arXiv:2603.02081v1 [cs.DB] 2 Mar 2026

GenDB：查询处理的下一代——由合成而非工程设计

Jiale Lao 康奈尔大学
jiale@cs.cornell.edu

Immanuel Trummer 康奈尔大学
itrummer@cornell.edu

摘要

传统的查询处理依赖于由众多专家精心优化和设计的引擎。然而，新技术和用户需求不断快速演进，现有系统往往难以跟上步伐。与此同时，这些系统内部复杂，难以扩展；而开发新系统则需要投入大量工程精力和成本。本文认为，大语言模型（LLM）的最新进展正开始塑造新一代查询处理系统。

我们提出，利用大语言模型为每个到来的查询合成执行代码，而不是持续构建、扩展和维护复杂的查询处理引擎。作为概念验证，我们展示了 GenDB：一个由大语言模型驱动的智能体系统，可生成实例优化且经过定制的查询执行代码，以适配特定的数据、工作负载和硬件资源。

我们实现了 GenDB 的早期原型，该原型在多智能体系统中使用 Claude Code Agent 作为底层组件，并在 OLAP 工作负载上对其进行评估。我们采用了广为人知的 TPC-H 基准测试中的查询，并构建了一个新的基准测试，旨在减少大语言模型训练数据可能造成的数据泄漏。我们将 GenDB 与当前最先进的查询引擎进行了比较，包括 DuckDB 、 Umbra 、 MonetDB 、 ClickHouse 和 PostgreSQL。GenDB 的性能显著优于这些系统。最后，我们讨论了 GenDB 当前的局限性，并概述了未来的扩展方向和相关研究挑战。

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

1 引言

数十年来，人们开发了大量数据库管理系统（DBMS）[7, 11, 38, 40, 45, 48, 65]。这些系统设计复杂，经过精心工程化实现，以实现高性能和丰富的功能。然而，新技术和用户需求发展迅速。因此，现有系统往往跟不上变化，必须依赖外部扩展，甚至从头开发全新系统。例如，PostgreSQL 在 OLTP 方面表现出色，但在 OLAP 工作负载、时间序列数据的实时分析，以及需要向量嵌入存储和相似性搜索的工作负载方面效果较差。人们开发了许多扩展，为 PostgreSQL 添加这些功能 [9, 43, 54], ，其中一些扩展试图缓解原有系统的架构限制（例如，面向页的存储布局不适用于向量索引 [35]）。此外，人们还设计了若干针对特定工作负载的专用系统，通常能显著提升性能 [7, 11, 40, 44, 48, 49, 62]。

由于系统复杂，扩展数据库系统困难重重，而且往往会引入不兼容性和系统故障 [20]。另一种做法是开发新系统，但这需要投入大量工程工作和资金成本。近年来，涌现出更多新技术，例如针对多模态数据的语义查询处理，已集成到一些工业系统中 [7, 11], ；以及原生 GPU 分析处理，其成本效率已优于传统的基于 CPU 的处理 [63]。面对这些及未来的技术，我们应该继续扩展现有系统（如 PostgreSQL），还是设计并实现新系统？**或者还有别的选择吗？**

在本文中，我们认为，大语言模型的近期进展正开始塑造下一代查询处理方式：为每个查询生成特定的执行代码，而不是持续构建、扩展和维护复杂的查询引擎。作为概念验证，我们提出了 GenDB，这是一个由大语言模型驱动的智能体系统，能够生成针对特定数据、工作负载和硬件资源量身定制的实例优化执行代码。为每个查询生成代码，为查询优化提供了前所未有的空间，也为集成新技术带来了极强的可扩展性。下面我们给出一个示例，并将在第 5 节中进一步讨论。

示例 *1.1.* 查询重复是真实世界工作负载的一项关键特征：在 50% 的 Amazon Redshift 集群中，80% 的查询会与此前观察到的查询完全重复 [61]。短时间运行和长时间运行的查询都会重复，而长时间运行的查询几乎总会在不同集群中再次出现，因为它们对应于定期的转换或分析任务 [61]。GenDB 非常适合这类工作负载，因为前期的代码生成成本可以分摊到多次执行中。对于即席查询，GenDB 可以与传统数据库管理系统组成混合架构：传统系统处理一次性查询，而 GenDB 加速频繁查询。GenDB 服务于不同的用户。对于开发人员，它会自动生成实例优化执行代码，其正确性可通过人工检查来验证。对于没有 SQL 背景的用户，GenDB 可以轻松扩展自然语言接口，类似于部署在生产系统中的对话服务，而这类服务通常不提供正确性保证 [8, 14, 50]。

GenDB 是一个基于大语言模型构建的智能体系统。它接收数据库模式、自然语言问题或 SQL 查询以及数据作为输入，同时还接收用户配置（例如优化目标和预算）以及可用资源（例如硬件资源或远程大语言模型 API 的访问权限）。GenDB 首先分析工作负载和硬件，以提取结构化特征，为后续设计决策提供指导。随后，它会生成定制的存储布局和索引结构。GenDB 同时支持标准关系型数据和

