Sema：一种用于基于大语言模型的语义查询处理的高性能系统

Kangkang Qi 北京理工大学
qikangkang0126@gmail.com

Dongyang Xie 武汉大学
xdy1122310013
@163.com

Wenbo Li 武汉大学
li
wenbo82@whu.edu.cn

Hao Zhang 香港中文大学
zhanghaowuda12@gmail.com

Yuanyuan Zhu 武
汉大学
yyzhu@whu.edu.cn

Jeffrey Xu Yu 香港科技大学
（广州）
jeffreyxuyu@hkust-gz.edu.cn

Kangfei Zhao*北京理工
大学 zkf1105@gmail.com

arXiv:2603.11622v1 [cs.DB] 12 Mar 2026

摘要

将大语言模型集成到数据分析中，为对大规模结构化与非结构化数据进行推理释放了强大能力。然而，现有系统通常要么依赖 DataFrame 原语，而这些原语缺乏现代数据库管理系统的高效执行基础设施；要么依赖 SQL 用户定义函数（用户定义函数），这些函数将语义逻辑与查询优化器隔离开来，并使用户承担实现上的复杂性。由大语言模型驱动的语义算子还因大语言模型调用的高成本和非确定性而带来新的挑战，传统的优化规则和成本模型无法适用于其优化。

为弥合这些差距，我们提出 Sema，一个基于 DuckDB 构建的高性能语义查询引擎，它将大语言模型驱动的语义算子视为一等公民。Sema 引入了 SemaSQL，一种声明式方言，允许用户将自然语言表达式无缝注入标准 SQL 子句，从而实现端到端优化与执行。在逻辑层，Sema 的优化器压缩自然语言表达式，并从语义算子中推导关系约束。在运行时，Sema 采用自适应查询执行（AQE），动态重排算子、融合语义操作并应用提示批处理。该方法在准确率约束下寻求一条平衡令牌消耗与延迟的帕累托最优执行路径。我们在涵盖分类、摘要和提取任务的 20 个语义查询上评估了 Sema。实验结果表明，Sema 相对于三个基线系统实现了 2 – 10× 的加速，同时取得了具有竞争力的结果质量。

1 引言

现代语言模型（大语言模型）强大的语义推理能力，正在为基于人工智能的分析开辟新的机遇，使其能够处理结构化和非结构化数据。科学、医疗保健、商业和工业领域的应用程序日益需要对大型数据集进行复杂推理，例如总结最新研究、从患者记录中提取生物医学洞见，或大规模分析企业会议文字记录。要实现这些能力，分析系统需要以高效且易用的方式编排复杂的批量语义处理，有望革新现有数据库系统。这一愿景引出了两个根本性问题：开发者应如何表达语义查询？开发者又应如何设计数据分析系统，以实现高效率和高准确率？

近期的一些系统开始探索面向结构化 [18, 36] 和非结构化数据 [5, 48, 50]的大语言模型驱动的语义查询处理。DocETL [48], 、ZenDB [34],、DocWanger [47],、Aryn [5] 和 Unify [50]等系统支持以扩展 SQL、声明式操作或自然语言表达的语义查询。这些系统不仅利用大语言模型理解查询，还通过查询规划 [5], 、重写、提示优化和操作分解 [50]等技术优化执行。Palimpzest [35] 及其优化器 Abacus [46] 专注于通过逻辑和物理优化降低大语言模型查询延迟和成本，包括帕累托高效规划。Lotus [42]针对表格数据构建了基于大语言模型的语义操作，并提出了适用于高成本算子的高效执行算法。近期的基准测试 [4, 18] 指出了现有数据库在支持语义查询处理方面面临的主要挑战，例如确保结构化输出、支持批处理以及实现有效的查询规划。

当前系统通常以两种不同方式表达语义查询：DataFrame 原语 [35, 42, 46] 和 SQL 用户定义函数（UDF） [4, 18, 36], ，这两种方式都与底层系统设计高度交织。基于 DataFrame 原语构建的系统会定义自定义语义操作，并使用专用算法对其进行优化，但往往缺乏数据库管理系统高效的执行基础设施。因此，开发者不得不重新部署向量化和流水线并行执行、输入/输出以及内存管理等执行环境。另一方面，将语义算子集成为 SQL 用户定义函数虽然灵活性更高，却把实现、调试和维护复杂 AI 算子的负担交给了用户，并引发了对安全性、可靠性和效率的担忧。

随着语义查询需求不断增长，一组核心语义算子（如语义筛选、投影、连接和聚合）逐渐涌现，成为经典关系算子的一种由 AI 驱动的类比 [42, 50]。这一趋势自然引出了一个设想：我们能否将这些语义算子作为原生数据库算子高效支持，从而充分利用数据库管理系统的优化能力

*通讯作者

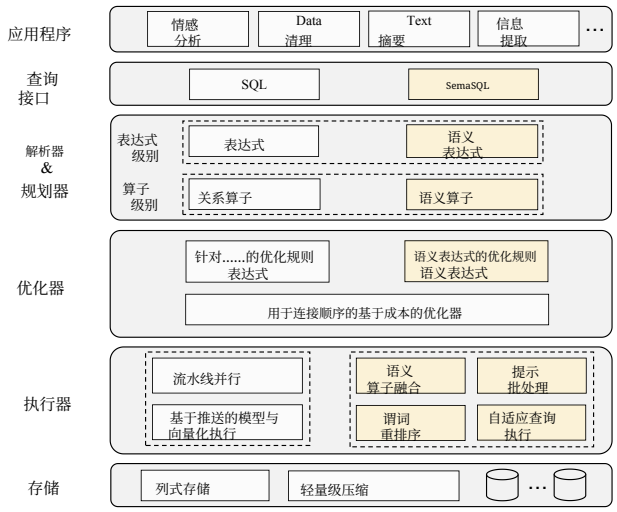

图 1：Sema 的系统架构

处理？然而，实现这一设想并非仅仅是添加一些实现层面的扩展。关系算子具有确定的语义、稳定的代数定律，以及可根据数据统计信息近似得出的成本模型；与之不同，语义算子依赖上下文和模型：其执行不仅取决于输入元组，还取决于自然语言（NL）表达、提示策略以及所调用的特定大语言模型。外部大语言模型服务环境中的大语言模型推理优化（例如 KV缓存、分页注意力和连续批处理）也会显著影响执行性能。对这些由大语言模型驱动的算子进行成本建模，已超出对复杂、CPU 密集型算子进行建模的范畴 [21]。这一转变打破了传统查询优化背后的关键假设：标准重写规则可能并不安全，选择率和成本也无法通过静态方式估算。这些挑战需要针对语义算子设计新的系统级优化。系统应将语义算子视为一等公民，并着力降低大语言模型调用的成本，因为它们主导着查询执行的性能。此外，语义算子还带来了多项优化目标方面的额外考量，包括查询准确率、延迟以及大语言模型调用的货币成本。

作为一种初步实现，我们设计并开发了一个名为 Sema 的原型系统，该系统以 DuckDB [43], 为后端，并注入了可调用本地或远程大语言模型的原生语义算子。基于向量化和流水线执行基础设施，Sema 的设计原则是降低大语言模型调用的成本。图 1 展示了 Sema 的架构；如图中所示，Sema 相应地扩展了 DuckDB 的查询语言接口、解析器和规划器、优化器以及执行器。为表达语义算子，Sema 引入了 SemaSQL，使用户能够将自然语言（NL）表达式插入 SQL 子句中。这些封装数据库列名的自然语言表达式传达了用户在语义谓词、排序或聚合条件方面的意图。为了高效且灵活地处理这些原生语义算子，我们在优化器中提出了逻辑上的算子内优化，用于压缩自然语言表达式并从语义算子推导出关系算子；其中，优化器借助辅助大语言模型来完成这些优化。在执行器中，Sema 集成了语义算子融合和提示批处理，以减少批量数据处理中的大语言模型调用次数。由于语义算子实例的统计模式难以在执行前刻画，Sema 采用自适应查询执行（AQE）对连续的语义算子进行重排序，并结合算子融合和批量提示。在查询准确率约束下，AQE 试图在大语言模型令牌消耗和查询延迟方面寻找帕累托最优执行路径。我们在来自 BIRD [32]的 3 类分析任务（分类、摘要和关键词提取）中的 20 个语义查询上，对 Sema 进行了全面评估。结果表明，与基于 DataFrame 的系统 Lotus 相比，Sema 实现了 2–4× 倍加速；与基于流水线执行的系统 Palimpzest 相比，实现了 2–10× 倍加速；同时，在保持与这两个系统相当的结果质量的情况下，显著优于基于 UDF 的系统 FlockMTL。本文的贡献总结如下。

- 我们设计并开发了 Sema，这是一种构建于 DuckDB 之上的高性能语义查询引擎，将由大语言模型驱动的语义算子视为一等查询计划算子，从而在列式、向量化数据库系统中实现端到端的语义查询优化与执行。
- 我们提出了 SemaSQL，这是一种声明式 SQL 方言，允许用户将自然语言表达式注入标准 SQL 子句中，以定义语义算子，从而实现语义计算与关系计算的无缝组合。
- 我们针对语义算子设计了专门的优化技术，以协同降低大语言模型调用的开销，包括逻辑级语义表达式优化，以及通过结合算子融合与提示批处理的自适应查询执行（AQE）实现的运行时优化。
- 我们设计了一个语义查询基准，其中包含针对 9 个数据集的 20 个查询，并对 Sema 进行了全面评估。结果表明，与基于 DataFrame 和基于 UDF 的系统相比，Sema 在查询效率方面显著更优，同时保持了具有竞争力的结果质量，展现出很高的易用性。

论文结构。本文后续内容安排如下：第 2 节介绍背景和研究动机；第 3 节介绍 Sema 系统的剧情简介；第 4 节和第 5 节分别详述其优化器和执行器；第 6 节阐述 Sema 的关键工程细节；第 7 节报告系统评估结果；第 8 节评价相关工作；第 9 节总结全文。

2 背景

2.1 语义算子

语义算子是对表格数据进行声明式转换的算子，其由作为谓词的自然语言表达式参数化。在本文中，我们考虑由大语言模型（LLM）驱动的语义算子。我们用 $T(A_1, \cdots, A_j)$ 或简称 T来表示一张表，其中 $A_1, \cdots, A_j$是 T的属性。$t \in T$表示表 T中的一个元组，而 $t(A_j)$ 表示元组 t中属性 A_j的值。M: T $\mapsto$ T 是一个预训练的大语言模型，它接受输入并在文本空间 T 中生成输出。Sema 从对语义查询处理的关系代数推广开始，

表1：语义算子

算子	表达式	描述	# 大语言模型调用
语义筛选	$\{t \mid t \in T \wedge M(t,e) = \text{正确}\}$	选择能够满足自然语言谓词的元组 e	$O(\|T\|)$
语义投影	$\{M(t,e) \mid t \in T\}$	投影由自然语言谓词指定的属性值 e	$O(\|T\|)$
语义连接	$\{(t_i,t_j) \mid t_i \in T_1 \wedge t_j \in T_2 \wedge M(t_i,t_j,e) = \text{True}\}$	根据连接键连接通过自然语言谓词的两个表 e	$O(\|T_1\| \cdot \|T_2\|)$
语义排序	$[t_1,\cdots,t_n], s.t. \forall [t_i,t_j] \Rightarrow M(t_i,t_j,e) = t_i \preceq t_j$	按照自然语言指定的标准对元组重新排序	$O(\|T\|^2)$
语义聚合	$M(\{t_1,\cdots,t_n\},e), t_1,\cdots,t_n \in T$	根据自然语言指定的聚合函数对元组进行聚合	$O(1)$

Lotus 中语义算子的一个核心集 [42]，而其他特定系统可能会提供这些算子之外的额外算子。

表1总结了我们在 Sema 中实现的五种语义算子，即语义筛选（*SemFilter*）、语义投影（*SemProj*）、语义连接（*SemJoin*）、语义排序（*SemOrderBy*）和语义聚合（*SemAgg*）。每种算子均由一个默认参考算法实现。对于 *SemFilter* 和 *SemProj*，其参考算法会扫描表 T 中的每个元组；系统要求大语言模型针对 *SemFilter* 返回 True 或 False，或针对 *SemProj*返回文本属性，两者分别需要 $O(|T|)$ 次大语言模型调用。对于 *SemJoin*，参考算法对两个表 T_1 和 T_2执行嵌套循环连接，并使用大语言模型评估一对元组是否能根据语义谓词进行连接，需要 $O(|T_1| \cdot |T_2|)$ 次大语言模型调用。*SemOrderBy* 的参考算法是一种选择排序算法，它根据语义比较准则比较 T中的两对元组，产生 $O(|T|^2)$ 次大语言模型比较。对于*SemAgg*，参考算法执行多对一归约，将 T中某个属性的所有值放入大语言模型提示中，并生成聚合结果。系统负责处理底层大语言模型有限的上下文限制，并可对值的子集进行分层聚合。与关系算子类似，同一个语义算子的求值可以通过不同算法实现；然而，在数据库引擎中支持基于大语言模型的语义算子并非易事，也不只是实现层面的问题。

2.2 语义算子处理

与关系算子不同，关系算子的行为建立在确定且可判定的语义、定义明确的代数定律以及稳定的成本模型之上，而语义算子本质上依赖于上下文和模型。其结果不仅取决于输入元组，还取决于自然语言表达式、提示策略以及所调用的具体大语言模型。因此，求值过程本质上具有不确定性，无法用单一的规范真值函数来刻画。这种语义转变也削弱了经典查询优化器赖以运作的基础。标准代数重写规则可能不再成立：例如，*SemFilter* 只有在严格条件下（如采用元组独立提示和零温度推理）才满足交换律，而结合律通常并不适用。对于 *SemProj* 和 *SemJoin*，交换律和结合律通常无法保持语义，因此传统的基于规则的重写可能不安全或不适用。最后，语义算子的成本特征主要受外部推理环境支配，也就是说，延迟、吞吐量和货币成本会随模型、提示、推理优化和缓存机制而变化；因此，仅凭数据库统计信息很难估算这些成本。综上，这些特性要求系统设计将语义算子视为一等公民，而非黑盒附加组件。

现有语义查询处理方案受到其执行环境的限制。基于轻量级库（如 Pandas 或自定义 DataFrame API）构建的实现提供了易于上手的编程模型，但缺少成熟的数据库能力，例如向量化执行、流水线并行和稳健的查询优化，因此在处理大规模混合分析工作负载时效率低下。相比之下，以数据库为中心的方案通常将大语言模型调用封装为用户定义函数（UDF）。虽然这种做法很方便，却使语义算子对优化器和执行器不透明，因而无法针对语义工作负载进行系统级优化。

针对这些不足，我们构建了 Sema，这是一个将语义算子直接集成到 DuckDB 中的原型系统。Sema 利用 DuckDB 所具备的高性能 OLAP 执行引擎、以表达式为中心的设计、向量化处理和流水线并行能力，实现大规模语义算子处理。通过将语义原语嵌入查询规划器、优化器和执行器，Sema 为探索语义特定优化提供了一个具体平台，包括表达式优化和自适应查询执行，旨在降低大语言模型调用的开销。在这项工作中，我们着重于系统设计，以及语义算子在关系引擎中的端到端集成。我们的目标并非全面优化每个算子，也不涵盖生产环境中的所有问题。以下内容留待未来工作：针对单个算子的专门算法改进、存储层优化、大语言模型路由与模型选择，以及针对外部推理的隐私、治理和容错综合机制。

3 系统概述

我们将概述 Sema，包括其查询接口（§3.1）和工作流程（§3.2）。图 1 展示了 Sema 的系统架构。

3.1 查询语法

Sema 引入了 SemaSQL，这是一种注入自然语言（NL）表达式的声明式语法，用于语义查询处理。SemaSQL 的设计遵循三项主要原则：透明性：与底层实现和优化隔离；灵活性：能够充分表达用户意图并具备较高效用；兼容性：对 SemaSQL 的扩展与 SQL 的语法标准及实现兼容。

SemaSQL 的一等公民是包含一个或多个占位符的自然语言表达式，表示为 *s*'$\cdots${}$\cdots$'。其中，{} 是一个占位符，用于标识表达式中对列/属性的提及，或属性的自然语言描述，并指示系统处理相应的表和属性。语义

