015年5月7日至9日，会议论文集。[16] Sanjay Krishnan、Zongheng Yang、Ken Goldberg、Joseph Hellerstein 和 Ion Stoica。2018年。利用深度强化学习优化连接查询。arXiv 预印本 arXiv:1808.03196（2018年）。[17] Viktor Leis、Andrey Gubichev、Atanas Mirchev、Peter Boncz、Alfons Kemper 和 Thomas Neumann。2015年。查询优化器到底有多好？Proceedings of the VLDB Endowment 9，3（2015年），204–215。[18] Viktor Leis、Bernhard Radke、Andrey Gubichev、Atanas Mirchev、Peter Boncz、Alfons Kemper 和 Thomas Neumann。2018年。透过棱镜审视查询优化，以及我们运行连接顺序基准测试后的发现。The VLDB Journal 27（2018年），643–668。[19] Henry Liu、Mingbin Xu、Ziting Yu、Vincent Corvinelli 和 Calisto Zuzarte。2015年。使用神经网络进行基数估计。载于《第25届计算机科学与软件工程年度国际会议论文集》。53–59。[20] Tie-Yan Liu。2009年。信息检索的学习排序。Foundations and Trends in Information Retrieval 3，3（2009年），225–331。[21] Guy Lohman。2014年。查询优化是一个“已解决”的问题吗？载于《数据库查询优化研讨会论文集》，第13卷。俄勒冈研究生中心计算机科学技术报告，10。[22] R Duncan Luce。1959年。个体选择行为。[23] Ryan Marcus、Parimarjan Negi、Hongzi Mao、Nesime Tatbul、Mohammad Alizadeh 和 Tim Kraska。2022年。Bao：让基于学习的查询优化变得实用。ACM SIGMOD Record 51，1（2022年），6–13。[24] Ryan Marcus 和 Olga Papaemmanouil。2018年。用于连接顺序枚举的深度强化学习。载于《首届数据管理人工智能技术应用国际研讨会论文集》。1–4。[25] Ryan C. Marcus、Parimarjan Negi、Hongzi Mao、Chi Zhang、Mohammad Alizadeh、Tim Kraska、Olga Papaemmanouil 和 Nesime Tatbul。2019年。Neo：一个基于学习的查询优化器。Proc. VLDB Endow. 12，11（2019年），1705–1718。https://doi.org/10.14778/3342263.3342644[26] Tom M Mitchell。1980年。学习泛化时需要偏置。Citeseer。[27] Lili Mou、Ge Li、Lu Zhang、Tao Wang 和 Zhi Jin。2016年。用于编程语言处理的树结构卷积神经网络。载于《第三十届AAAI人工智能会议。[28] Robin L Plackett。1975年。排列分析。英国皇家统计学会会刊C辑：应用统计 24, 2（1975年），193–202。[29] Meikel Poess 和 Chris Floyd。2000年。面向决策支持和网络商务的新 TPC 基准测试。ACM SIGMOD Record 29, 4（2000年），64–71。[30] Viswanath Poosala 和 Yannis E Ioannidis。1997年。不依赖属性值独立性假设的选择率估计。载于 VLDB，第 97 卷。486–495。[31] P Griffiths Selinger、Morton M Astrahan、Donald D Chamberlin、Raymond A Lorie 和 Thomas G Price。1979年。关系数据库管理系统中的访问路径选择。载于 1979年 ACM SIGMOD数据管理国际会议论文集。23–34。[32] Ji Sun 和 李国良。2019年。一种端到端学习型成本估算器。Proc. VLDB Endow. 13, 3 2019 307 319 10 14778 3368289 3368296 ( ), – 。[33] 1933 https://doi.org/ 。/ 。William R Thompson。. 关于基于两组样本证据，一个未知概率大于另一个未知概率的似然。Biometrika 25, 3-4（1933年），285–294。[34] Immanuel Trummer、Junxiong Wang、Ziyun Wei、Deepak Maram、Samuel Mose- ley、Saehan Jo、Joseph Antonakakis 和 Ankush Rayabhari。2021年。Skinnerdb：通过强化学习实现遗憾有界查询评估。ACM Transactions on Database Systems（TODS）46, 3（2021年），1–45。[35] Dimitris Tsesmelis 和 Alkis Simitsis。2022年。学习时代的数据库优化器。载于 2022 IEEE 第 38 届国际数据工程会议（ICDE）。IEEE，3213–3216。[36] Francesco Ventura、Zoi Kaoudi、Jorge Arnulfo Quiané-Ruiz 和 Volker Markl。2021年。拓展训练限制！为基于机器学习的数据管理生成训练数据。载于 2021年数据管理国际会议论文集。1865–1878。[37] 吴文涛、Yun Chi、Shenghuo Zhu、Junichi Tatemura、Hakan Hacigümüs 和 Jeffrey F Naughton。2013年。预测查询执行时间：优化器成本模型真的无法使用吗？载于 2013 IEEE 第 29 届国际数据工程会议（ICDE）。IEEE，1081–1092。[38] Fen Xia、Tie-Yan Liu、Jue Wang、Wensheng Zhang 和 李航。2008年。学习排序的列表式方法：理论与算法。载于第 25 届国际机器学习会议论文集。1192–1199。[39] Bing Xu、Naiyan Wang、陈天奇和李沐。2015年。卷积网络中修正激活函数的实证评估。arXiv 预印本 arXiv:1505.00853（2015年）。[40] Zongheng Yang、Wei-Lin Chiang、Sifei Luan、Gautam Mittal、Michael Luo 和 Ion Stoica。2022年。Balsa：无需专家示范即可学习查询优化器。载于 SIGMOD '22：数据管理国际会议，费城，美国宾夕法尼亚州，2022年6月12日至17日，Zachary Ives、Angela Bonifati 和 Amr El Abbadi（编）。ACM，931–944。https://doi.org/10.1145/3514221.3517885[41] Zongheng Yang、Amog Kamsetty、Sifei Luan、Eric Liang、Yan Duan、Xi Chen 和 Ion Stoica。2020年。NeuroCard：适用于所有表的单一基数估算器。Proceedings of the VLDB Endowment 14, 1（2020年），61–73。[42] Zongheng Yang、Eric Liang、Amog Kamsetty、Chenggang Wu、Yan Duan、Xi Chen、Pieter Abbeel、Joseph M Hellerstein、Sanjay Krishnan 和 Ion Stoica。2019年。深度无监督基数估计。arXiv 预印本 arXiv:1905.04278（2019年）。[43] 于海涛、Rajesh Piryani、Adam Jatowt、Ryo Inagaki、Hideo Joho 和 Kyoung- Sook Kim。2023年。对抗式学习排序的深入研究。信息检索杂志 26, 1（2023年），1。[44] Daokun Zhang、Jie Yin、Xingquan Zhu 和 Chengqi Zhang。2018年。网络表示学习：综述。IEEE大数据汇刊 6, 1（2018年），3–28。[45] Ji Zhang。2020年。AlphaJoin：AlphaGo 式的连接顺序选择。载于 PVLDB-PhD。[46] Zhibing Zhao、Ao Liu 和 Lirong Xia。2022年。利用不完整偏好中的特征学习随机效用模型混合。arXiv 预印本 arXiv:2006.03869（2022年）。[47] Zhibing Zhao 和 Lirong Xia。2018年。随机效用模型的复合边际似然方法。载于国际机器学习会议。PMLR，5922–5931。