扩展数据类型，例如图像和音频。基于工作负载分析和存储设计，GenDB 会生成资源感知型执行计划并输出可执行代码。它利用运行时反馈迭代优化性能。达到用户定义的优化目标或预算后，GenDB 会输出经过优化的存储结构、索引，以及每个查询对应的一个可执行文件，所有内容均针对特定数据、工作负载和资源量身定制。

我们在 OLAP 工作负载上评估了 GenDB 的早期原型。我们使用来自 TPC-H 基准测试的查询，并构建了一个新的基准测试，旨在降低大语言模型训练数据可能造成的数据泄漏。我们将 GenDB 与最先进的查询引擎进行比较，包括 DuckDB、Umbra、MonetDB、ClickHouse 和 PostgreSQL。通过根据运行时反馈迭代优化代码，GenDB 最终生成了正确的实现，并取得了显著更优的性能。目前，GenDB 通过将生成的查询结果与传统数据库系统产生的结果进行比较来验证查询正确性。当无法获得真实查询结果时，我们建议由人类专家手动验证生成代码。这一验证过程通常比实现高性能代码或手动调优查询引擎更为容易。对于不具备 SQL 背景、并通过自然语言接口进行交互的用户场景，GenDB 不提供形式化正确性保证，这与 Databricks [8]、BigQuery [14], 和 Snowflake [50]已在生产环境中部署的对话服务相一致。在未来的研究中，我们计划开发更多安全机制，以降低生成错误代码的可能性。与此同时，大语言模型的进步将有助于整体提升代码质量。最后，我们讨论了当前的局限性，并概述了未来的扩展方向及相关研究挑战。

2 背景与相关工作

基于编译的查询代码生成。 生成定制代码以实现高效查询处理的想法并非新鲜事，GenDB 与使用编译技术生成此类代码的先前工作有关 [5, 21]。HIQUE [21] 通过将执行计划中的每个算子替换为人工编写的 C 代码模板，为每个查询生成定制代码。随后，这些生成的代码片段会被编译并动态链接，从而生成一个可执行文件。LegoBase [5] 使用高级编程语言 Scala 实现查询引擎，然后利用编译器为每个查询生成优化后的 C 代码。这些工作的动机是观察到，手写的 C/C++ 代码明显优于速度极快的向量化系统 [39, 65]。这种性能提升源于函数调用更少、硬件利用率更高（从而改善数据和代码的局部性），以及更有效的编译器优化。聘请人类专家为大量长时间运行查询编写定制代码，成本高昂且难以扩展。因此，这些方法被提出，作为自动代码生成的早期尝试。然而，由于依赖人工设计的代码模板和基于编译器的优化，这些方法提供的定制能力有限。生成过程本身是固定的，并不了解数据特征、工作负载模式或硬件属性。

大语言模型增强的代码生成。 GenDB 与使用大语言模型进行代码生成的先前工作相关 [6, 56, 59]。CodexDB [56] 通过将执行计划中的每个算子转换为基于算子特定文本模板的自然语言描述，然后调用 GPT-3 Codex 模型生成 Python 代码，从而生成用于查询处理的 Python 代码。GenesisDB [59] 在 CodexDB 的基础上，生成可复用的通用关系算子实现。这些系统提供了自定义选项，例如选择数据处理库（如 Python 中的 Pandas 或 Vaex），以及启用调试功能（如在执行期间打印中间结果）。不过，这些系统并不以系统化的性能优化为目标，主要面向特定的评估场景。ADRS [6] 将现有系统中的选定算法替换为 AI生成算法。它通过迭代优化提示、使用大语言模型生成或改进解决方案，并在系统中对其进行评估。结果表明，AI生成算法可以胜过人工设计的先进方法。ADRS 与 GenDB 的研究方向正交。ADRS 旨在自动为现有系统发现有效解决方案，而 GenDB 则以合成代码取代传统引擎。**大语言模型增强的数据库系统。** 更广泛地说，GenDB 与利用大语言模型增强数据库系统的方法相关，包括数据库管理系统调优 [13, 15, 24, 25, 57], 、查询重写 [34, 51], 、学习型查询优化 [33, 52, 53], 、工作负载生成 [22, 23, 64], 、语义查询处理 [17, 26, 32, 42],和自然语言接口 [12, 28, 31]。

3 系统概览

图1展示了 GenDB 的概览。它以目标工作负载（包括数据库模式、SQL查询或自然语言问题，以及数据）、用户配置（例如优化目标和预算）以及可用资源（例如 GPU 等硬件资源，或通过 API 访问远程大语言模型以处理多模态数据）作为输入。基于这些输入，GenDB 会生成高度优化且定制化的数据库存储结构（例如采用编码和压缩的列式存储）、索引，以及每个查询对应的一个可执行文件。所有生成的组件都针对给定的数据、工作负载和硬件资源进行了定制。