算子通过将自然语言表达式注入 SELECT、WHERE、JOIN ON 等 SQL 子句中隐式实现。具体而言，*SemProj* 将自然语言表达式注入 SELECT，而 *SemFilter* 则分别将自然语言表达式注入 WHERE 或 HAVING 子句。对于*SemJoin*，我们将自然语言表达式注入 JOIN ON 子句，这也兼容其他连接类型，包括 OUTER JOIN、SEMI JOIN 等。对于 *SemAgg*，我们引入一个特殊的聚合函数 sem_agg() 来封装自然语言表达式，并且 sem_agg() 可以与标准 GROUP BY 子句结合使用。此外，用户可以使用 AS 为新的文本列重命名，例如由 *SemProj* 和 *SemAgg*生成的列；这些列还可以在其他自然语言表达式中进一步引用。图2展示了一个 SemaSQL 查询，其中 WHERE 子句包含两个 *SemFilter* 谓词。该查询对两个表执行关系连接，其中包含两个关系算子，以及两个作用于表 user_reviews 的译名为_评价属性上的 *SemFilter* 算子。在此，自然语言表达式用作语义谓词，其取值由大语言模型推断。更多示例查询见附录A图9-12。

SemaSQL 通过对自然语言表达式进行插值来支持语义算子，同时保留 SQL 符号逻辑的自然特性。Sema 还支持使用 SemaSQL 编写复杂语法，例如子查询、公用表表达式（CTE）和窗口函数等。与 NL2SQL 不同，SemaSQL 只进行最小限度的扩展，即在 SQL 上加入自然语言表达式，从而实现与关系数据库管理系统的完全兼容，并支持充分的语义推理。

3.2 系统工作流程

我们使用图2中的贯穿示例来说明 Sema 的端到端工作流程，如图3所示。该查询针对每个应用，统计满足两个语义谓词（“是有效的用户评论”和“是正面用户评论”）以及类别和类型上的标准关系谓词的评论数量。用户提交查询后，Sema 会解析 SemaSQL，并生成一棵抽象语法树（AST）和一个初始逻辑计划，其中自然语言表达式被识别为显式语义算子。此处，两个 *SemFilter* 算子成为一等计划节点，Sema 可以分别在逻辑层和物理层对其进行分析和转换。

首先，优化器中的逻辑自然语言表达式优化先于传统 SQL 重写执行，重点在于精简和重构算子内语义表达式。具体而言，Sema 采用冗余感知表达式压缩来缩短自然语言谓词并稳定提示。更重要的是，Sema 会为 *SemFilter* 执行谓词推导：它会择机从自然语言谓词中提取可完全或部分推导的符号条件（例如，‘translated_review !=“nan”’），并将其重写为关系过滤器，再将这些过滤器下推至经过剪枝的元组，从而避免不必要的大语言模型调用。这些表达式优化由采用思维链提示的辅助大语言模型指导。第 4 节将详细介绍这些作为特殊规则注入优化器、用于语义表达式的优化。

然后，在向量化、流水线并行执行引擎下，物理执行优化和自适应查询执行（AQE）针对的是语义算子的主要运行时成本。Sema 的执行器应用语义算子融合和提示批处理，以减少大语言模型调用的次数并提高吞吐量。因为

```
-- 计算类型为“免费”且类别为“ART_AND_DESIGN”的每个应用的有效用户评论数和正面用户评论数。SELECT ur.app，计数 (*) AS positive_and_valid_count FROM user_reviews AS ur INNER JOIN Play 商店 AS p ON ur.app =p.app WHERE p. 类别 = 'ART_AND_DESIGN ' AND p. 类型 =' 免费 ' AND s'{ translated_review } 是有效的用户评论 ' AND s'{ translated_review } 是正面用户评论 ' GROUP BY ur.app;
```

图 2：一个贯穿全文的语义查询示例

由于这些优化的效果以及*SemFilter*的选择率难以静态预测，Sema 为 *SemFilter* 算子引入 AQE。在运行时，执行器会对少量输入进行微执行，以收集谓词统计信息，探索多条候选执行路径，并在满足准确率容差的前提下，选择最能平衡延迟与令牌成本的路径。随后，执行器会使用所选路径大规模处理剩余数据。我们将在第 5 节介绍这些技术细节。

4 SEMA 的优化器

为与系统设计目标保持一致，Sema 的优化器着重于降低大语言模型调用的成本，同时不牺牲准确率。由于最终用户可能会在 SemaSQL 的语义算子中提交任意自然语言表达式，这些表达式本身往往冗长或含义模糊，其求值会给大语言模型推理带来额外开销。为解决这些问题，Sema 的优化器引入了专门的规则来优化自然语言表达式，包括表达式压缩和谓词演绎，并将这些规则逐步应用于查询计划。由于自然语言表达式是自由文本，传统的规则匹配无法全面捕捉压缩和演绎模式，因此我们采用大语言模型辅助优化。

自然语言表达式压缩。对于每个语义算子中的自然语言表达式，优化器首先使用带有思维链（CoT）提示 [51]和示例的轻量级大语言模型，对自然语言表达式进行压缩和优化。思维链提示包含分解后的指令，用于删除停用词、简化句子结构（例如将被动语态改为主动语态），以及将重复的代词指代合并为占位符；这些指令以多条规则的形式列出。我们还为每条规则加入示例，以说明其用途。此步骤对每个语义算子仅引入一次大语言模型调用，同时有望降低大语言模型推理成本并提升语义清晰度。

谓词推导。 表达式压缩后，优化器会寻找机会将语义过滤器推导为关系过滤器，即将*SemFilter* 中的全部或部分自然语言表达式重写为 SQL 谓词。完全推导可以绕过大语言模型调用，

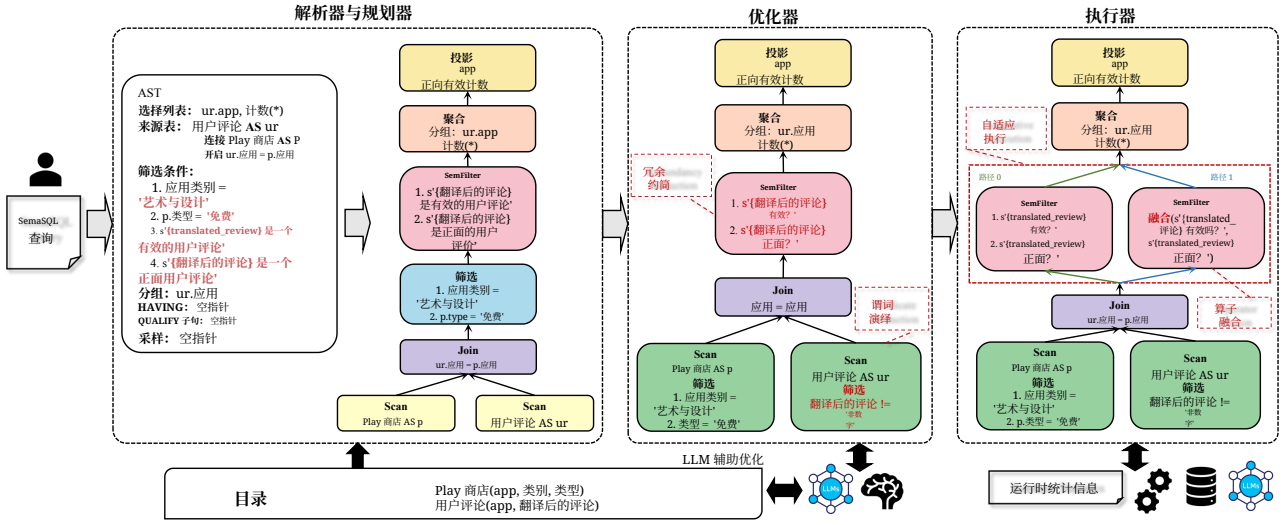

图 3：Sema 的系统工作流程

而部分推导则只对复合逻辑中的部分过滤谓词进行精准替换，将其替换为作为必要条件的 SQL 谓词，同时保留其余语义，并将其表示为更简洁的自然语言表达式。借助下推的 SQL 谓词，这种部分推导有望减少大语言模型调用。SemFilter 的谓词推导类似于关系查询处理中谓词简化的做法，类似思想也已被引入基于机器学习的谓词 [13]。值得一提的是，现有的自然语言转 SQL 方法 [15] 无法直接执行此任务。基于规则的方法完全依赖规则匹配进行 SQL 转换，因此不具备灵活推理语义表达式的能力。基于深度学习的方法依赖高质量的标注数据进行训练。基于 LLM 的方法有望推导自然语言谓词；然而，它们无法支持部分推导，即将 SQL 谓词转换为必要条件，也不具备相应的验证机制。

Sema 支持推导类别匹配、数值比较以及字符串上的 LIKE 操作谓词。首先，我们采用思维链提示，将数据库模式作为上下文输入，引导大语言模型适时识别可推导的谓词，并将其转换为字符串格式的 SQL 表达式。指令明确分解为多个步骤：首先从表达式中提取相关的表和列，判断查询是否具有客观意图（例如数值或字符串比较），以及能否映射为符号比较模式；随后判断其是否可推导，最后生成 SQL 语句。每个步骤中，我们都提供少样本示例，同时展示大小写变体、模糊匹配和数据类型一致性等特殊情况。为缓解大语言模型的幻觉问题，我们引入了前向—后向自我反思机制。一旦推导出语法有效的 SQL 语句，我们会通过另一个思维链提示要求大语言模型判断该 SQL 语句是否为原始自然语言表达式的必要条件。如果自我反思未能推导出结果，系统便会退回到原始的 *SemFilter*，以保证正确性。

示例 *4.1*。（表达式压缩与谓词推导）我们使用图2中的查询，其中包含两个作用于表 user_reviews 中属性 translated_view 的 *SemFilter* 算子。Sema 的解析器将查询转换为初始的未优化计划，如图3所示。在优化器中，两个自然语言表达式首先被压缩，分别形成两个较短的句子。由于占位符已包裹列名的精确匹配项，元实体解析可直接将其绑定到列对象。随后，谓词推导从带有表达式 ‘s{translated_review} is a valid user review’ 的 *SemFilter* 中提取出 SQL 谓词 ‘translated_review != “nan”’。此处，辅助大语言模型以数据库元数据为上下文接受提示；元数据表明，对于列 translated_review，约 41.8% 的条目为 ‘nan’。显然，‘nan’ 是无效评论，但用户可能不会注意到这一点。谓词推导会从 *SemFilter* 中自动识别出这一符号逻辑，并进一步下推推导出的 SQL 谓词以实现早期筛选，如图3所示，从而绕过大量大语言模型调用。

在 Sema 的优化器中，面向标准 SQL 的基于规则的优化之前会应用 LLM 辅助表达式优化。通过策略性地采用 LLM 辅助表达式优化，优化器提升了普通用户使用 SemaSQL 执行分析任务的便捷性，并增强了 Sema 的可靠性、准确率和效率。我们在附录中提供该算法，以展示表达式优化的整体流程，以及 Sema 用于谓词推导的提示。

5 执行优化

Sema 的执行器整合了专为语义算子设计的新型物理优化（第 5.1 节）和自适应查询执行（第 5.2 节）。

5.1 物理优化

语义算子融合。 在语义查询评估中，每个语义算子通常都会触发一次独立的大语言模型调用。

表2：Sema 中的算子融合

# 大语言模型调用	$\sigma_a \oplus \sigma_b$	$\Pi_a \oplus \Pi_b$	$\sigma_a \oplus \Pi_b$	$\Pi_a \oplus \sigma_b$
原始	$2\|T\|$	$2\|T\|$	$(1+s)\cdot\|T\|$	$(1+s)\cdot\|T\|$
融合后	$\|T\|$	$\|T\|$	$\|T\|$	$\|T\|$

尽管这种方式保留了模块化和算子独立性，但对于包含级联语义算子的复杂查询，中间数据会在多个算子之间反复物化和传输，从而导致较高的内存占用、令牌消耗和 API调用延迟。受关系算子 [40] 和线性代数算子 [9], 的算子融合启发，Sema 引入了语义算子融合，将连续的语义算子合并为一个算子。与融合传统算子时通过编译优化来综合生成代码不同，融合语义算子本质上是将它们的自然语言语义表达式合并为一个表达式，并对每个元组执行一次大语言模型推理。

如表2所示，Sema 支持在同一查询流水线中，对同一张表上的两个“单目”语义算子 *SemFilter* 和*SemProj* 进行四种模式的算子融合。我们从这四种模式入手实现语义算子融合，因为融合更多算子（同时包含单目和二元算子）需要解析并组合复杂语义，以确保评估准确性。对于*SemFilter*优先的算子，待融合的两个算子的自然语言表达式，其占位符必须指向同一个属性。对于*SemProj*优先的算子，重命名算子 AS 表示中间列，该列会被后续算子的表达式引用。一旦 Sema 识别出符合这些模式的情况，就会将两个连续算子的自然语言表达式拼接成一个自然语言表达式，并添加一条逐步处理的指令。

如表2所示，对于*SemProj*优先的模式，一次融合可节省 $|T|$ 次大语言模型调用；对于*SemFilter*优先的模式，则可节省 $s\cdot|T|$ 次大语言模型调用。其中，s是表 T中第一个*SemFilter*的选择率。

示例*5.1*。（语义算子融合）我们使用图2中的查询来说明算子融合。该查询包含两个*SemFilter*，它们引用了表 user_review 中同一列 translated_review。Sema 将两个压缩表达式 s’translated_review is valid’ 和 s’translated_review is positive’ 合并为一个表达式 ‘stranslated_review is valid and positive’，然后为该表达式生成一个新的*SemFilter*，以替换原来的两个算子。

提示批处理。 作为分析型系统，DuckDB 的向量化执行是 Sema 批量查询处理的基石。在此基础上，Sema 进一步通过提示批处理来提升语义算子的令牌利用率和吞吐量，即将多个元组合并到一个大语言模型提示 [33], 中，用于*SemFilter*、*SemProj*和*SemJoin*。

尽管减少了大语言模型调用，但算子融合和提示批处理打破了算子无关和元组无关的执行方式。一个潜在风险是，与逐次进行大语言模型调用相比，它们通常会导致推理结果不一致。考虑到这一问题，首先，对于提示批处理，执行器利用大语言模型API调用的结构化输出，将API调用的返回值指定为 JSON 数组。这确保一次批量提示的输出生成数量一致的结果。其次，在融合以 *SemFilter* 开头的模式时，Sema 会在组合提示中添加一条指令，要求大语言模型明确生成第一个算子的中间结果，以保持融合算子的评估一致性与原始算子流程一致。Sema 将算子融合限制为仅融合两个算子，将更多算子的融合留作后续工作。第三，对于 *SemFilter*，Sema 允许用户指定结果一致性的容忍度。自适应查询执行会在运行时根据这一用户偏好选择实际的执行计划，我们将在第 5.2 节详细阐述。

5.2 *SemFilter*的自适应查询执行

语义算子的行为取决于自然语言表达式和用于解释该表达式的特定大语言模型，因此选择率、延迟以及货币成本/令牌成本等关键计划指标难以静态预测。此外，减少大语言模型调用的优化（例如算子融合和批量提示）可能导致难以建模的准确率变化。为应对这些挑战，我们通过自适应查询执行（AQE）策略，将算子间重排序和融合等优化推迟到运行时。具体而言，Sema 的执行器会在多个候选物理计划（即执行路径）上评估少量数据，以收集运行时反馈，并选择一条最优路径来处理剩余数据。不同于通常以延迟为唯一优化目标的关系查询处理，语义查询处理需要考虑多个目标，包括查询准确率、查询延迟以及大语言模型调用的货币成本。因此，Sema 的 AQE 旨在查询延迟和货币成本方面找到帕累托最优路径，同时满足用户指定的准确率不一致容忍度。如果对准确率损失的容忍度较高，或者延迟和成本是重点关注因素，则可以轻松扩展系统的帕累托优化目标。由于语义查询的真实值通常无法预先获得，我们使用一条参考路径（例如优化器生成的、不采用算子融合和批量提示的路径）来评估其他候选路径的准确率偏差。

AQE框架分为三个阶段：表达式探索阶段、路径探索阶段和路径利用阶段。算法1展示了AQE的整体流程。在表达式探索阶段（第3–4行），我们使用一小部分 δ_1 元组，收集每个 *SemFilter* 的统计信息以及参考路径 p_{ref}的伪真实值。随后，我们利用这些统计信息生成一组候选路径P，并在另一小部分 δ_1 元组上评估这些路径的性能，根据时间效率偏好或令牌效率偏好，选出P中的最优路径 $p^\star$（第5–11行）。最后，在路径利用阶段（第12行），执行器将处理剩余的$1-(\delta_1+\delta_2)$元组。用户可以通过调整系统参数 δ_1 和 δ_2，来平衡探索与利用。请注意，整体查询结果是三个阶段所得结果的拼接：（1）分别在每个 *SemFilter* 上逐个评估 δ_1 元组，并对结果求交集；（2）通过 p_{ref} 评估 δ_2 元组；（3）通过 $p^\star$评估剩余的 $1-(\delta_1+\delta_2)$ 元组。各阶段的完整算法见附录B。下文将详细介绍各阶段的关键设计。

算法1： 自适应查询执行

输入： 输入数据块 D，expr_限制 $\delta_1 \cdot |D|$，path_限制 $\delta_2 \cdot |D|$

$n \leftarrow 0$;
对于数据块 $c \in \mathcal{D}$ **执行**
  如果 $n <$ *expr*_限制**则**
    收集 $c \triangleright$ 阶段 *1*：表达式探索 中筛选器的统计信息;
  否则，如果 $n <$ 路径_限制，**则**
    如果 未生成路径，**则**
      利用 $c \triangleright$的统计信息生成候选路径 P
      阶段 *2*：路径探索 ;
    评估 c 上的 P 中的路径，并收集其指标;
  else
    若未选中最优路径**则**
      根据收集到的指标选择最优路径 $p^\star$;
    执行最优路径 $p^\star$ 于 $c \triangleright$ 阶段 *3*：路径利用;
  $n \leftarrow n + |c|$;

阶段 *1*：表达式探索。给定一组*SemFilter*$\Sigma = \{\sigma_1, \cdots, \sigma_n\}$，在表达式探索阶段，我们在一部分 δ_1 元组上分别独立执行每个 *SemFilter* σ_i，从而得到其选择率 s_i 和查询结果 R_i，后者表示为布尔向量。我们利用查询结果评估算子融合的潜力。具体而言，两个过滤器之间的相关性越高，融合越有可能保留其语义。我们使用马修斯相关系数（MCC） [39]，从结果的角度对两个*SemFilter*之间的成对相关性进行统计量化；其中，MCC(σ_i, σ_j)等于1且 −1时，分别表示 σ_i和 σ_i之间完全正相关和完全负相关；MCC(σ_i, σ_j)等于0则表示随机关联。只有MCC大于某一阈值的一对*SemFilter*才会被融合为一个*SemFilter*。

阶段 *2*：路径探索。根据阶段 1 收集的统计信息，我们首先生成一组候选路径 P，其中包括参考路径 p_{ref} 和基础路径 p_{base}。参考路径 p_{ref} 保留 SemaSQL 查询中用户指定的 Σ 顺序，而基础路径 p_{base} 则按照选择率升序重新排列 Σ 中的过滤器。对于算子融合，如果 MCC(σ_i, σ_j) 大于阈值 τ，我们就将两个 *SemFilter*算子 σ_i、σ_j 合并为一个*SemFilter*$\sigma_{i\oplus j}$，并将其选择率 $s_{i\oplus j}$视为s_i和s_j中的最小值。对于包含 n个*SemFilter*算子的情况，我们只考虑在候选路径中引入一次融合，以防止查询准确率下降和探索空间膨胀；因此，最多生成$n(n-1)/2$条候选路径，其中 $n-1$ 个算子按选择率升序排列。对于所有包含融合算子的路径，以及参考路径和基础路径，我们还会生成采用批量提示的对应路径。

接下来，执行器在 δ_2 元组的一部分上评估这些候选路径，并收集查询延迟、令牌消耗和查询结果。由于事先无法获得语义查询的真实值，我们将 p_{ref} 未采用批量提示时的查询结果视为代理真实值，用于评估其他候选路径的准确率。如果准确率（例如 F1值或准确率）低于阈值 τ_{acc}，则表明相应路径生成的结果与参考路径相差甚远，因此将舍弃该路径，以保证较高的结果质量。

在路径评估过程中，我们设计了两种动态剪枝策略来降低探索成本。首先，如果采用批量提示的参考路径准确率低于阈值，系统将不再探索所有采用批量提示的路径。其次，如果某个采用算子融合的候选路径准确率未达到阈值，则会跳过对其对应的批量提示路径的评估。评估完候选路径后，我们在查询延迟和令牌成本构成的双目标空间中计算帕累托前沿。根据用户或应用的偏好，选择延迟优先或令牌优先的最优路径作为最优路径 $p^\star$。

阶段 *3*：路径利用。所选路径 $p^\star$ 用于处理剩余的大部分数据。由于该计划已通过效率和准确率验证，此阶段能够确保性能可预测且经过优化，同时摊销路径探索的成本。

我们在 §7.5 中通过一个包含三个 *SemFilter* 算子的特定查询，展示 AQE 案例研究，作为说明性示例。目前，AQE 仅支持 DuckDB 单个执行管道中的 *SemFilter* 算子。在运行时优化更复杂的算子，尤其是跨多个管道的算子，会导致实现过于复杂，并产生较大的上下文切换开销，因此值得进一步进行深入的工程优化。

6 工程细节

我们介绍 Sema 的关键工程细节，重点关注并发大语言模型调用和管道并行自适应查询执行（AQE）。

并发大语言模型调用。为了将大语言模型推理集成到系统中，Sema 通过线程本地并发 I/O 设计扩展了 DuckDB 的管道并行引擎。DuckDB 使用基于 morsel 的执行框架 [31]，将工作分解为细粒度的 morsel，每个 morsel 在一个管道中处理一小批元组。中央调度器从共享池中动态地将 morsel 分配给工作线程，从而确保负载均衡，并以最小的同步开销高效利用 CPU。在 Sema 中，每个工作线程都维护一个专用的 CURL 多句柄实例，作为其私有 HTTP 连接池。执行期间，线程在 morsel 循环中同步发出 LLM API 请求，而 CURL 多句柄则支持非阻塞推进和高效复用连接。这种设计避免了跨线程同步，使多个工作线程能够并发发出和管理大语言模型请求。因此，Sema 保留了 DuckDB 的细粒度并行性，并充分利用其任务调度器处理计算和 I/O 密集型工作负载。

流水线并行 AQE。Sema 通过定制算子 *PhysicalSemAdap-tiveFilter*，将 AQE 直接集成到 DuckDB 的流水线并行引擎中，将自适应决策与流式执行融为一体。每个流水线执行器都会逐步拉取数据块，并通过 Sema 的自适应三阶段执行流程对其进行处理，动态选择并应用最优执行路径。借助 DuckDB 的流式原语（NEED_MORE_INPUT 和 HAVE_MORE_OUTPUT），Sema 逐步进行数据切片，在探索阶段仅评估少量输入数据，然后为剩余数据采用最优路径。

在执行过程中，多个并发执行器在互不相交的数据分区上并行运行，各自维护线程本地状态

（包括表达式求值器和中间缓冲区），同时向共享全局状态报告轻量级性能指标（例如，选择率、延迟和令牌级成本）。这种双状态架构能够在保持局部自主性的同时实现全局协调：自适应决策被集体制定，却独立执行，从而尽量降低同步开销。随着流水线推进，全局控制器通过原子更新聚合反馈，并将优化后的执行选择传播给所有线程。因此，Sema 能够在不同的数据分布下实现细粒度自适应性与稳定的吞吐量，并与 DuckDB 轻量级、流式且内存高效的执行模型保持兼容。

7 系统评估

我们在§7.1中介绍实验设置，并从以下方面进行全面评估：1）将 Sema 与基线系统在输出质量和效率方面进行比较（§7.2）；2）评估算子融合和提示批处理对 Sema 的影响（§7.3）；3）探究 LLM 辅助表达式优化的效用（§7.4）；4）研究分解优化的有效性（§7.5）。

7.1 实验设置

数据集。 我们使用来自 BIRD [32], a 的 9 个数据集。BIRD 是一个广泛使用的、最初为 NL2SQL 设计的基准。附录C中的表7总结了这些数据集的概况，其中每个数据集都代表不同的分析应用程序。具体而言，*Appstore*、*Superstore* 和*Food_Inspection* 侧重于实体级语义提取，而 *Movies_4* 和*Authors* 等其他数据集则强调多表推理和文本密集型查询评估。这种多样性确保了能够在结构化数据和非结构化数据上，对 Sema 的优化策略进行稳健验证。

基准查询。 基于这9个数据集，我们设计了20个语义查询，构成了用于语义查询处理的全面测试平台。这些查询分为三类：❶ 单算子查询（Q1-Q5）：包含一个*SemFilter* 或*SemProj*。❷ 连续*SemFilter* 查询（Q6-Q10）：包含2-3个连续的*SemFilter*算子，用于评估AQE框架和执行优化。❸ 包含连续语义算子（但不包括*SemFilter*）的查询（Q14-Q20）：包括包含*SemProj* 和*Sem-Filter* 的查询（Q14-Q15）、包含两个*SemProj* 算子的查询（Q14-Q15）、包含*SemFilter* 和*SemProj* 的查询（Q16-Q18），以及包含*SemFilter* 和*SemAgg* 的查询（Q19-Q20）。Se- maSQL中的全部20个查询见附录C图13-32。Q1-Q5的真实值结果取自原始数据集，而Q6-Q20的结果则来自参考大语言模型（默认使用Gemma-3 27B）的推理结果。

评估指标。 在质量方面，针对每个查询的具体目标，我们采用一系列评估指标：❶ Q1-Q14：单标签、多分类分类；我们报告准确率、精确率、召回率和F1值。❷ Q15：多标签、多分类分类。我们分别报告子集准确率、微平均F1分数、宏平均F1和杰卡德系数。❸ Q16-Q20：每个查询都涉及两个连续算子。我们首先评估*SemFil- ter*的二分类性能，然后评估下游任务。具体而言，Q16是关键词提取任务（通过宏平均精确率/召回率/F1值和杰卡德系数进行评估）；Q17和Q18是分类任务（加权F1分数、微平均F1分数、宏平均F1）；Q19和Q20侧重于聚合和摘要（通过生成结果与真实结果之间的词语重叠进行衡量）。在时间和成本效率方面，我们使用查询延迟、大语言模型调用次数、输入/输出令牌数量以及API调用的成本（¢）作为评估指标。

基线系统。 我们将 Sema 与三个具有代表性的系统进行比较：基于 Pandas 构建的 Lotus [42]，其针对*SemFilter* 和 *SemJoin*采用了优化算法；配备级联优化器的 Palimpzest [35]；以及在 DuckDB 中将语义算子实现为用户定义函数的 FlockMTL [18],。这些基线系统与 Sema 在向量化执行、流水线并行和异步执行方面的能力如下所示：

能力	Lotus	FlockMTL	Palimpzest	SEMA
向量化执行	✓	✗	✗	✓
流水线并行	✗	✓	✓	✓
异步执行	✓	✗	✓	✓

实现细节。 Sema 基于 DuckDB v1.2.2 构建，同时支持本地和远程大语言模型调用。我们分别使用 vLLM [30] 作为本地大语言模型推理引擎，并使用 *OpenRouter* 作为远程大语言模型 API。我们部署了谷歌的 Gemma-3 模型 [49]，包括 4B、12B 和 27B 版本，并将温度设为 0，以确保推理可复现性。温度为 0 的大语言模型无论如何重排序 *SemFilter*，都会生成相同的查询结果。对于 vLLM，所有模型均配置为使用张量_并行_大小 =4、bfloat16 精度，并将 GPU 内存利用率限制设为 0.9。本地实验在一台 Linux 服务器上进行，该服务器配备四块 NVIDIA RTX 5090 GPU 和一颗 AMD EPYC 9554 CPU（64 核，503 GB 内存）。远程 API调用的定价遵循 DeepInfra 在 *OpenRouter*上的费率：Gemma-3 12B 的输入令牌为每百万 0.04 美元，输出词元为每百万 0.13 美元；GPT-4.1 的输入令牌为每百万 2 美元，输出词元为每百万 8 美元。

7.2 质量与效率总体评估

我们使用 20 个语义查询，将 Sema 的输出质量和查询延迟与三个基线系统进行比较。为降低网络延迟和波动的影响，所有系统均使用本地 vLLM。对于 Q1–Q5，由于这些查询仅涉及单个算子且相对简单，我们使用 Gemma-3 4B 模型；对于由多个算子组成、更加复杂的查询 Q6–Q20，则采用 Gemma-3 12B 模型。对于 Palimpzest，我们使用 *MinTime* 优化器策略执行所有查询。

整体效率比较。如图 4 所示，在全部 20 个查询中，Sema 的查询延迟始终优于所有三个基线系统。尤其是在复杂查询（Q6–Q20）中，Sema 的性能优势十分显著，最高可实现 6× 倍加速。这一提升归功于 Sema 的向量化执行、流水线并行和异步执行机制，这些机制使其能够有效利用硬件并发，并在进行大语言模型调用时与 vLLM 高效协同。相比之下，三个基线系统因缺少执行优化而表现出明显局限。Lotus 虽然支持向量化和异步执行，但缺少流水线并行，因此无法实现算子之间的有效重叠。FlockMTL 则存在更根本的

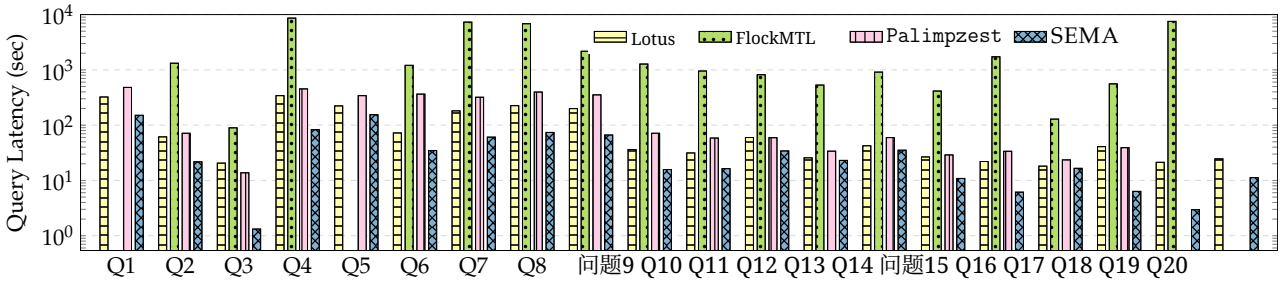

图4：总体查询延迟（秒）

表3：Q1-Q14 的查询准确率（FlockMTL 不支持 Q1、Q5 和 Q20。）

系统	Q1 Acc.	Q1 Pre.	Q1 Rec.	Q1 F1	Q2 Acc.	Q2 Pre.	Q2 Rec.	Q2 F1	Q3 Acc.	Q3 Pre.	Q3 Rec.	Q3 F1	Q4 Acc.	Q4 Pre.	Q4 Rec.	Q4 F1	Q5 Acc.	Q5 Pre.	Q5 Rec.	Q5 F1	Q6 Acc.	Q6 Pre.	Q6 Rec.	Q6 F1	Q7 Acc.	Q7 Pre.	Q7 Rec.	Q7 F1
Lotus	0.9940	1.0000	0.9940	0.9970	0.7367	0.8148	0.7367	0.7450	0.7157	1.0000	0.7157	0.8343	0.4194	0.8837	0.4194	0.4534	0.3040	1.0000	0.3040	0.4662	0.6977	1.0000	0.2561	0.4078	0.8489	0.8722	0.6102	0.7180
FlockMTL	-	-	-	-	0.7973	0.8221	0.7973	0.7994	0.2294	1.0000	0.2294	0.3732	0.4897	0.7182	0.4897	0.4543	-	-	-	-	0.7195	0.9023	0.3475	0.5017	0.7504	0.9558	0.2185	0.3557
Palimpzest	0.9461	1.0000	0.9461	0.9723	0.3832	0.4471	0.3832	0.4022	0.9426	1.0000	0.9426	0.9705	0.2573	0.4080	0.2573	0.2326	0.5799	1.000	0.5799	0.7341	0.7011	0.9928	0.2664	0.4201	0.8556	0.8316	0.6796	0.7480
Sema	0.9934	1.0000	0.9934	0.9967	0.7369	0.8154	0.7369	0.7407	0.7124	1.0000	0.7124	0.8320	0.4428	0.7349	0.4428	0.4878	0.1265	1.0000	0.1265	0.2245	0.7030	1.0000	0.2693	0.4243	0.8148	0.7825	0.5715	0.6606