GenDB 是一个基于大语言模型的智能体系统，它将复杂的端到端查询处理与优化任务拆解为一系列更小且定义明确的步骤，并由专门的大语言模型智能体分别处理。首先，WorkloadAnalyzer 分析硬件资源、数据库模式、SQL 查询和底层数据，以生成结构化的工作负载特征（例如存储类型和 SIMD 支持、表和列统计信息、连接模式以及过滤选择率）。下游智能体利用这些特征来指导优化决策。其次，存储/索引设计器设计并实现代码，将原始数据格式转换为经过优化的存储结构（例如，为 OLAP 工作负载采用带有编码和压缩的列式存储），并构建相应的索引。未来，我们计划扩展 GenDB，采用扩展的关系数据模型，使单元格除标准 SQL 数据类型外还可以存储图像或音频文件。语义查询处理引擎已采用类似模型 [7, 11, 17, 32, 42]。结合由大语言模型驱动的语义算子，这种扩展的数据模型使系统能够在统一框架中处理结构化和非结构化数据。第三，Query Planner 会生成一个

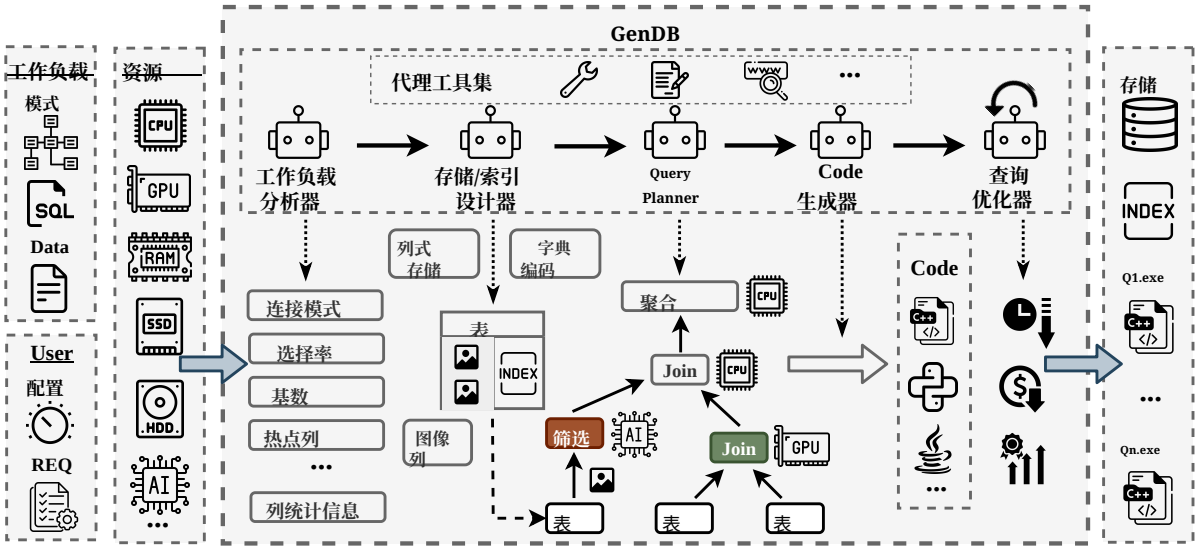

图1：GenDB的系统概览

高效的执行计划，依据是工作负载分析以及存储和索引设计。每种物理算子的实现都能感知资源。例如，聚合算子会根据硬件缓存层次结构调整策略：对于针对低基数列（如 nation，含 25 个不同键）的分组操作，它使用可完全放入一级缓存、且哈希开销为零的直接扁平数组；而对于包含数百万个不同分组的情况，它会改用大小适合各核心私有缓存的线程局部哈希表，并且仅在扫描完成后才并行合并结果。第四，代码生成器使用适当的编程语言实现执行计划（例如，使用 C++ 实现高性能代码，使用 Python 实现人工智能操作），并应用系统级优化（例如编译器优化和内存映射等操作系统函数）。生成的代码会在数据库上执行，以衡量其正确性和效率。GenDB 通过将生成的查询结果与传统数据库系统生成的结果进行比较，来验证查询的正确性。如果没有真实结果可用，我们建议由人类专家验证生成的代码；与手动实现高性能代码或调优查询引擎相比，这通常更容易。对于没有 SQL 背景、通过自然语言界面交互的用户，GenDB 不提供形式化正确性保证。这一设计选择与现有投入生产的对话式分析服务一致 [8, 14, 50]。第五，查询优化器利用反馈迭代改进执行计划和代码实现，以提升性能。优化目标由用户定义，且具有灵活性。目前，我们支持针对单一指标进行优化，例如热运行时间。未来，我们计划尝试约束优化，例如在满足查询执行时间服务等级协议（SLA）的同时，最大限度地降低内存和磁盘使用量；还计划尝试多目标优化，同时权衡性能和成本。最后，在用尽用户配置的优化预算（例如，最大优化迭代次数）或达到用户指定的优化目标，GenDB输出经实例优化的数据存储、索引，以及每个查询对应的一个可执行文件，所有这些均针对特定数据、工作负载和可用资源进行定制。