系统	Q8 Acc.	Q8 Pre.	Q8 Rec.	Q8 F1	Q9 Acc.	Q9 Pre.	Q9 Rec.	Q9 F1	Q10 Acc.	Q10 Pre.	Q10 Rec.	Q10 F1	Q11 Acc.	Q11 Pre.	Q11 Rec.	Q11 F1	Q12 Acc.	Q12 Pre.	Q12 Rec.	Q12 F1	Q13 Acc.	Q13 Pre.	Q13 Rec.	Q13 F1	Q14 Acc.	Q14 Pre.	Q14 Rec.	Q14 F1
Lotus	0.8542	0.6728	0.8921	0.7670	0.9959	0.5000	0.1154	0.1875	0.9456	0.9412	0.3743	0.5356	0.8697	0.9887	0.3954	0.5648	0.7358	0.6534	0.4096	0.5035	0.7181	0.8864	0.6754	0.7666	0.7168	0.7767	0.7168	0.7115
FlockMTL	0.7445	0.9706	0.0574	0.1083	0.9962	1.0000	0.0769	0.1429	0.9251	0.6809	0.1882	0.2949	0.1563	0.0870	0.3103	0.1360	0.6613	0.4722	0.2993	0.3664	0.7440	0.8196	0.8035	0.8115	0.7059	0.7684	0.7059	0.7249
Palimpzest	0.8644	0.7979	0.6680	0.7272	0.9956	0.4167	0.1923	0.2632	0.9407	0.5953	0.9000	0.7166	0.9381	0.8503	0.8621	0.8562	0.6855	0.5168	0.5951	0.5532	0.7305	0.8441	0.7445	0.7912	0.6517	0.6763	0.6517	0.6543
Sema	0.8426	0.6958	0.7375	0.7160	0.9955	0.3636	0.1538	0.2162	0.9543	0.9364	0.4873	0.6410	0.9651	0.9740	0.8598	0.9133	0.7273	0.6082	0.4683	0.5292	0.7315	0.9012	0.6834	0.8883	0.7570	0.7954	0.7570	0.7563

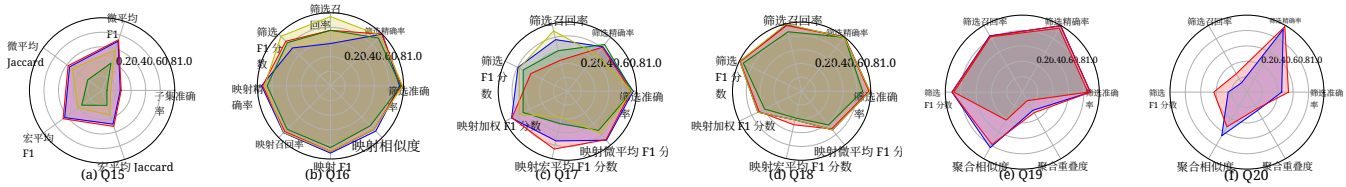

图 5：查询准确率对比：Lotus、Sema、Palimpzest 和FlockMTL在 Q15-Q20 上的表现

效率问题：它既未利用向量化执行，也不支持异步执行，导致资源利用效率低下、执行速度较慢。Palimpzest 虽然支持流水线并行和异步执行，但未利用向量化执行，因此无法充分受益于高吞吐量执行模型，整体性能也因此受限。由于某些列包含多种数据类型，FlockMTL 无法支持查询 Q1、Q5、Q19 和 Q20。Palimpzest 目前不支持使用 *SemAgg* 进行语义聚合，因此无法执行查询 Q20。

总体质量比较。 表 3 和图 5 显示，Sema、Lotus 和 Palimpzest 在大多数评估指标上的准确率相当，各系统在不同查询上各有优势。这种差异很大程度上源于大语言模型输出对提示的敏感性；我们在测试中观察到，提示处理上的细微差异可能导致结果出现明显波动。尽管 Sema 的提示格式经过精心调整，与 Lotus 保持一致以确保公平比较，但由于系统架构和执行流程存在差异，仍有细微出入。总体而言，在简单和复杂查询上，Sema 与 Lotus 和 Palimpzest 相比，性能均衡且具有竞争力。相比之下，FlockMTL 在几乎所有指标上都明显逊色，这反映出其模型调用策略和整体系统鲁棒性的局限。

7.3 执行优化评估

为评估执行优化的效用，我们比较了 Sema 在四种配置下的性能：(1)不进行优化，(2) 使用批处理大小为 16 的提示批处理，(3) 使用算子融合，以及 (4) 使用批处理与融合。所有实验均使用 Gemma-312B，并分别通过远程 API 和本地方式调用。我们选取了四个查询 Q8、Q13、Q14 和 Q16，这些查询包含不同的算子组合，用于评估上述配置。

实验结果汇总于图 6，该图比较了在远程和本地部署设置下，不同工作负载中提示批处理与算子融合的延迟、执行成本和输出质量。在延迟方面，远程部署时，批处理和融合都能缩短整体执行时间，因为它们减少了大语言模型调用的次数，从而分摊了网络和调度开销。由于批处理会将更多请求合并到一次调用中，因此其延迟始终低于融合。不过，在本地部署时，批处理和融合可能会增加延迟。这两种技术都以更长的输入和输出序列换取更少的调用次数，而 vLLM 连续批处理调度器针对大量短请求进行了优化，

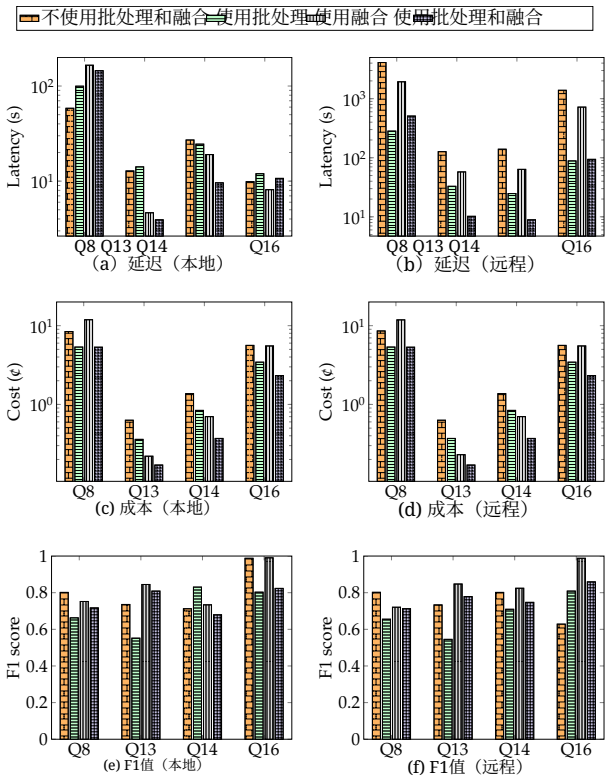

图6：执行优化的有效性

而非较少的长请求，因此会降低调度效率。在成本方面，提示批处理和算子融合通常会减少模型处理的冗余输入令牌，从而降低查询执行成本。Q8 是一个例外：融合未能降低成本，甚至可能使成本上升。这主要是因为融合合并了涉及过滤算子的操作；当过滤谓词的选择率较低时，仍需处理额外的不必要令牌，抵消了潜在的节省。在输出质量方面，由于单个提示中同时处理多个查询会相互干扰，批处理始终会导致性能下降。融合的效果则因任务而异：它能提升 Q13、Q14 和 Q16 的质量，但会降低 Q8 的质量。原因在于，Q8 由相互独立的任务组成，联合处理会引入干扰；而 Q13、Q14 和 Q16 是具有顺序依赖关系的任务，共享上下文使模型能够复用中间推理，从而取得更好的结果。

7.4 表达式优化评估

我们在 Q3、Q6、Q13 和 Q14 上，对两种自然语言表达式优化——表达式压缩和谓词推导——进行了有效性研究，结果汇总于表4。为验证表达式压缩，我们特意在查询的自然语言表达式中引入了语义冗余内容。为验证部分推导，我们将某些 SQL 谓词改写为自然语言表达式，以评估谓词推导能否有效实现演绎。这些查询变体见附录C中的图33–40。

表4：表达式优化的影响。此处，“+”表示辅助大语言模型的调用成本（以红色标示）

Q	Opt.	Pre.	Rec.	F1	延迟（秒）	输入标记数（千）	输出标记数（千）	大语言模型调用
Q3	不含早期退出	1.0000	0.9880	0.9944	95.43	91.24	1.08	401
	使用早期退出	1.0000	0.9890	0.9945	102.05	72.20+1.69	1.00+0.09	401+1
	不使用提示分解	1.0000	0.9950	0.9975	410.66	191.24	6.08	856
	使用提示分解	1.0000	0.9950	0.9975	135.11	73.24+1.07	1.07+0.051	492+2
Q6	不使用早期退出	0.9868	0.9418	0.9637	256.42	9.16	0.15	59
	使用早期退出	0.9818	0.9795	0.9803	239.32	8.15+1.46	0.14+0.03	59+1
	不使用提示分解	1.0000	0.9072	0.9513	443.66	7.76	0.16	59
	使用提示分解	1.0000	0.9072	0.9513	447.67	7.76 + 1.13	0.16 + 0.03	59+2
Q13	不使用早期退出	0.8112	0.4610	0.5878	342.25	38.73	3.98	318
	使用早期退出	0.7647	0.6569	0.7067	357.97	34.84+3.38	4.28+0.07	318+1
	无提示分解	0.8057	0.5293	0.6388	637.30	57.10	4.62	601
	有提示分解	0.8182	0.5505	0.6582	346.65	31.64+1.25	3.90+0.03	318+2
Q14	无早期退出	0.5314	0.2744	0.3031	554.89	74.53	8.04	548
	带早期退出	0.4603	0.2506	0.2814	556.16	62.47+3.42	9.54+0.08	548+1
	不使用提示分解	0.6134	0.616	0.6147	1055.36	97.15	7.65	1023
	使用提示分解	0.3787	0.6338	0.4741	548.93	57.78+0.99	8.80+0.02	552+2

表达式压缩（EC）主要通过减少自然语言表达式中的冗余来提升词元效率。由于 EC 会将冗长的自然语言表达式改写为更紧凑且语义等价的形式，使大语言模型能够处理更短的提示，因此在保持执行质量相近的同时，输入令牌和输出词元的消耗都会降低。Q3 中的一个典型例子是：应用 EC 后，输入令牌从 73.24K（未使用 EC）降至 72.20K，输出词元从 1.07K 降至 1.00K，而精确率、召回率和 F1值几乎保持不变。Q6 和 Q13 中也呈现出类似趋势，这表明 EC 能有效提升词元效率，同时不会造成语义失真或降低结果质量。

谓词推导（PD）侧重于减少查询执行期间所需的大语言模型调用次数。PD直接从自然语言表达式推导出必要的SQL谓词，并将其注入执行过程，从而在大语言模型调用之前筛除元组。这一优势在Q13中尤为明显：引入PD能以更少的调用和更低的开销带来效率提升，同时保持相当的结果质量。

7.5 Sema的分解评估

为研究Sema的AQE框架的有效性，我们在远程场景和本地场景中对查询Q6–Q10进行实验，每个查询都涉及多个语义过滤谓词。我们将用于表达式探索和路径探索的数据比例分别设为 $\delta_1 = \frac{1}{32}$ 和 $\delta_2 = \frac{3}{32}$，其余28 32 3 12数据由所选路径处理。我们使用本地Gemma- B进行大语言模型推理。我们将*SemFilter*融合和 τ_{acc} 的MCC阈值分别设为0.5和0.80。

图 7 展示了在不同 AQE 配置以及两种环境（本地和远程）下，Q6–Q10 的综合查询级指标和延迟细分情况。三项指标在本地和远程环境中表现出显著差异。总体而言，结果表明，在本地环境中，批处理会带来成本与延迟之间的权衡，而融合对延迟的影响可能是正面的，也可能是负面的，具体取决于数据集；在远程部署中，两种优化均能有效实现帕累托改进，体现了环境感知自适应优化。

在本地环境中，提示批处理和算子融合都能持续降低总令牌成本。不过，它们对延迟的影响

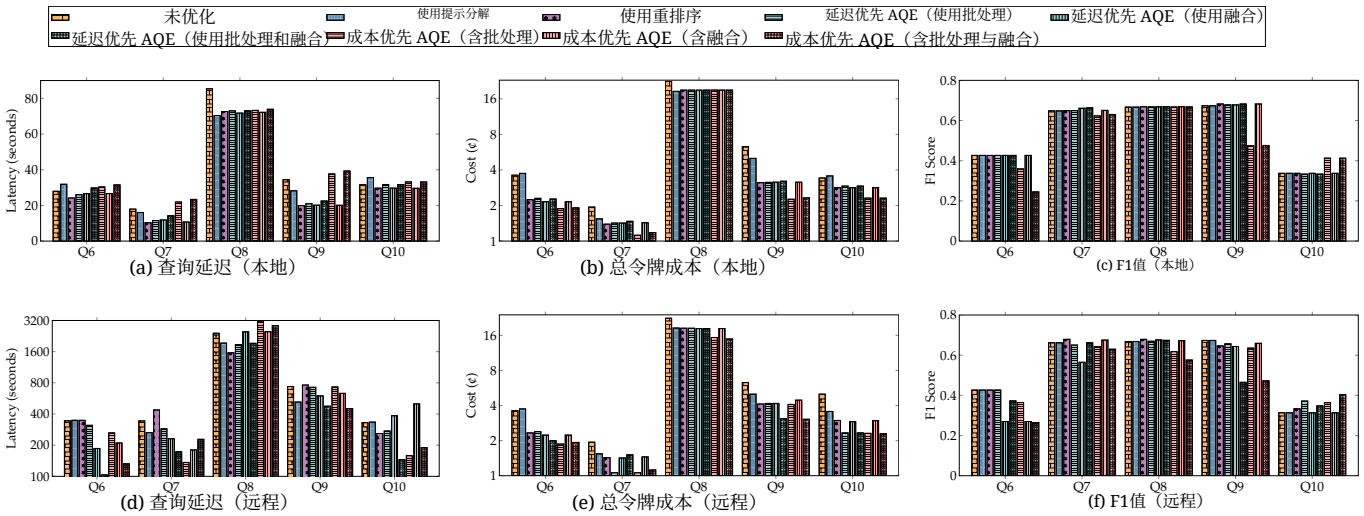

图7：Sema 在本地和远程 API 下的查询延迟、令牌成本和 F1值分解评估

各不相同：提示批处理会增加查询延迟，而算子融合带来的延迟变化则取决于数据集——根据工作负载特征，延迟可能增加，也可能降低。这些技术的效果还受到 vLLM 调度器的影响；该调度器相对更适合处理数量较多、长度较短的大语言模型提示，而非数量较少、长度较长的大语言模型提示。因此，提示批处理和算子融合的副作用是，它们可能会根据提示的长度降低 Sema 的并行性。由此，在成本优先的 AQE 目标下，优化器倾向于选择融合和／或批处理计划，以更高的延迟为代价（主要由批处理导致）实现显著的成本降低。在延迟优先的 AQE 目标下，优化器会避免激进的批处理，并仅在融合有利于给定数据集的延迟时才应用融合，因此相较于基线，延迟收益有限或不稳定。在远程环境中，情况则截然不同：由于网络往返开销占据执行时间的主要部分，通过融合和批处理减少远程调用次数，能够同时降低延迟和成本。因此，延迟优先和成本优先 AQE 往往会选择相似的优化计划，并在两项指标上都取得稳定的改善。

性能特征：本地与远程。基于大语言模型的查询执行延迟主要由两个因素决定：大语言模型调用次数以及每次调用的令牌长度。不过，主导因素会随环境而变化。对于本地大语言模型调用，执行主要受本地 GPU 的计算能力和内存大小限制。在这种情况下，吞吐量相对固定。因此，延迟对每次调用的令牌长度非常敏感，而调用次数的影响则较小。对于远程大语言模型调用，性能则主要受网络开销（例如往返时间、握手和排队时间）以及 API 提供商速率限制的影响。在这种情况下，每次调用的开销相当大，因此延迟对大语言模型调用次数非常敏感；而每次调用的令牌长度适度增加，对总耗时的影响可以忽略不计。

```
SELECT coins.id AS coin_id FROM coinsWHERE category
 = 'coin' AND description is NOTNULLAND s'{
description } 表明这种加密货币的供应量很大。'AND s'{
description } 表明这种加密货币仍在流通。'AND s'从 {
description } 可以推断，这种加密货币的价格正在上涨。');
```

图8：AQE 案例研究中的 Q6

表 5：Q6 候选路径的微执行结果

描述	Path	准确率 (%)	延迟 (s)	成本 ($)
$([\sigma_3, \sigma_1, \sigma_2])$	p_1	100.00	0.997	0.0008
$([\sigma_3, \sigma_1, \sigma_2])$ 含批处理	$\hat{p}_1$	94.04	1.936	0.0006
$[\sigma_2 \oplus \sigma_3, \sigma_1]$	p_2	100.00	0.504	0.0007
$[\sigma_2 \oplus \sigma_3, \sigma_1]$ 含批处理	$\hat{p}_2$	91.66	1.412	0.0005

多目标策略下的权衡。当 AQE 被配置为多目标优化时，结果揭示了延迟与成本之间清晰且可控的权衡，这与我们的设计直觉一致。延迟优先策略始终获得最短的查询运行时间（例如，问题6为29.64秒，问题9为51.94秒），而成本优先策略则产生最低的货币成本（例如，问题6为2.21美分，问题9为3.04美分）。

AQE案例研究。我们进一步使用带有本地大语言模型调用的 Q6（图8）来说明 AQE 的动态过程。该查询会检索供应量大（σ_1）、流通活跃（σ_2）且价格上涨（σ_3）的加密货币。在表达式探索阶段，我们在一个 $1\,32$ 子样本（27 行）上分别评估了三个谓词，估算其选择率为 $s(\sigma_1) = 0.66$、$s(\sigma_2) = 0.77$ 和 $s(\sigma_3) = 0.22$。对 MCC 进行相关性分析后发现，$\mathrm{MCC}(\sigma_1, \sigma_2) = 0.755$，$\mathrm{MCC}(\sigma_1, \sigma_3) = 0.377$。

表 6：规划时间（毫秒），以及 AQE 阶段1-3 的分解时间（t1、t2、t3，单位为秒）

指标 -	Q6				Q7				Q8				Q9				Q10			
	p_{ref}	p_{base}	$p^{\star}_{\text{延迟}}$	$p^{\star}_{\text{cost}}$	p_{ref}	p_{base}	$p^{\star}_{\text{延迟}}$	$p^{\star}_{\text{cost}}$	p_{ref}	p_{base}	$p^{\star}_{\text{延迟}}$	$p^{\star}_{\text{cost}}$	p_{ref}	p_{base}	$p^{\star}_{\text{延迟}}$	$p^{\star}_{\text{cost}}$	p_{ref}	p_{base}	$p^{\star}_{\text{延迟}}$	$p^{\star}_{\text{cost}}$
计划（毫秒）	30	30	29	30	54	51	52	55	51	55	49	49	54	54	56	50	49	46	56	54
t1（秒）	0.00	0.57	0.57	0.57	0.00	0.32	0.33	0.34	0.00	0.59	0.62	0.65	0.00	0.61	0.60	0.64	0.00	1.30	1.35	1.32
t2（秒）	0.00	0.00	3.45	4.86	0.00	0.00	1.79	1.81	0.00	0.00	3.93	5.72	0.00	0.00	1.42	1.45	0.00	0.00	3.24	3.24
t3（秒）	0.00	24.60	25.61	30.85	0.00	41.49	40.86	71.81	0.00	70.87	70.24	120.05	0.00	39.86	39.30	54.82	0.00	28.75	27.29	26.60

σ_1 与 σ_2之间存在很强的正相关，表明它们有潜力进行算子融合；而 σ_3 是选择性最高且相对独立的谓词。根据这些统计信息，Sema 生成了四条新的执行路径：一条带有 *SemFilter*重排序（$[\sigma_3, \sigma_1, \sigma_2]$）的基础路径 p_1，一条采用算子融合（$[\sigma_2 \oplus \sigma_3, \sigma_1]$）的基础路径变体 p_2，以及分别对 p_1和 p_2进行提示批处理的变体，记为 $\widehat{p}_1$和 $\widehat{p}_2$。在路径探索阶段，这些候选路径会在一个 3 32 子样本（此处为84 行）上进行微执行，以比较准确率（相对于基础路径）、延迟和成本。这些路径的结果汇总于表 5，其中的“Acc”表示该路径相对于 p_1的准确率。

谓词融合（p_2）显著减少了令牌输入/输出和延迟；与基础路径（p_1）相比，延迟最高降低 49%，成本降低 9%，同时保持了完整的准确率。相比之下，在此情形下，批处理并不总能降低延迟。我们观察到，批处理执行（$\widehat{p}_1$ 和 $\widehat{p}_1$）有时比对应的非批处理执行耗时更长。这种差异可能是环境限制所致：批处理虽然减少了大语言模型请求，但会增大每个请求的规模并降低并发，从而延长本地环境中的端到端处理时间。给定准确率阈值 $\tau_{acc} = 0.80$，路径 $\widehat{p}_1$ 和 p_2 位于帕累托前沿。采用延迟优先策略时，p_2 被选为最优路径 $p^{\star}$；采用成本优先策略时，则优先选择 $\widehat{p}_1$。

规划时间。我们在表 6 中报告了 AQE 三个阶段的执行时间。总体而言，在所有查询和计划变体中，规划时间始终较短。端到端开销主要来自最终的路径利用阶段（t_3），而两个探索阶段（t_1 和 t_2）只产生了较小的成本。与利用阶段相比，AQE 的规划/探索成本仍然很低，这证实了该框架的大部分时间都花在利用而非探索上。

8 相关工作

数据库内机器学习。为数据库系统配备机器学习功能是一个长期存在的研究课题。早期研究 [38, 55] 使用关系代数表示线性代数运算，使机器学习模型能够在数据库管理系统中用 SQL 表示并执行。另一条研究路线是在矩阵和向量原语之上实现用户定义函数和计算库。例如，MADlib [20] 是一个数据库内库，提供矩阵和向量运算，并利用用户定义函数和用户定义聚合函数实现梯度下降 [10, 19]。为支持多种统计模型，研究人员设计了蒙特卡洛数据库系统 [11, 22]，在数据库中提供专门用于采样的用户定义函数，从而能够通过 SQL 查询进行贝叶斯学习。基于模型的视图 [16, 29] 这一概念也已被提出，用于在底层训练数据更新后维护数据库管理系统中的机器学习模型。如今，许多商用和云数据库系统都集成了机器学习功能，例如 BigQuery ML [2], 、Amazon Redshift ML [1], 和 Oracle Data Mining [3]。近来，数据库管理系统将深度学习模型注入用户定义函数，以处理复杂的分析查询；大语言模型要么在用户定义函数中被调用 [18, 36], ，要么用于生成用户定义函数的代码 [54]。

通用机器学习分析系统。 机器学习的广泛应用推动了通用机器学习分析系统的发展。视频数据库管理系统（VDBMS） [7, 25, 26, 28, 44, 52] 专为视频数据分析而设计，在数据库管理系统中将深度学习模型用作用户定义函数，以执行图像分类和目标检测等任务。为高效处理视频分析查询，这些系统采用了多种优化技术，包括对预处理和查询流水线进行端到端优化 [28], 、声明式查询提示 [44],、细粒度模型选择 [12],、查询无关语义索引 [7], ，以及针对特定算子的快速评估算法 [7, 25]。除视频分析外，基于机器学习的概率谓词 [17, 37, 53] 也已应用于更广泛的应用领域。这些研究构建轻量级代理模型，并利用不同谓词之间的相关性。对于结合机器学习算子和关系算子的混合查询，RAVEN [41] 对机器学习算子和关系算子进行交叉优化，利用数据统计信息从多种执行策略中进行选择。此外，一些系统还采用自适应查询处理，根据运行时统计信息动态调整查询计划和资源分配，或选择成本更低的代理模型。ABAE [27], 和 InQuest [45]专注于通过引入基于采样的聚合算法，优化机器学习模型上的聚合查询。

基于大语言模型的语义查询处理。 大语言模型为结构化数据 [18, 36] 和非结构化数据 [5, 48, 50]的语义查询处理带来了新的范式。Do- cETL [48],ZenDB [34],DocWanger [47],Aryn [5] 和 Unify [50]等系统利用大语言模型通过语义查询对非结构化文本进行分析，并支持多种查询接口，包括扩展 SQL 方言 [34], 声明式操作 [48] 和自然语言 [5, 47, 50]。在这些文档分析系统中，大语言模型还可在查询执行过程中充当优化代理，协助完成查询规划 [5], 查询重写、提示优化和操作分解等任务 [47]。UQE [14] 扩展了 SQL，允许使用自然语言编写连接谓词和选择谓词，从而内在支持由大语言模型执行的语义连接和语义筛选。作为由大语言模型驱动的分析系统，Palimpzest [35] 提供逻辑和物理优化，以降低查询时间和大语言模型调用的经济成本。在 Palimpzest 的基础上，Abacus [46] 引入了用于查询规划的帕累托优化算法和成本估计。Lotus [42] 将基于大语言模型的语义操作应用于表格数据，并针对成本高昂的语义操作提出了优化算法。 [4] 评估了语义

查询性能，并考察了 PostgreSQL 搭配 pgAI 以及 DuckDB 搭配 FlockMTL 的表现 [18], ，指出在结构化输出、批处理和查询规划方面仍面临挑战。据我们所知，Sema 是首个支持原生语义算子并在 DuckDB 之上实现端到端优化的系统。

自适应查询执行（AQE）。在过去二十年中，AQE 已从细粒度元组路由发展到稳健的、面向特定领域的执行框架。持续自适应的概念由 Eddies [6],率先提出，它以实时元组路由取代了静态算子树。这一思想后来由 CBR [8], 加以改进，证明基于数据内容的路由优于简单的平均统计信息。然而，为降低元组级路由的高开销，后续研究已转向更实用的基于阶段的自适应。例如，Polar [23] 通过最小阻力计划自适应地切换连接顺序，从而缓解基数估计误差，在无需重构优化器的情况下实现稳健执行。最近，AQE 进一步扩展，以支持数据库管理系统中日益普及的机器学习算子。Aero [24] 将 AQE 应用于优化以机器学习为核心的查询，根据运行时数据特征动态调整代价高昂的用户定义函数和概率模型的调用。在本工作中，由于基于大语言模型的算子执行具有固有的不可预测性和不确定性，Sema 在执行器中采用 AQE，以探索成本、延迟与结果质量之间的权衡。

9 结论

本文介绍了 Sema，这是一个高性能系统，它将语义算子集成到数据库系统中，用于处理基于大语言模型的语义查询。为优化此类语义查询，Sema 在优化器阶段针对语义算子引入了自然语言表达式优化，并在执行器阶段采用运行时优化策略。考虑到大语言模型固有的不确定性，Sema 采用自适应查询执行框架，在执行过程中实现质量、成本和延迟之间的平衡。我们的结果展现了将语义推理与传统数据管理相结合，以实现更自适应、更智能的查询处理的潜力。系统 Sema 的可执行版本已发布于 https: //github.com/BITQiKangK/SemaSystem.git。

参考文献

[1] [无日期]。Amazon Redshift ML。https://aws.amazon.com/redshift/features/redshift- ml/。访问日期：2025-06-06。[2] [无日期]。BigQuery ML。https://cloud.google.com/bigquery/docs/bqml- introduction。访问日期：2025-06-06。[3] [无日期]。Oracle Data Mining。https://www.oracle.com/big-data/technologies/ dataminer/。访问日期：2025-06-06。[4] Kerem Akillioglu、Anurag Chakraborty、Sairaj Voruganti 和 M. Tamer Özsu。2025。面向大语言模型查询的关系数据库管理系统研究挑战。arXiv:2508.20912 [cs.DB] https://arxiv.org/abs/2508.20912[5] Eric Anderson、Jonathan Fritz、Austin Lee、Bohou Li、Mark Lindblad、Henry Lindeman、Alex Meyer、Parth Parmar、Tanvi Ranade、Mehul A. Shah、Benjamin Sowell、Dan Tecuci、Vinayak Thapliyal 和 Matt Welsh。2025。大语言模型驱动的非结构化分析系统的设计。发表于 第*15*届创新数据系统研究会议（*CIDR 2025*），荷兰阿姆斯特丹，*2025*年*1*月*19*日至*22*日。www.cidrdb.org。https://vldb.org/cidrdb/2025/the-design-of-an-llm-powered-unstructured-analytics-system.html[6] Ron Avnur 和 Joseph M. Hellerstein。2000。Eddies：持续自适应查询处理。发表于 *2000*年 *ACM SIGMOD*国际会议论文集

数据管理，*2000*年*5*月*16*日至*18*日，美国德克萨斯州达拉斯，Weidong Chen、Jeffrey F. Naughton 和 Philip A. Bernstein（编）。ACM，261–272。doi:10.1145/342009.335420[7] Jaeho Bang、Gaurav Tarlok Kakkar、Pramod Chunduri、Subrata Mitra 和 Joy Arulraj。2023。SEIDEN：重新审视视频数据库系统中的查询处理。*VLDB Endow.*论文集 16, 9 2023 2289 2301 10 14778 3598581 3598599 ( )，–。doi: . / .[8] Pedro Bizarro 、 Shivnath Babu 、 David J. DeWitt 和 Jennifer Widom。2005。基于内容的路由：针对不同数据采用不同计划。载于第*31*届超大型数据库国际会议论文集，挪威特隆赫姆，*2005*年*8*月*30*日至*9*月*2*日，Klemens Böhm、Christian S. Jensen、Laura M. Haas、Martin L. Kersten、Per-Åke Larson 和 Beng Chin Ooi（编）。ACM，757–768。http://www.vldb.org/archives/website/2005/program/paper/thu/p757-bizarro.pdf[9] Matthias Boehm、Berthold Reinwald、Dylan Hutchison、Prithviraj Sen、Alexan- dre V. Evfimievski 和 Niketan Pansare。2018。优化 SystemML 中的大规模机器学习算子融合计划。*VLDB Endow.*论文集 11，12 (2018 1755 1768 10 14778 3229863 3229865)，–。doi: . / .[10] Vinayak R. Borkar、Yingyi Bu、Michael J. Carey、Joshua Rosen、Neoklis Polyzotis、Tyson Condie、Markus Weimer 和 Raghu Ramakrishnan。2012。面向大规模机器学习的声明式系统。*IEEE* 数据工程通报 35，2（2012），24–32。http://sites.computer.org/debull/A12june/declare.pdf[11] Zhuhua Cai、Zografoula Vagena、Luis Leopoldo Perez、Subramanian Arumugam、Peter J. Haas 和 Christopher M. Jermaine。2013。使用 SimSQL 模拟数据库值马尔可夫链。载于*ACM SIGMOD*国际数据管理会议论文集，*SIGMOD 2013*，*27*，*2013*，*22* 纽约州纽约市，美国，*6*月-Kenneth A. Ross、Divesh Srivastava 和 Dimitris Papadias（编）。ACM，637–648。doi:10.1145/2463676.2465283[12] Jiashen Cao、Karan Sarkar、Ramyad Hadidi、Joy Arulraj 和 Hyesoon Kim。2022。FiGO：视频分析中的细粒度查询优化。载于*SIGMOD ' 22*：国际数据管理会议，美国宾夕法尼亚州费城，*2022*年*6*月*12*日至*17*日，Zachary G. I ves、Angela Bonifati 和 Amr El Abbadi（编）。ACM，559–572。doi:10.1145/3514221.3517857[13] Surajit Chaudhuri、Vivek Narasayya 和 Sunita Sarawagi。2002。含挖掘谓词的查询的高效求值。载于第*18*届国际数据工程会议论文集。IEEE，529–540。[14] Hanjun Dai、Bethany Wang、Xingchen Wan、Bo Dai、Sherry Yang、Azade Nova、Pengcheng Yin、Mangpo Phothilimthana、Charles Sutton 和 Dale Schuurmans。2024。UQE：非结构化数据库的查询引擎。神经信息处理系统进展 37（2024），29807–29838。[15] Naihao Deng、Yulong Chen 和 Yue Zhang。2022。文本到 SQL 的最新进展：现状与展望综述。载于*COLING*。国际计算语言学委员会，韩国庆州。https: //aclanthology.org/events/coling-2022/[16] Amol Deshpande 和 Samuel Madden。2006。MauveDB：在数据库系统中支持基于模型的用户视图。载于*ACM SIGMOD*国际数据管理会议论文集，美国伊利诺伊州芝加哥，*2006*年*6*月*27*日至*29*日，Surajit Chaudhuri、Vagelis Hristidis 和 Neoklis Polyzotis（编）。ACM，73–84。doi:10.1145/1142473.1142483[17] Dujian Ding、Sihem Amer-Yahia 和 Laks V. S. Lakshmanan。2022。关于机器学习模型上的高效近似查询。*VLDB Endow.*论文集 16，4 (2022 918 931 10 14778 3574245 3574273)，–。doi: . / .[18] Anas Dorbani、Sunny Yasser、Jimmy Lin 和 Amine Mhedhbi。2025。超越嘎嘎声：将语言模型和 RAG 深度集成到 DuckDB 中。*VLDB Endow.*论文集 18, 12 2025 5415 5418 10 14778 3750601 3750685 ( )，–。doi: . / .[19] Xixuan Feng、Arun Kumar、Benjamin Recht 和 Christopher Ré。2012。。迈向 RDBMS 内分析的统一架构。载于*ACM SIGMOD*国际数据管理会议论文集，*SIGMOD 2012*，美国亚利桑那州斯科茨代尔，*2012*年*5*月*20*日至*24*日，K. Selçuk Candan、Yi Chen、Richard T. Snodgrass、Luis Gravano 和 Ariel Fuxman（编）。ACM，325 336 10 1145 2213836 2213874 –。doi: . / .[20] Joseph M. Hellerstein、Christopher Ré、Florian Schoppmann、Daisy Zhe Wang、Eugene Fratkin、Aleksander Gorajek、Kee Siong Ng、Caleb Welton、Xixuan Feng、Kun Li 和 Arun Kumar。2012。MADlib 分析库，亦称 MAD Skills，即 SQL。*VLDB Endow.*论文集 5, 12 2012 1700 1711 10 14778 2367502 2367510 ( )，–。doi: . / .[21] Joseph M Hellerstein 和 Michael Stonebraker。1993。。谓词迁移：优化含昂贵谓词的查询。载于*1993*年*ACM SIGMOD*数据管理国际会议论文集。267–276。[22] Ravi Jampani、Fei Xu、Mingxi Wu、Luis Leopoldo Perez、Chris Jermaine 和 Peter J. Haas。2011。蒙特卡罗数据库系统：贴近数据的随机分析。*ACM*数据库系统汇刊 36，3（2011），18:1–18:41。doi:10.1145/ 2000824.2000828[23] David Justen、Daniel Ritter、Campbell Fraser、Andrew Lamb、Nga Tran、Allison Lee、Thomas Bodner、Mhd Yamen Haddad、Steffen Zeuch、Volker Markl 和 Matthias Boehm。2024。POLAR：通过最小阻力计划进行自适应、非侵入式连接顺序选择。*VLDB Endow.*论文集 17，6（2024），1350–1363。doi:10. 14778/3648160.3648175[24] Gaurav Tarlok Kakkar、Jiashen Cao、Aubhro Sengupta、Joy Arulraj 和 Hyesoon Kim。2025。Aero：机器学习查询的自适应查询处理。*ACM*管理论文集