GenDB中的每个智能体都以大语言模型作为其核心推理组件，用于规划并自动调用工具以完成分配的任务。每个智能体都配备了内置工具，包括读取、写入和编辑文件等文件操作；用于执行命令和脚本的终端访问；以及网络搜索和网页内容检索等网络相关工具。智能体能力可通过集成智能体技能 [29] 或实现模型上下文协议（MCP） [10]进一步扩展。这种工具使用能力使智能体能够完成复杂任务。例如，如果某个查询包含多表连接且性能非常差，智能体可以生成采样程序来评估候选连接顺序、测量其性能，然后优化代码实现以使用实测最佳连接顺序。

4 实验

4.1 设置

对比系统。 我们将 GenDB 与最先进的查询引擎进行比较，使用撰写时的最新版本：DuckDB v1.4.4 [45], ClickHouse v26.2.1 [48], Umbra（2026年2月发布版） [40],MonetDB v11.55.1 [16], 以及 PostgreSQL v18.2 [38]。我们确保所有系统使用相当的硬件资源。GenDB 不需要手动调优系统参数或索引，因为它会在代码生成过程中确定合适的设置。为确保公平，我们报告两个版本：一个是每个系统仅使用其自动创建的索引（例如主键索引，或 OLAP 引擎默认创建的区域映射），另一个是在这些系统创建的索引基础上补充由 GenDB 生成的额外索引。

硬件。 所有实验均在一台 Ubuntu 服务器上进行，该服务器配备两颗英特尔至强金牌 5218 处理器（2.3 GHz，32 个物理核心）。

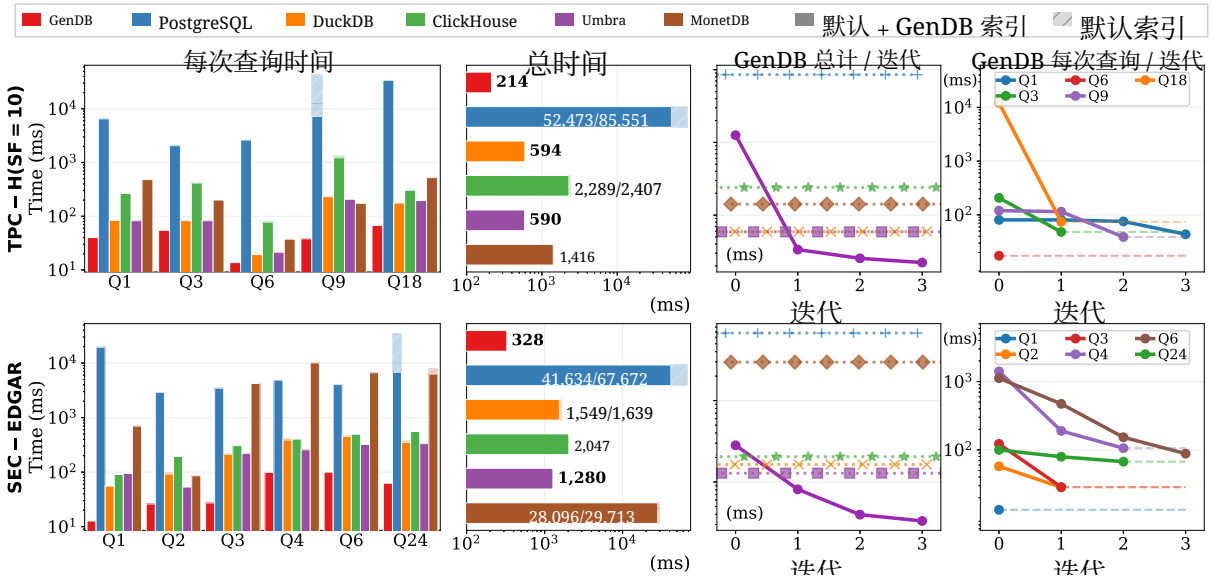

图 2：TPC-H（SF = 10）和 SEC-EDGAR 基准测试的实验结果

系统配备 384 GB 随机存取存储器，可确保整个数据库都缓存在内存中，配置与 [27] 相同。我们报告各系统热运行的执行时间。**实现。** GenDB 使用 JavaScript 实现，并采用 Claude Agent 构建智能体系统。Claude Sonnet 4.6 作为底层大语言模型，在成本与性能之间取得平衡。生成代码使用 C++ 实现。