数据 3, 3 2025 174 1 174 27 10 1145 3725408 ( ), : – : . doi: . / Daniel Kang、Peter Bailis 和 Matei Zaharia。。BlazeIt：为基于神经网络的视频分析优化声明式聚合与限制查询。VLDB Endow.论文集 [25] 2019 13, 4 2019 533 546 10 14778 3372716 3372725 ( ), – 。doi: . / . Daniel Kang、Edward Gan、Peter Bailis、Tatsunori Hashimoto 和 Matei Zaharia。2020。利用代理实现带保证的近似选择。VLDB Endow.论文集13，11（2020），1990–2003。http://www.vldb.org/pvldb/vol13/p1990-kang.pdf[26] Daniel Kang、John Guibas、Peter Bailis、Tatsunori Hashimoto、Yi Sun 和 Matei Zaharia。2021。利用昂贵谓词加速近似聚合查询。VLDB Endow.论文集14，11（2021），2341–2354。doi:10.14778/3476249. 3476285[27] Daniel Kang、Ankit Mathur、Teja Veeramacheneni、Peter Bailis 和 Matei Zaharia。2020。联合优化基于深度神经网络的可视分析中的预处理与推理。VLDB Endow.论文集 [28] 14, 2 2020 87 100 10 14778 3425879 3425881 ( ), – 。doi: . / . Mehmet Levent Koc 和 Christopher Ré。。使用关系数据库管理系统增量维护分类。VLDB Endow.论文集4，5（2011），302–313。doi:10.14778/1952376.1952380[29] 2011 [30] Woosuk Kwon、Zhuohan Li、Siyuan Zhuang、Ying Sheng、Lianmin Zheng、Cody Hao Yu、Joseph Gonzalez、Hao Zhang 和 Ion Stoica。2023。利用 PagedAttention 高效管理大语言模型服务的内存。载于 Proc. SOSP。ACM, 611 626 10 1145 3600006 3613165 – 。doi: . / . Viktor Leis、Peter A. Boncz、Alfons Kemper 和 Thomas Neumann。2014。基于 morsel 的并行性：面向多核时代的 NUMA 感知查询求值框架。载于国际数据管理会议，SIGMOD 2014，美国犹他州斯诺伯德，2014年6月22日至27日，Curtis E. Dyreson、Feifei Li 和 M. Tamer Özsu（编）。ACM，743–754。doi:10.1145/2588555.2610507[31] [32] Jinyang Li、Binyuan Hui、Ge Qu、Jiaxi Yang、Binhua Li、Bowen Li、Bailin Wang、Bowen Qin、Ruiying Geng、Nan Huo、Xuanhe Zhou、Chenhao Ma、Guoliang Li、Kevin Chen-Chuan Chang、Fei Huang、Reynold Cheng 和 Yongbin Li。2023。LLM 是否已能充当数据库接口？面向大规模数据库基础文本转SQL的 BIg Bench。载于神经信息处理系统进展36：神经信息处理系统年会2023，NeurIPS 2023，美国路易斯安那州新奥尔良，2023年12月10日至16日，Alice Oh、Tristan Naumann、Amir Globerson、Kate Saenko、Moritz Hardt 和 Sergey Levine（编）。http://papers.nips.cc/paper_files/paper/2023/hash/83fc8fab1710363050bbd1d4b8cc0021-Abstract-Datasets and Benchmarks.html [33] Jianzhe Lin、Maurice Diesendruck、Liang Du 和 Robin Abraham。2024。Batch-Prompt：以少博多。载于第十二届国际学习表征会议。https://openreview.net/forum?id=Agyicd577r[34] Yiming Lin、Madelon Hulsebos、Ruiying Ma、Shreya Shankar、Sepanta Zeighami、Aditya G. Parameswaran 和 Eugene Wu。2024。借助大语言模型实现准确、高效的文档分析。CoRR abs/2405.04674（2024）。doi:10.48550/ARXIV.2405.04674 arXiv:2405.04674[35] Chunwei Liu、Matthew Russo、Michael Cafarella、Lei Cao、Peter Baile Chen、Zui Chen、Michael Franklin、Tim Kraska、Samuel Madden、Rana Shahout 等。2025。Palimpzest：通过声明式查询处理优化人工智能驱动的分析。载于创新数据库研究会议论文集（CIDR）。2。[36] Shu Liu、Asim Biswal、Amog Kamsetty、Audrey Cheng、Luis Gaspar Schroeder、Liana Patel、Shiyi Cao、Xiangxi Mo、Ion Stoica、Joseph E. Gonzalez 和 Matei Zaharia。2025。优化关系数据分析工作负载中的大语言模型查询。载于第八届机器学习与系统会议。https://openreview.net/ forum?id=R7bK9yycHp [37] Yao Lu、Aakanksha Chowdhery、Srikanth Kandula 和 Surajit Chaudhuri。2018。利用概率谓词加速机器学习推理。载于2018年国际数据管理会议论文集，SIGMOD 2018，美国德克萨斯州休斯敦，2018年6月10日至15日，Gautam Das、Christopher M. Jermaine 和 Philip A. Bernstein（编）。ACM，1493–1508。doi:10.1145/3183713.3183751[38] Shangyu Luo、Zekai J. Gao、Michael N. Gubanov、Luis Leopoldo Perez 和 Christopher M. Jermaine。2018。关系数据库系统上的可扩展线性代数。SIGMOD Rec. 47, 1 2018 24 31 10 1145 3277006 3277013 ( ), – 。doi: . / . B.W. Matthews。。T4噬菌体溶菌酶二级结构的预测结果与观测结果比较。生物化学与生物物理学报（BBA）- 蛋白质结构 [39] 1975 405, 2 1975 442 451 10 1016 0005 2795 75 90109 9 ( ), – 。doi: . / - ( ) - Thomas Neumann。。面向现代硬件，高效编译高效查询计划。VLDB Endow.论文集4，9（2011），539–550。doi:10.14778/2002938.2002940[40] [41] Kwanghyun Park、Karla Saur、Dalitso Banda、Rathijit Sen、Matteo Interlandi 和 Konstantinos Karanasos。2022。机器学习预测查询的端到端优化。载于SIGMOD '22：国际数据管理会议，美国宾夕法尼亚州费城，2022年6月12日至17日，Zachary G. Ives、Angela Bonifati 和 Amr El Abbadi（编）。ACM，587–601。doi:10.1145/3514221.3526141[42] Liana Patel、Siddharth Jha、Melissa Pan、Harshit Gupta、Parth Asawa、Carlos Guestrin 和 Matei Zaharia。2025。语义算子及其优化：在 LOTUS 中实现带有准确性保证的基于大语言模型的数据处理。VLDB基金会论文集18，11（2025），4171–4184。[43] Mark Raasveldt 和 Hannes Mühleisen。2019。DuckDB：可嵌入式分析数据库。载于2019年国际数据管理会议论文集，SIGMOD 2019，荷兰阿姆斯特丹，2019年6月30日至7月5日，*2019*，Peter A. Boncz、Stefan Manegold、Anastasia Ailamaki、Amol Deshpande 和 Tim Kraska（编）。ACM，1981–1984。doi:10.1145/3299869.3320212[44] Francisco Romero、Johann Hauswald、Aditi Partap、Daniel Kang、Matei Zaharia 和 Christos Kozyrakis。2022。通过声明式模型关系优化视频分析。*VLDB Endow.* 论文集 16, 3 (2022), 447–460。doi:10.14778/ 3570690.3570695[45] Matthew Russo、Tatsunori Hashimoto、Daniel Kang、Yi Sun 和 Matei Zaharia。2023。在非结构化数据流上加速聚合查询。*VLDB Endow.* 论文集 16, 11 (2023), 2897–2910。doi:10.14778/3611479.3611496[46] Matthew Russo、Sivaprasad Sudhir、Gerardo Vitagliano、Chunwei Liu、Tim Kraska、Samuel Madden 和 Michael Cafarella。2025。Abacus：面向语义算子系统的基于成本的优化器。*arXiv*预印本 *arXiv:2505.14661* (2025)。[47] Shreya Shankar、Bhavya Chopra、Mawil Hasan、Stephen Lee、Björn Hartmann、Joseph M. Hellerstein、Aditya G. Parameswaran 和 Eugene Wu。2025。使用 DocWrangler 引导语义数据处理。*CoRR* abs/2504.14764 (2025)。doi:10.48550/ARXIV.2504.14764 arXiv:2504.14764[48] Shreya Shankar、Aditya G. Parameswaran 和 Eugene Wu。2024。DocETL：面向复杂文档处理的智能体查询重写与评估。*CoRR* abs/ 2410 12189 2024 10 48550 2410 12189 2410 12189 。( )。doi: 。/ARXIV. [49] 。arXiv: 。Gemma Team、Aishwarya Kamath、Johan Ferret、Shreya Pathak、Nino Vieil- lard、Ramona Merhej、Sarah Perrin、Tatiana Matejovicova、Alexandre Ramé、Morgane Rivière、Louis Rouillard、Thomas Mesnard、Geoffrey Cideron、Jean bastien Grill、Sabela Ramos、Edouard Yvinec、Michelle Casbon、Etienne Pot、Ivo Penchev、Gaël Liu、Francesco Visin、Kathleen Kenealy、Lucas Beyer、Xiaohai Zhai、Anton Tsitsulin、Robert Busa-Fekete、Alex Feng、Noveen Sachdeva、Ben- jamin Coleman、Yi Gao、Basil Mustafa、Iain Barr、Emilio Parisotto、David Tian、Matan Eyal、Colin Cherry、Jan-Thorsten Peter、Danila Sinopalnikov、Surya Bhu- patiraju、Rishabh Agarwal、Mehran Kazemi、Dan Malkin、Ravin Kumar、David Vilar、Idan Brusilovsky、Jiaming Luo、Andreas Steiner、Abe Friesen、Abhanshu Sharma、Abheesht Sharma、Adi Mayrav Gilady、Adrian Goedeckemeyer、Alaa Saade、Alex Feng、Alexander Kolesnikov、Alexei Bendebury、Alvin Abdagic、Amit Vadi、András György、André Susano Pinto、Anil Das、Ankur Bapna、Antoine Miech、Antoine Yang、Antonia Paterson、Ashish Shenoy、Ayan Chakrabarti、Bilal Piot、Bo Wu、Bobak Shahriari、Bryce Petrini、Charlie Chen、Charline Le Lan、Christopher A. Choquette-Choo、CJ Carey、Cormac Brick、Daniel Deutsch、Danielle Eisenbud、Dee Cattle、Derek Cheng、Dimitris Paparas、Divyashree Shiv- akumar Sreepathihalli、Doug Reid、Dustin Tran、Dustin Zelle、Eric Noland、Er- win Huizenga、Eugene Kharitonov、Frederick Liu、Gagik Amirkhanyan、Glenn Cameron、Hadi Hashemi、Hanna Klimczak-Plucińska、Harman Singh、Harsh Mehta、Harshal Tushar Lehri、Hussein Hazimeh、Ian Ballantyne、Idan Szpek- tor、Ivan Nardini、Jean Pouget-Abadie、Jetha Chan、Joe Stanton、John Wieting、Jonathan Lai、Jordi Orbay、Joseph Fernandez、Josh Newlan、Ju yeong Ji、Jyotinder Singh、Kat Black、Kathy Yu、Kevin Hui、Kiran Vodrahalli、Klaus Greff、Linhai Qiu、Marcella Valentine、Marina Coelho、Marvin Ritter、Matt Hoffman、Matthew Watson、Mayank Chaturvedi、Michael Moynihan、Min Ma、Nabila Babar、Natasha Noy、Nathan Byrd、Nick Roy、Nikola Momchev、Nilay Chauhan、Noveen Sachdeva、Oskar Bunyan、Pankil Botarda、Paul Caron、Paul Kishan Rubenstein、Phil Culli- ton、Philipp Schmid、Pier Giuseppe Sessa、Pingmei Xu、Piotr Stanczyk、Pouya Tafti、Rakesh Shivanna、Renjie Wu、Renke Pan、Reza Rokni、Rob Willoughby、Rohith Vallu、Ryan Mullins、Sammy Jerome、Sara Smoot、Sertan Girgin、Shariq Iqbal、Shashir Reddy、Shruti Sheth、Siim Põder、Sijal Bhatnagar、Sindhu Raghuram Panyam、Sivan Eiger、Susan Zhang、Tianqi Liu、Trevor Yacovone、Tyler Liechty、Uday Kalra、Utku Evci、Vedant Misra、Vincent Roseberry、Vlad Feinberg、Vlad Kolesnikov、Woohyun Han、Woosuk Kwon、Xi Chen、Yinlam Chow、Yuvein Zhu、Zichuan Wei、Zoltan Egyed、Victor Cotruta、Minh Giang、Phoebe Kirk、Anand Rao、Kat Black、Nabila Babar、Jessica Lo、Erica Moreira、Luiz Gustavo Martins、Omar Sanseviero、Lucas Gonzalez、Zach Gleicher、Tris Warkentin、Vahab Mirrokni、Evan Senter、Eli Collins、Joelle Barral、Zoubin Ghahramani、Raia Hadsell、Yossi Matias、D. Sculley、Slav Petrov、Noah Fiedel、Noam Shazeer、Oriol Vinyals、Jeff Dean、Demis Hassabis、Koray Kavukcuoglu、Clement Farabet、Elena Buchatskaya、Jean-Baptiste Alayrac、Rohan Anil、Dmitry、Lepikhin、Sebastian Borgeaud、Olivier Bachem、Armand Joulin、Alek Andreev、Cassidy Hardin、Robert Dadashi 和 Léonard Hussenot。2025。Gemma 3 技术报告。arXiv:2503.19786 [cs.CL] https://arxiv.org/abs/ 2503 19786 。Jiayi Wang 和 Jianhua Feng。2025。Unify：一个非结构化数据分析系统。载于第*41*届 *IEEE* 国际数据工程会议（*ICDE 2025*），香港，*2025*年*5*月*19*日至*23*日。IEEE，4662–4674。doi:10.1109/ICDE65448.2025.00374[50] [51] Jason Wei、Xuezhi Wang、Dale Schuurmans、Maarten Bosma、Brian Ichter、Fei Xia、Ed H. Chi、Quoc V. Le 和 Denny Zhou。2022。思维链提示可激发大语言模型的推理能力。载于神经信息处理系统进展第*35*卷：神经信息处理系统年度会议*2022*（*NeurIPS 2022*），美国路易斯安那州新奥尔良，*2022*年*11*月*28*日至*12*月*9*日。http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html[52] Ziyang Xiao、Dongxiang Zhang、Zepeng Li、Sai Wu、Kian-Lee Tan 和 Gang Chen。2023。DoveDB：一种声明式低延迟视频数据库。*VLDB Endow.* 论文集 16, 12 (2023), 3906–3909。doi:10.14778/3611540.3611582