基准测试。我们使用（1）规模因子为 10 的 TPC-H [55]，以及（2）一个全新、由人工构建的基准测试 SEC-EDGAR。我们报告各个查询的详细结果，而不仅仅是聚合结果。遵循 [19],，我们选取一个具有代表性的子集（Q1、Q3、Q6、Q9、Q18），以涵盖 TPC-H 中的主要性能挑战。先前的分析表明，在这些查询上表现出色的引擎，很可能在整个工作负载上也表现良好 [4]。我们引入 SEC-EDGAR，以减少潜在的数据泄漏。TPC-H 已得到充分研究，其查询和优化策略在大语言模型训练数据中都有充分体现。相比之下，SEC-EDGAR 基于一个很少用于数据库基准测试的真实世界财务报表数据集。我们使用 2022年 至 2024年 的数据（5 GB）[60]。我们使用 SQLSmith [1] 生成 1,000 个随机查询，然后应用基于多样性的采样，选出六个查询。

4.2 实验结果

图 2 展示了实验结果；在两项基准测试的每个查询中，GenDB 的表现都优于所有基线系统。在 TPC-H 上，GenDB 执行五个代表性查询的总耗时为 **214 毫秒**。这一结果比两款速度最快的基线系统 DuckDB（594 毫秒）和 Umbra（590 毫秒）快 **2.8**× 倍，比 ClickHouse 快 **11.2**× 倍。在 SEC-EDGAR 上，GenDB 的耗时为 **328 毫秒**，比 DuckDB 快**5.0**× 倍，比 Umbra 快**3.9**× 倍。查询复杂度越高，性能差距越大。例如，在 TPC-H Q9 中，这是一项包含 LIKE 筛选条件的五表连接，GenDB 仅用 38 毫秒便完成查询，比 DuckDB 快**6.1**× 倍。

GenDB 采用带有提前停止准则的迭代优化。在 TPC-H 上，Q6 在第 0 次迭代时通过区域映射剪枝和无分支扫描达到接近最优的 17 毫秒耗时，无需进一步优化。相比之下，Q18 的耗时从 12,147 毫秒降至第 1 次迭代时的 74 毫秒，提升了 **163**×。这一提升源于将导致缓存抖动的哈希聚合替换为索引感知顺序扫描。在 SEC-EDGAR 上，Q4 的耗时经过三次迭代从 1,410 毫秒降至 106 毫秒，提升了**13.3**× 倍；Q6 的耗时经过四次迭代从 1,121 毫秒降至 88 毫秒，提升了**12.7**× 倍。在 Q6 中，优化器逐步将扫描、压缩和合并操作融合到单个 OpenMP 并行区域中，从而消除了三次线程创建开销。到第 1 次迭代时，GenDB 已经优于所有基线系统。

GenDB 通过针对工作负载和硬件特性的优化实现高性能，而这些优化在通用引擎中难以实现。我们将这些优化分为四类。第一，数据感知列编码：GenDB 分析值分布，以选择紧凑的物理布局来减少内存流量。第二，算法级重构：GenDB 不从固定的一组连接与聚合算子中进行选择，而是生成适配数据布局和可用索引的算法。第三，缓存自适应聚合：GenDB 根据分组基数和缓存容量选择聚合策略，例如对低基数分组使用直接数组，否则使用哈希表。第四，工作负载特定的派生数据结构：GenDB 构建查询特定结构，例如半字节打包查找文件和批量预取探测模式。这些优化需要综合考虑存储布局、查询语义、数据分布和硬件特性，而通用引擎中固定的算子抽象无法做到这一点。

由于篇幅所限，我们介绍一个具有代表性的示例。通用引擎对所有查询使用相同的哈希聚合算子。相比之下，GenDB 会根据查询生成的分组数量以及聚合状态所在的缓存级别，生成不同的聚合

代码，如下面的代码片段所示。测试机器具有 64 个核心，每个核心配备 32 KB 一级缓存和 512 KB 二级缓存，并共享 44 MB 三级缓存。TPC-H Q1 按退货标志 × 行状态分组，仅生成 $3 \times 2 = 6$ 个分组。每个累加器包含 6 个字段（cnt、sum_qty、sum_price 等），并填充至 64 字节缓存行以防止伪共享，因此 6 个累加器在线程中仅占用 $6 \times 64 = 384$ 字节，完全可以放入 32 KB 一级缓存。因而，GenDB 跳过哈希操作，使用直接数组。分组索引通过对两列值执行简单的乘加运算计算得到。全部 64 个线程完成后，合并操作就是对 $6 \times 64 = 384$ 个值进行顺序求和。TPC-H Q3 按订单键分组，最多包含 4M 个不同分组。如果 64 个线程中的每个线程都维护自己的 4M 条目哈希表，则总内存将达到 $64 \times 4\text{M} \times 12\text{B} \approx 3\text{GB}$（每个条目存储一个 4 字节键和一个 8 字节收入值），远超 44 MB 三级缓存。因此，GenDB 使用单个共享哈希表，并采用列分离布局：键（int32_t，$4\text{M} \times 4\text{B} = 16\text{MB}$）与值（double，$4\text{M} \times 8\text{B} = 32\text{MB}$）分开存储，使得探测只需访问键数组，而键数组可以放入 44 MB 三级缓存。多个线程通过无锁的比较并交换（CAS）更新共享值数组，从而同时避免锁和单独的合并阶段。在这两个极端之间，GenDB 还会根据分组数量，将每线程哈希表调整至适合放入二级缓存或三级缓存的大小，并相应选择合适的合并策略。这种适应性要求在代码生成时了解分组基数和缓存拓扑，而通用引擎无法做到这一点，因为它们必须使用单一的算子实现来处理任意基数。