[53] Zhihui Yang、Zuozhi Wang、Yicong Huang、Yao Lu、Chen Li 和 X. Sean Wang。2022。使用相关代理模型优化机器学习推理查询。*VLDB Endow.* 论文集 15, 10 (2022), 2032–2044。doi:10.14778/3547305. 3547310[54] Enhao Zhang、Nicole Sullivan、Brandon Haynes、Ranjay Krishna 和 Magdalena Balazinska。2025。利用大语言模型处理组合事件的自增强视频数据管理系统。*ACM*数据管理论文集 3, 3 (2025), 215:1–215:29。doi:10.1145/3725352[55] Kangfei Zhao、Jeffrey Xu Yu、Yu Rong、Ming Liao 和 Junzhou Huang。2021。探索在关系数据库管理系统中使用 SQL 实现期望最大化。载于高级数据库应用系统——第*26*届国际会议，*DASFAA 2021*，台北，台湾，*2021*年*4*月*11*日至*14*日，论文集，第二部分（计算机科学讲义，第*12682*卷）。Springer，778–794。doi:10.1007/978-3-030-73197-7_53

A 示例查询

```
-- 从内容字段中提取关键词
SELECT id, s'从{
    content}' AS 提取的_关键词
FROM 文档;
-- 创建语义摘要 来自多个
    字段
SELECT s'基于{title 的事件摘要
    }和{description}'
作为 事件_摘要 来自 新闻_文章;
```

图9：语义投影示例

```
-- 根据某个字段的情感筛选行
    特定字段
SELECT *FROM 产品_评价
WHERE s'这条{评价_文本}表达了正面
    情感';
```

图10：语义筛选示例

```
-- 分析会议论文标题的语言风格WITH title_style AS(SELECT
Title, s'分析论文标题{Title}的语言风格特征。描述其关
键特征，例如：正式程度、技术术语的使用、长度、结构和用
词。'AS style_featuresFROM Paper WHERE ConferenceId !
= 0 ANDTitle ISNOT NULL)SELECT sem_agg(s'根据这些会议
论文标题{Title}及其风格分析{style_features}，总结会议
论文标题的整体语言风格和术语特征。')FROM title_style;
```

图11：语义聚合示例

算法2:自然语言表达简化

输入： 用户输入自然语言表达式
输出： 已处理的自然语言表达式

1 约简状态 ←AssessDbReduction(用户输入自然语言表达式)
2 **如果** 约简状态 ="*not _reducible*" **则**
3 **返回** 已处理的自然语言表达式 ← cleanedNLE
4 已处理的自然语言表达式 ←表达式约简(用户输入的自然语言表达式)
5 **返回**已处理的自然语言表达式

算法 3： 表达式探索与成对相似度估计
度估计

输入： $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$, 样本块 D_1
输出： 选择率 s，结果向量 R，相似度矩阵 MCC

1 **对于** $i \leftarrow 1$ 至 n **执行**
2 $R[i] \leftarrow$ 执行谓词(σ_i, D_1);
3 $S[i] \leftarrow \frac{\text{计数真值}(R[i])}{|\mathcal{D}_1|}$;
4 **对于** $i \leftarrow 1$ 至 n **执行**
5 **对于** $j \leftarrow i+1$ 至 n **执行**
6 $(TP, TN, FP, FN) \leftarrow$混淆计数$(R[i], R[j])$;
7 MCC$[i][j] \leftarrow$马修斯(TP, TN, FP, FN);
8 **返回** S $, R,$ MCC

```
-- 根据文本字段的语义相似度连接两个表 SELECT * FROM
news_articles a JOIN blog_posts b ON s'主题{a.headline}
与{b.title}相关'; -- 同时包含句法和语义条件的复杂混合连接
SELECT e.employee_id, eh.manager_id FROM 员工 e
INNER JOIN employee_hierarchy eh ON e.manager_id =
eh.employee_id AND s'{e.title}是一个技术岗位，且层级
{eh.level} < 3';
```

图12：语义连接示例

B 算法

算法3在采样数据块（D_1）上评估每个谓词（$\sigma_i \in \Sigma$），以生成布尔结果向量 $R[i]$，并据此将选择率 $S[i]$ 估计为满足该谓词的元组所占比例（第1-3行）。利用这些结果向量，该算法随后通过为每对谓词推导混淆矩阵，并应用马修斯相关系数（MCC）来计算成对谓词相似度，从而填充相似度矩阵（第4-7行）。

算法4通过谓词相关性引导的动态剪枝生成候选执行路径集。给定一组谓词 Σ，该算法首先按照选择率升序排列所有谓词，以构建基线顺序执行路径（第1行），并将其纳入候选集 P（第2行）。随后，算法使用预先计算的

算法4： 采用动态剪枝

输入： S，MCC，Σ，μ_{thresh}
输出： 候选路径 P

$p_{\text{sorted}} \leftarrow$ 按选择率升序排序(Σ, S);
P $\leftarrow$ {序列路径(p_{sorted}) };
$\mathcal{F} \leftarrow \emptyset$;
对 $i \leftarrow 1$ 至 n **执行**
  对 $j \leftarrow i+1$ 至 n **执行**
    若 MCC$[i][j] > \mu_{thresh}$ **则**
      F $\leftarrow$F $\cup\{(\sigma_i, \sigma_j)\}$;
对每个 $(\sigma_a, \sigma_b) \in \mathcal{F}$ **执行**
  $\sigma_a \oplus \sigma_b \leftarrow$ 融合(σ_a, σ_b);
  P $\leftarrow$P $\cup$ {SeqPath(单次替换$(p_{\text{sorted}}, \{\sigma_a, \sigma_b\} \rightarrow \sigma_a \oplus \sigma_b)$) };
返回$\mathcal{P}$

算法5： 路径探索与最佳路径选择

输入： 路径 P，样本块 D_2, 阈值 τ_{acc}
输出： 最佳路径 $p^\star$

执行参考路径以获取真实值 G ;
对于每个 $p \in \mathcal{P}$ **执行**
  (延迟, 成本, 选择) $\leftarrow$运行(p, D_2);
  准确率 $\leftarrow$ 准确率(选择, G);
F $\leftarrow$ 帕累托前沿(P, lat, cost);
$p^\star \leftarrow \arg\min_{p \in \mathcal{F}} M[p]$.lat;
返回 $p^\star$

马修斯相关系数（MCC）矩阵检查谓词对之间的相关性。它仅保留相关性超过阈值 μ_{thresh} 的谓词对（第4—7行），从而有效剪除关联较弱的组合。对于每个保留的谓词对（σ_i、σ_j），算法应用融合函数形成融合谓词 $\sigma_i \oplus \sigma_j$，并通过在排序序列中将原谓词的一次出现替换为融合谓词来生成新的候选路径，同时保持其余谓词的相对顺序不变（第8—10行）。所有生成的路径都会被收集到P中，并作为后续路径探索阶段的候选路径。

算法5探索候选执行路径，并选择最适合利用阶段的路径。给定候选路径集合 P 和一个采样数据块 D_2,，算法首先执行参考路径 p_{ref}，以获得真实结果（第 1 行）。随后，在 D_2 上执行每条候选路径，测量其延迟、执行成本和选择输出，并据此与真实值进行比较，计算准确率（第 2–4 行）。算法根据收集到的延迟和成本指标构建帕累托前沿，仅保留非支配路径（第 5 行）。在这些帕累托最优候选项中，选择延迟或成本最低的路径作为最终最佳路径 $p^\star$，并将其返回以供执行（第 6–7 行）。

C 数据集与基准查询

表 8：查询集概况

类别	子类别	查询 ID
单一 *SemFilter*/*SemProj*	单一算子	Q1–Q5
连续的 *SemFilter*	筛选 → 筛选	Q6–Q10
SemProj-优先	投影 → 筛选	Q11–Q13
	投影 →投影	Q14–Q15
SemFilter-优先	筛选 → 投影	Q16–Q18
	筛选 → 聚合	Q19–Q20

```
SELECT id FROM(SELECT TweetID AS id , 情感 , text
FROMTwitter LIMIT32768)WHERE s'情感标签规则如下：- 情感
 > 0 -> 正面 - 情感 = 0 -> 中性 - 情感 < 0 -> 负面
任务：判断 { sentiment } 标签是否与 {text} 内容一致。');
```

图13：问题1：情感分析

```
SELECT s'分类 给定的 {" 产品名称"}
    分为以下类别中的恰好一类：
[家具, 办公用品, 技术 ]。
仅输出类别名称, 不要
    解释, 不要引号, 也不要多余的词
    .
示例：
输入：Hon 金属书柜, 黑色输出：
    家具
输入：AT&T CL2909 输出：技术
输入：Belkin F9S820V06 8插孔电涌保护器 输出：
    办公用品' 作为 预测的_类别 来自
     产品;
```

图14：问题2：将产品归为三个类别之一

表7：实验中使用的数据集和查询

数据集	表数量	表（行数）	属性	查询
Appstore	2	用户_评价(10,840), Play 商店(64,286)	用户评论.翻译后的评论_ _	Q7、Q8
芝加哥_犯罪	1	IUCR (401)	主要_描述, 次要_描述, 索引_代码	Q3
Superstore	1	产品 (5,298)	产品名称, 类别	Q2
食品_检查	1	违规事项 (36,050)	描述, 风险_类别	Q4
食品_检查_2	1	违规 (525,709)	检查员_意见，罚款	Q5
社交媒体_	3	Twitter (99,901), 用户 (99,260), 位置(6,211)	Twitter 文本	Q1, 问题9, Q16–Q18
电影 4_	7	电影 (4,627), 国家(88), 制作_国家 (6,436), 类型 (20), 电影_类型 (12,160),语言 (88), 电影_语言 (11,740)	电影简介	Q10, Q13–Q15
CoinMarketCap	2	币种(8,927), 历史(4,441,972)	币种.描述	Q6, Q11, Q12
Authors	5	论文 (2,254,920),期刊 (15,151), 会议 (4,545), 作者(247,030), PaperAuthor(2,315,574)	论文关键词, 论文标题	Q19、Q20

```
SELECT iucr_no FROMIUCRWHEREs'索引_代码标签规则如下：- 索引_代码 = I -> 已索引（严重，例如谋杀、强奸、纵火和抢劫）- 索引_代码 =N-> 未索引（较轻，例如故意破坏、武器违规和扰乱治安）任务：判断该犯罪''的 { primary_描述 } 与 { secondary_描述 } 是否表明该犯罪与 { index_代码 } 一致');
```

图15：第三季度：确定犯罪描述是否与索引代码一致

```
SELECT s'任务：将给定的违规事项 { description } 准确分类为以下类别之一：- 低风险 - 中等风险 - 高风险 仅输出风险类别，不要解释，不要加引号，也不要添加其他文字。示例：输入：洗手设施不足且无法使用。输出：中等风险 输入：地板、墙壁或天花板不干净或破损。输出：低风险 输入：食品接触表面不干净或不卫生。输出：高风险 ' AS 谓词_类别，风险_类别 ASground_truth FROM违规事项;
```

图16：Q4：根据违规描述确定风险程度

```
SELECT inspection_id FROM ( SELECT distinct inspection_
id , inspector_comment , fine FROM violation WHERE
inspector_comment IS NOT NULL AND fine IN
 [100， 250， 500]LIMIT 16384 ) WHERE s'食品安全问题越
严重，罚款就越高。罚款标签规则如下：- 罚款 = 100 -> 轻
微 - 罚款 = 250 -> 严重 - 罚款 = 500 -> 危急 罚款示例:
1. { inspector_comment }：修理三格水槽右侧格的排水塞，使
其能够盛水。{fine }：100 2. { inspector_comment }：现场没
有化学检测试纸——必须提供。严重违规事项 7 -38 -030。
{fine }：250 3. { inspector_comment }：发现步入式冷藏室温
度为 45.3 华氏度。危急违规事项 .7 -38 -005( A) {fine }：500
任务：判断 { inspector_comment } 与 {fine } 是否一致。仅输
出 true 或 false。';
```

图 17：Q5：根据违规描述确定风险程度

```
SELECT 币种.id AS coin_id FROM 币种WHERE 类别 =
'coin' AND 描述 is NOTNULLAND s'{ description } 表明这种加
密货币的供应量很大。'AND s'{ description } 表明这种加密货
币仍在流通。'AND s'从 { description } 可以推断出这种加密货
币的价格正在上涨。'
```

图 18：Q6：查找仍在流通且供应量巨大、价格上涨的加密货币

```
SELECT ur.id AS id FROM 用户_评价 ur 连接 Play 商店 p
ON ur.应用 =p.应用 WHERE p. 类别 = 'TOOLS ' AND
s'{ur. Translated_评价 } 是一个完整的句子，并具有信息丰富
的内容。' AND s'{ur. Translated_评价 } 是一条正面用户评论。
' AND s'{ur. Translated_评价 } 表明这个应用是一个有用的工
具。'
```

图 19：Q7：查找工具类应用的正面、完整且信息丰富的用户评论，表明该应用是一个有用的工具

```
SELECT ur.id AS id FROM user_reviews ur JOIN playstore
p ON ur.App =p.App WHERE p. Category = 'GAME '
AND s'{ Translated_review } 是一个完整的句子，且包含信息
丰富的内容。' AND s'{ Translated_review } 是正面用户评论。
' AND s'{ Translated_Review } 表明用户使用此应用时体验良
好且乐在其中 ';
```

图 20：Q8：查找正面、完整且信息丰富的用户表明用户对应用体验良好、玩得开心的游戏类应用评价验并且觉得应用很有趣

```
SELECT ur.id AS id FROM user_评价 ur 连接 playstore p
ON ur.App =p.App WHERE p. 类别 = '家庭 ' AND s'{
Translated_评价 } 是一个完整的句子，且包含信息丰富的内
容 .' AND s'{ Translated_评价 } 是一条正面用户评论 .'
AND s'{ Translated_评价 } 表明该应用适合儿童。';
```

图21：问题9：查找正面、完整且信息丰富的家庭应用用户评论，这些评论表明该应用适合儿童

```
SELECT m.电影_idFROM 电影 mWHERE s'{m.overview}表明该电影是一部动作片。'AND s'{m.overview}表明该电影聚焦于一个核心角色。'AND s'{m.overview}表明该电影基于历史事件';
```

图 22：Q10：查找聚焦于核心角色且基于历史事件的动作片

```
SELECT coin_id FROM (SELECT s'根据{c.description}，介绍加密货币流通的情况' AS circulation, c.id AS coin_id FROM coins c JOIN historical h ON h.coin_id= c.id AND h.date ='2019-01-09' AND c.status IN ('untracked', 'inactive') AND c.description IS NOT NULL) WHERE s'{circulation}表明加密货币处于开发中或运行于测试网。';
```

图 23：Q11：查找在 2019 年 1 月 9 日处于非活跃或未跟踪状态、缺少价格数据，且其描述表明仍在开发中或运行于测试网的加密货币

```
SELECT coin_id FROM (SELECT s'从描述{description}中提取有关加密货币底层技术的信息，重点关注其中是否提及专有区块链、原生共识机制或独立网络。请简要总结这些技术特征。' AS tech_features, coins.id AS coin_id FROM coins c JOIN historical h ON h.coin_id= c.id WHERE c.platform_id IS NOT NULL AND c.description IS NOT NULL AND h.circulating_supply IS NOT NULL AND h.total_supply IS NOT NULL AND h.date ='2019-01-10') WHERE s'根据技术特征{tech_features}，判断该加密货币是否拥有自己的独立区块链或共识机制';
```

图 24：Q12：查找其技术特征摘要表明该加密货币拥有自有区块链或共识机制的币种

```
SELECT 电影 _标识符 来自
(SELECT s'总结 the 地理和时间
    背景 根据电影 to {
    剧情简介}。' 作为 背景, m. 电影_id 作为
    电影_id
    来自 电影 m
    半连接 连接 (
      SELECT mg. 电影_id
      来自 电影_类型 mg JOIN 类型 g位于 mg.
          类型_id = g. 类型_id
      WHERE g. 类型_name IN ('惊悚片', '
          喜剧片', '西部片')
    ) mg 连接 电影 m_编号 =电影 mg_编号
    WHERE 收入 >预算
      并且 长度（剧情简介） < 255
)
WHERE s'{ 背景 } 表示故事 设定
    处于现代，而非 历史
    时期。';
```

图 25：Q13：查找盈利丰厚的电影，这些电影的背景表明故事发生在现代，而非类型有限的历史时期

```
SELECT movie_id , s'将 {plot} 归入以下类别之一：[科幻片、
犯罪、惊悚片、爱情、剧情、恐怖、喜剧片、战争、奇幻、家
庭、动画、传记、其他]只输出最符合该电影的类别，不要添加
任何其他文字或推理内容。' AS category FROM ( SELECT s'概
述 { overview } 的主要剧情' AS plot , movie_id FROM
movie m SEMI JOIN ( SELECT ml. movie_id FROM movie_
languages ml JOIN language l ON ml. language_id = l.
language_id AND l. language_name = ' 英语 ' ) ml ON
m. movie_id = ml. movie_id AND movie_status = '已上
映 ' AND popularity > 10 AND length ( overview ) <255
AND length ( overview ) > 0);
```

图 26：Q14：总结英语上映的热门电影的主要剧情，并将剧情归类到不同类型

```
SELECT movie_id , s'这是一项多标签分类任务。根据{
conflict }，判断主人公与什么发生冲突，并按照“人物 vs X”的
格式进行分类，其中 X 是以下一个或多个类别： [人物、自我、
社会、自然、技术、机器、超自然、命运、其他]。仅输出 X,
并以逗号分隔（例如：自我、社会）。不要包含解释或推理。'
AS categories FROM ( SELECT movie_id , s'用一句话提取电
影{ overview }中的所有冲突。' AS conflict FROM movie
SEMI JOIN ( SELECT pc. movie_id FROM production_
country pc JOIN country c ON pc. country_id = c.
country_id AND c. country_iso_code = '美国 ') pc ON
movie . movie_id =pc. movie_id AND ( movie . release_
date BETWEEN '2000 -01 -01 ' AND '2015 -12 -31 ') AND
movie . vote_average > 6.0 AND length ( overview )< 255;
```

图 27：Q15：提取 2000 年至 2015 年间高评分电影中的所有冲突，并对冲突进行分类

```
SELECT s'从招聘推文 {t. text} 中提取提及的具体职位名称和所需技能。以逗号分隔的列表形式返回（例如：软件工程师、Python、AWS、机器学习）。重点关注具体的职位名称和技术技能' AS info ,t.TweetID AS Twitter_idFROM Twitter tJOIN 用户 uON t.UserID = u.UserIDWHERE user. Gender ='女性 'AND t. 情感 = 0.0AND t.TweetID IS NOT NULLAND Klout > 50AND s' 内容为 {text} 的推文涉及 AWS 或云计算相关的职位发布、招聘公告、招聘通知或职业机会（包含 hiring、job opening、career opportunity、we are looking for、join our team 等内容）。';
```

图28：Q16：筛选由女性发布的、与招聘相关的中性推文，并提取具体的职位名称和技术技能。

```
SELECT s'根据推文内容："{ text }"，识别其中表达的主要安全或隐私问题类型。将内容分类为数据泄露、未授权访问、加密弱点、合规违规、身份盗窃、服务漏洞、配置错误、监控问题或其他具体的安全／隐私问题类型。' AS 类型，twitter . TweetID AS twitter_idFROM twitterWHERE 情感 < -2.5 AND Klout > 20AND t.TweetID IS NOT NULLAND s'推文 "{ text }" 讨论了与 AWS服务相关的安全问题、隐私问题、数据保护、网络安全威胁、身份验证问题或机密性问题';
```

图29：Q17：查找与 AWS 服务潜在风险相关、影响力较大的负面推文，并识别安全或隐私问题的主要类型

```
SELECT s'从这条推文 "{t.text }" 中提取正在讨论的编程／开发主题或主要内容。返回简洁的主题标签，例如性能优化、学习资源、就业机会、技术问题、新功能等。' twitter . TweetID AS twitter_id FROM twitter t JOIN location l On t. LocationID = l. LocationID WHERE l. Country IN ('美国 ', '加拿大 ', '英国 ') AND 情感 >= 3.0 AND Reach >1000 AND t. TweetID IS NOT NULL s'筛选提及亚马逊云科技或云服务的推文 "{t.text }"，包括具体的 AWS产品或一般云计算问题';
```

图30：Q18：筛选与 AWS 产品相关的推文，并提取其中讨论的话题或主题

```
SELECT sem_agg（"总结以下关键词所反映的主要研究主题：{p. Keyword }。请用简洁、高层次的科学主题作答"，Keyword） FROM Paper p JOIN Journal j ON j.Id = P. JournalId WHERE p.Year BETWEEN 2000 AND 2011 AND j. ShortName IN（'TODS '，'VLDB '，'TKDE '，' JDM '，'DKE '）AND s'根据关键词"{p. Keyword }"，判断这篇论文是否与数据库系统、数据管理或查询处理相关';
```

图31：Q19：查找数据库期刊中与数据库系统、数据管理或查询处理相关的论文，并聚合这些论文的关键词，以找出主要研究主题

```
SELECT ConferenceId，sem_agg（'根据以下关键词识别并概述主要的跨学科研究趋势：{p. Keyword }。仅返回一个句子。', Keyword）AS trends FROM paper p JOIN Conference c ON p. ConferenceId = c.Id 半连接（SELECT DISTINCT pa. PaperId FROM PaperAuthor pa WHERE pa. AuthorId IS NOT NULL AND（LOWER（COALESCE（pa. Affiliation, ''））LIKE '% institute %' OR LOWER（pa. Affiliation）LIKE '% laboratory % ' OR LOWER（pa. Affiliation）LIKE ' % research center %'））filtered_papers ON p.Id = filtered_papers . PaperId AND p. Keyword IS NOT NULL AND length（p. Keyword） > 0WHERE s'判断关键词为 { Keyword } 的论文是否与跨学科方法相关。' GROUP BY ConferenceId HAVING COUNT（*） >= 3 AND trends IS NOT NULL
```

图 32：问题20：查找作者隶属于研究所、实验室或研究中心的论文，并识别跨学科研究趋势

```
-- 注：引入了冗余表达——重复引用字段（{ primary_description }、{ secondary_description }、{ index_code } 各出现两次）、冗余修饰语（“审查……后，再次检查以确保一致性”）以及复杂的句子结构。SELECT COUNT (*) FROM chicago_crime .IUCR WHERE s'索引代码标签规则如下：\n\ - 索引代码 _I -> 已索引（严重犯罪，例如谋杀、强奸、纵火和抢劫）\n\ - 索引代码 =N -> 未索引（较轻微的犯罪，例如故意破坏、武器违规和扰乱治安）\n\ 根据这些规则，应将主要描述为 { primary_description }、次要描述为 { secondary_description } 的犯罪与索引代码 { index_code } 进行核对；再次检查主要描述 { primary_description } 以确保一致性后，应根据既定的分类标准，判断该犯罪是否与指定的索引代码 { index_code } 一致。';
```

图 33：表达式压缩前的第三季度

```
SELECT iucr_不 FROMIUCRWHERES'索引_代码标签规则为：- 索引_代码 =I -> 已索引（严重犯罪，例如谋杀、强奸、纵火和抢劫）- 索引_代码 =N -> 未索引（较轻微犯罪，例如故意破坏、武器违规和扰乱治安）任务：判断犯罪''s{primary_描述 } 和 { secondary_描述 } 是否表明该犯罪符合 { index_代码 }');
```

图 34：部分推导前的第三季度

```
-- 注意：引入了重复的字段引用（{ description } 出现三次）和冗余的句子结构（重复出现“and the { description } indicates that ...”模式）。SELECT coins.id AS coin_id FROM coinsWHERE 类别 ='币种' AND 状态 ='活跃 'AND 描述为非空 AND s'{ description } 显示，这种加密货币的供应量很大；{ description } 表明，这种加密货币仍在流通；{ description } 则可用于推断，这种加密货币的价格正在上涨。';
```

图 35：表达式压缩前的 Q6

```
SELECT coins.id AS coin_id FROM coinsWHERE 类别 = 'coin' AND 描述 is NOTNULLAND s'{描述} 表明这种加密货币的供应量很大。'AND s'{描述} 表明这种加密货币仍在流通。'AND s'可以从{描述}推断出，这种加密货币的价格正在上涨。'
```

图 36：Q6 部分推导前

```
-- 注：添加冗余修饰语（“用于描述电影叙事”）、重复的字段引用（{ overview } 出现两次）以及复杂表达式（“在评价.....以寻找时间线索之后”）。SELECT movie_id FROM ( SELECT s'根据提供的电影概要 { overview }（用于描述电影叙事），总结电影的地理和时间背景，包括此概要 { overview } 中提到的具体地点和时间段。' AS background , movie_id FROM movie SEMI JOIN ( SELECT mg. movie_id FROM movie_genres mg JOIN genre g ON mg. genre_id = g. genre_id WHERE g. genre_name IN ('惊悚片 ', '喜剧片 ', '西部片 ') ) mg ON m. movie_id =mg. movie_id WHERE revenue > budget AND length ( overview ) < 255) WHERE s'背景摘要 { background } 表明故事设定在现代，而不是历史时期；在评价背景 { background } 以寻找时间线索之后，这一背景分类得到确认。';
```

图 37：表达式压缩前的 Q13

```
-- 注：将原本可用 SQL 表达的数值比较谓词 revenue > budget 替换为自然语言描述“{ revenue } 大于 { budget }”。
SELECT movie_id FROM ( SELECT s'根据 { overview } 总结电影的地理和时间背景。' AS background , movie_id FROM movie SEMI JOIN ( SELECT mg. movie_id FROM movie_genres mg JOIN genre g ON mg . genre_id =g. genre_id WHERE g. genre_name IN ('惊悚片 ', ' 喜剧片 ', '西部片 ') ) mg ON m. movie_id = mg. movie_id WHERE s'{ revenue } 大于 { budget }' AND length ( overview ) < 255) WHERE s'{ background } 表明故事设定在现代，而不是历史时期。';
```

图 38：谓词推导前的 Q13

```
-- 注：引入冗长的分类描述和重复的字段引用（{ plot}出现三次），并添加了不必要的解释性文本。 SELECT plot ,s'出于类型分类目的，应检查情节概要{plot}，并在对该剧情{plot}进行评价后，必须将其分配至预定义类别之一，包括：科幻片、犯罪惊悚片、爱情剧情片、恐怖、喜剧片、战争、奇幻、家庭动画片、传记，若均不匹配，则分类为其他。' AS category FROM ( SELECT s'在阅读提供该电影描述的电影概要{ overview }后，应以简洁方式对这一剧情简介{ overview }的主要情节点和关键叙事元素进行摘要。' AS plot FROM movie m SEMI JOIN ( SELECT ml. movie_id FROM movie_languages ml JOIN " language " l ON ml. language_id =l . language_id AND l. language_name ='英语 ' ) ml ON m. movie_id = ml. movie_id AND movie_status = '已上映 ' AND popularity > 10 AND length ( overview ) <255 AND length ( overview ) > 0);
```

图 39：表达式压缩前的 Q14

```
-- 注：将可用SQL表达的movie_status= '已上映 '替换为原本的自然语言描述'{ movie_status }已上映'。 SELECT plot ,s'将{plot}分类为以下之一：科幻片、犯罪、惊悚片、爱情、剧情、恐怖、喜剧片、战争、奇幻、家庭、动画、传记、其他' AS category FROM ( SELECT s'对{ overview }的主要剧情进行摘要' AS plot FROM movie m SEMI JOIN ( SELECT ml. movie_id FROM movie_languages ml JOIN " language " l ON ml. language_id =l . language_id AND l. language_name ='英语 ' ) ml ON m. movie_id = ml. movie_id AND s'{ movie_status }已上映' AND popularity > 10A ND length ( overview ) <255 AND length ( overview ) > 0);
```

图40：谓词演绎前的 Q14

用于（部分）谓词演绎的提示

你是一名数据库优化器助手。你的任务是根据提供的列统计信息，为自然语言表达式（NLEs）推导必要的 SQL 前置条件（下推谓词）。大语言模型会对 NLE 进行评估，以确定某一行是否满足特定条件。由于大语言模型评估成本高昂，我们希望先在 CPU 上执行简单的 SQL 谓词，筛选出那些可以确定为负面的行。重要提示：你的目标不是将 NLE 转换为等价的 SQL 语句。相反，你必须根据列统计信息，找出 NLE 谓词的必要条件。每列提供的统计信息包括：- 是否可为空 - 不同值的数量 - 频率最高的前五个值（从部分行中采样）步骤：1. 理解 NLE 的语义。2. 检查列统计信息，尤其是频率最高的值。3. 识别明确违反 NLE 谓词的值。例如，非数字、N/A、空字符串或 NULL 通常表示缺失数据。如果 NLE 谓词要求有效或有意义的值，则可以筛选掉这些缺失值。4. 构造用于筛选掉这些违规值的 DuckDB SQL 谓词。如果你能够推导出必要条件，请将其输出为有效 JSON 数组，其中每个字符串都必须是有效的 DuckDB SQL 谓词（例如，"col != 'nan'"）。最终筛选条件将是数组中所有字符串的逻辑与。如果无法推导出有意义的必要条件，只需输出空数组。示例：NLE 谓词：Translated_review is a positive user review 列统计信息：Translated_Review：nullable=true，distinct=10537，top5=["nan":711，"Good":13，"Negative":10，"Great":6，"Really good":5]输出：["Translated_Review != 'nan'"，"Translated_Review != 'Negative'"]注意：输出必须是有效的 JSON 字符串数组（或空数组）。

用于（部分）谓词演绎验证的提示

你是一名数据库优化器助手。你的任务
是验证每个候选 SQL 是否
谓词是一组的必要条件
自然语言表达式（NLEs）。

背景：
语义筛选会对每一行评估自然语言表达式（
自然语言条件）。由于大语言模型
评估成本高昂，，我们会预先筛选行，
使用成本低廉的 SQL 谓词。一个 SQL 谓词 P
只有当它是一个有效的下推过滤器，且满足
必要条件：每当 P 为假时，the
整体 NLE 过滤器必定为假。
关键概念——必要与充分：
对于每一列，某些值会使 NLE 为假（集合
A），某些值会使其为真（集合 B）。
- **必要条件（有效）**：对于
A 中的值，\"列 !{v1}值\"。筛除违反条件的值是
正确的。
- **充分条件（无效）**：\"列 = 值\" or
B 中的值，\"列 IN (...)\"。
将已知的好值列入白名单是错误的：它
会排除值不在
统计信息中的有效行。对于此类
谓词，输出 false。

必要条件的定义：
SQL 谓词 P 是 NLE
集合 S 的必要条件，当且仅当：
对于每一行可能的情况，非 P => （至少有一个
S 中的自然语言表达式为假）
等价地：如果某一行满足所有自然语言表达式，它就必须
也满足 P。
给定原始自然语言表达式和候选
SQL 谓词，输出一个 JSON 数组，其中包含
布尔值（长度与 SQL 谓词
列表相同）：
- true：该 SQL 谓词是有效的必要
条件（通常为“列 != 值”
筛除不符合条件的值）
- false：SQL 谓词无效（过于
严格，与自然语言表达式无关，或一个充分的
条件，如 \"列 = 值\" / \"列
IN (...)\"）

示例：\n"
自然语言表达式：[\"{Review} 是正面且
有意义的用户评价\"]
SQL 谓词：[\"评价 != '非数字'\"，\"评价 =
'好'\"，\"评分 > 10\"]
输出：[真，假，假]
(评价 != '非数字' 是必要的：'非数字' 违反了
自然语言表达式。评价 = “好”并非必要条件：它只是一个
充分条件，但会错误地
排除其他正面评价。评分 > 10 与
自然语言表达式无关。）
注意：输出必须是仅包含布尔值的有效 JSON 数组
。；