TPC-H Q1（6 个分组）：一级缓存中的直接数组，无需哈希

```
// 6 个分组 x 64 B = 384 B/线程，可容纳于 32 KB 一级缓存 结构体 alignas(64) Accum {int64_
t cnt, sum_qty, sum_price, ...;};
std::array<Accum, 6> 局部变量; // 每线程独有，无需哈希
int g = 退货标志[i] * 2 + 行状态[i]; // 直接索引
local[g].cnt++;
local[g].sum_price += price[i];
```

TPC-H Q3（400 万个分组）：共享列式布局哈希表，键位于三级缓存中

```
// 列分离：键和值存储在单独的数组中
int32_t oa_键[4194304]; // 400 万 x 4B = 16MB，可容纳于 44 MB 三级缓存中
double oa_rev[4194304]; // 400 万 x 8B = 32 MB

// 探测仅访问 oa_keys（命中三级缓存），不访问 oa_rev
uint64_t 槽 = oa_探测(oa_键, orderkey);
// 64 个线程共享一张表；使用无锁比较并交换，无需合并
atomic_add_double(&oa_rev[slot], revenue);
```

图3展示了对 GenDB 的消融研究。我们从完整的多智能体系统出发，分两步简化它。首先，将多智能体流水线替换为单个智能体，但仍向其提供有关数据库优化策略的领域专属提示（有指导 变体）。其次，完全移除这些提示，让单个智能体自主决定自己的方法（高层级 变体）。在 TPC-H 上，多智能体设计耗时 236 ms，比最佳单智能体变体（有指导，281 ms）**1.2**× 快。在未见过的 SEC-EDGAR 基准测试上，差距进一步扩大：比有指导变体（752 ms）快**2.3**×，比高层级变体（1,325 ms）快**4.1**×。多智能体设计的成本也更低：TPC-H 上为 14.15 美元，而单智能体为 17.54 美元；SEC-EDGAR 上为 23.49 美元，而单智能体为 27.46 美元。两步消融都产生了明显影响。在 TPC-H 上，单智能体变体仍有竞争力，因为其查询和优化策略在大语言模型训练数据中有充分体现；但在未见过的 SEC-EDGAR 基准测试上，它们的表现则有所下降。移除领域专属提示（有指导 → 高层级）会使 SEC-EDGAR 上的速度**1.8**× 下降，这表明即使是粗略的任务指导，也有助于大语言模型在超越记忆模式的情况下实现泛化。恢复多智能体协作（有指导 → 多智能体）则能进一步提升性能：该流水线被拆分为定义明确的步骤——分析、存储设计、规划、代码生成和优化——每个步骤都由专用智能体处理。多智能体设计还使用确定性代码来强制执行约束（例如迭代预算、停止标准），并使用结构化 JSON 进行智能体间通信；相比之下，单个智能体可能会创建忽略用户指定约束的子智能体，并依赖结构较松散的自由格式文本交流。

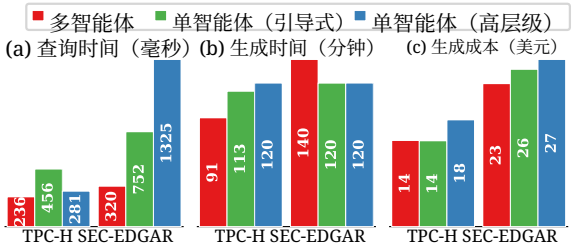

图3：GenDB 消融研究

5 研究议程

5.1 降低代码生成成本

根据任务复杂度自适应选择模型。 使用较小的模型可以缩短生成时间并降低金钱成本，但也可能降低输出质量。一个有前景的方向是分两个层级分配不同能力的模型。首先，在流水线步骤之间进行分配：我们的消融研究表明，结构化分解既能提升质量，也能降低成本，这说明分析等较简单的步骤可以使用较小的模型，而代码生成等复杂步骤则需要更大的模型。其次，在不同查询之间进行分配：TPC-H Q6 在迭代 0 时便收敛，而 Q18 需要经过多次迭代才能取得 **163**× 提升，这表明我们可以针对复杂度不同的查询采用不同的模型和投入。**生成可复用组件。** GenDB 不必为每个查询单独生成代码，而是可以通过生成供多个查询共享的可复用算子，或为查询模板生成代码，来降低总体生成成本。目前，GenDB 只复用基本的实用函数（例如日期转换）。一个有前景的方向是生成可复用的数据结构、关系算子和查询模板。我们的实验表明这切实可行：缓存自适应聚合适用于多个 TPC-H 查询，它们仅在分组基数上有所不同，而这类模式可以封装为参数化组件。除了共享之外，多查询优化 [18] 还可以使 GenDB 生成一个可执行文件，在一次执行中返回多个查询的结果。

降低令牌消耗。 通过设计更简洁的提示模板、将大语言模型配置为较低的推理力度，以及对智能体间通信（例如 TOON [37]）和数据库

模式 [58], 与查询 [13]应用压缩方法，可以降低令牌消耗。然而，这些方法可能会降低生成质量。如何平衡令牌消耗与生成质量，仍然是一项尚待解决的挑战。

5.2 提高代码生成的稳健性

将领域知识融入代码生成。目前，GenDB 不使用外部知识库，因为大部分领域知识已经包含在大语言模型训练数据中。然而，即使训练数据中存在相关知识，大语言模型也可能无法从其参数中正确或一致地提取这些知识。更重要的是，非公开知识——例如未发表的研究、专有优化技术和公司内部实践——完全不包含在训练数据中，但可能显著提升生成质量。原则上，可以通过网络搜索获取近期研究中的公开知识，但网络是一个庞大而嘈杂的信息来源，因此有效且高效地检索信息本身就是一项挑战。一个有前景的方向是整合并管理外部知识库，以补充大语言模型的参数化知识，同时避免 GenDB 被嘈杂或未经验证的信息误导。

处理智能体和大语言模型的静默故障。我们发现，代码生成过程中会出现两类静默故障。首先，大语言模型生成的代码可能效率低下，甚至无法终止，尤其是在模式和查询较复杂时。一个常见原因是，在使用开放寻址法进行连接和聚合时，哈希表构建不正确：大语言模型可能会遗漏扩容逻辑或探测边界、低估基数，或使负载因子接近 100%，导致探测循环无法终止。我们在具有高基数连接的查询中尤其观察到了这一问题，例如 TPC-H Q18；在此类查询中，优化器必须仔细推理如何为 64 个线程确定表的大小。其次，任务复杂时，推理和输出令牌总数可能超过大语言模型的最大输出限制，导致智能体挂起且不返回响应 [2]。这两类故障都会导致生成过程无限期停滞，并浪费服务器资源。我们当前的解决方案是设置查询级超时（300 秒）和智能体级超时（30 分钟）；若触发任一超时，便自动回滚到此前迭代中的最佳实现。未来仍需探索更系统的解决方案，例如自动检测潜在代码问题、优雅终止并明确报告错误。**改进并扩展智能体间通信**。智能体间的协同失调是导致故障的主要原因之一 [41], , 在查询处理的代码生成中尤为突出，因为消息复杂度会随着模式规模和查询复杂度的增加而上升。我们的消融研究证实，结构化通信至关重要：多智能体设计使用结构化 JSON 传递智能体间消息，而单智能体设计使用结构化程度较低的自由格式文本；后者仍可能生成子智能体，因此同样依赖智能体间通信，这也导致其在未见过的工作负载上表现较弱。我们发现的一个问题是，存储/索引设计器会将 DECIMAL 列表示为带有 scale_因子的 int64_t，并将其记录在逐查询文档中，供下游智能体使用。然而，后续智能体并不总是遵循这一规范——例如，两个 DECIMAL 列相乘后，可能会省略必要的除法操作因子，从而产生错误结果。目前，GenDB 使用 JSON 和预定义的 Markdown 模板。评估 TOON [37], 、MCP [3], 和 A2A [10] 等其他协议，是一个很有前景的未来研究方向。

评估并验证生成代码。GenDB 依靠反馈迭代地纠正和优化查询实现。目前，我们通过将 GenDB 的输出与传统数据库管理系统（例如 DuckDB）的真实结果进行比较来验证正确性；同时，借助基于通知的执行队列串行执行查询，并在不同查询之间并行开展规划和代码生成，以确保测量可靠。未来，我们计划探索三个方向。第一，在保持测量准确性的同时提高执行吞吐量，因为当前的串行队列已成为瓶颈。第二，为模板化查询模式提供更强的正确性保证，例如形式化验证。第三，提供更细粒度的反馈，例如逐算子性能分析或硬件计数器分析，帮助优化器更准确地识别瓶颈，并通过更少的迭代收敛。

开发一个随时间不断改进的自演进系统。生成实例优化执行代码是一个迭代过程，往往需要反复试验。我们不应反复运行 GenDB，再依赖人工总结观察结果来改进系统，而应设计一个能够从过往试验中学习并随时间不断改进的自演进系统。一种可行的扩展方案是维护一个名为“经验”的持久化模块。每次生成任务结束后，智能体都可以分析运行情况，并总结：（1）能够提升性能的技巧；（2）导致查询结果错误或执行效率低下的问题。随后，系统可以将经过验证的规则整合到指导大语言模型的提示中，或者根据积累的经验重新设计系统。

5.3 扩展代码生成的灵活性

为语义查询处理生成定制代码。数据库系统正不断扩展，以支持文本、图像、音频和视频等多种数据类型 [7, 11, 17, 26, 32, 36, 42], , 并且通常会使用机器学习模型或大语言模型处理多模态数据（例如，AI.IF(picture, “the picture shows a red car”)）。在这类工作负载中，模型推理占据了执行成本的主要部分，因此优化重点从关系算子转向降低推理成本并提升结果质量。GenDB 可以自然地扩展到这一场景：它能够生成 Python 代码，直接部署机器学习模型或调用远程大语言模型，并探索提示调优和模型选择等技术 [26]，以优化成本和质量。**为现代硬件生成专用代码**。GPU 驱动的分析处理展现出强劲的性能，在许多情况下，其成本效率超过传统的基于 CPU 的处理 [30, 63]。此外，GenDB 可以利用多个高性能 GPU 库生成 GPU 原生查询处理代码。例如，libcudf [46] 提供了高效的 GPU 关系算子实现，而 RMM [47] 支持高性能 GPU 内存管理。

6 结论

GenDB 将查询处理从人工设计的系统转变为由大语言模型驱动的逐查询代码生成。我们的原型展现出令人鼓舞的结果，同时也表明仍有改进空间。

参考文献

[1] 2020. A. Seltenreich. Sqlsmith. https://github.com/anse1/sqlsmith [2] Anthropic. 2024年。Claude Code issue #24055. https://github.com/anthropics/claude-code/issues/24055. 访问日期：2026年2月21日。[3] Anthropic. 2024年。模型上下文协议：简介。https:// modelcontextprotocol.io/。访问日期：2026年2月21日。[4] Peter Boncz、Thomas Neumann 和 Orri Erling。2013年。TPC-H 分析：来自一项重要基准测试的隐含信息与经验教训。载于《技术性能评估与基准测试会议》。Springer，第 61–76 页。[5] Hassan Chafi。2014年。使用高级语言构建高效查询引擎。VLDB 汇刊 7, 10 (2014), 853–864。[6] Audrey Cheng、Shu Liu、Melissa Pan、Zhifei Li、Shubham Agarwal、Mert Cemri、Bowen Wang、Alexander Krentsel、Tian Xia、Jongseok Park、Shuo Yang、Jeff Chen、Lakshya Agrawal、Ashwin Naren、Shulu Li、Ruiying Ma、Aditya Desai、Jiarong Xing、Koushik Sen、Matei Zaharia 和 Ion Stoica。2025年。让“野蛮人”入场：人工智能如何加速系统性能研究。arXiv:2512.14806 [cs.SE] https://arxiv.org/abs/2512.14806[7] Benoit Dageville、Thierry Cruanes、Marcin Zukowski、Vadim Antonov、Artin Avanes、Jon Bock、Jonathan Claybaugh、Daniel Engovatov、Martin Hentschel、Jiansheng Huang 等。2016年。Snowflake 弹性数据仓库。载于《2016 年国际数据管理会议论文集》，第 215–226 页。[8] Databricks。 [无日期]。商业智能。https://www.databricks.com/product/ business-intelligence。访问日期：2026年2月28日。[9] DuckDB。2025年。pg_duckdb。https://github.com/duckdb/pg_duckdb。访问日期：2026年2月27日。[10] Abul Ehtesham、Aditi Singh、Gaurav Kumar Gupta 和 Saket Kumar。2025年。智能体互操作协议综述：模型上下文协议（MCP）、智能体通信协议（ACP）、智能体对智能体协议（A2A）和智能体网络协议（ANP）。arXiv:2505.02279 [cs.AI] https://arxiv.org/abs/2505. 02279[11] Sérgio Fernandes 和 Jorge Bernardino。2015年。什么是 BigQuery？载于《第19届国际数据库工程与应用研讨会论文集》，第 202–203 页。[12] Dawei Gao、Haibin Wang、Yaliang Li、Xiuyu Sun、Yichen Qian、Bolin Ding 和 Jingren Zhou。2024年。大语言模型赋能的文本到SQL：基准评估。VLDB 汇刊 17, 5 (2024年1月), 1132–1145。https: //doi.org/ 10 14778 3641204 3641221 . / . Victor Giannakouris 和 Immanuel Trummer[13]。2025年。lambda-Tune：利用大语言模型实现数据库系统自动调优。ACM 数据管理会议论文集 3, 1, 2 2025 26 10 1145 3709652 文章（2025年2月），页数。[14] 2026 https://doi.org/ . / Google Cloud。 . 在 BigQuery 中引入对话式分析。https://cloud.google.com/blog/products/data-analytics/introducing-conversational-analytics-in-bigquery?e=48754805。发布日期：2026年1月29日。访问日期：2026年2月28日。[15] Xinmei Huang、Haoyang Li、Jing Zhang、Xinxin Zhao、Zhiming Yao、Yiyan Li、Tieying Zhang、Jianjun Chen、Hong Chen 和 Cuiping Li。2026年。E2ETune：通过微调生成式语言模型实现端到端旋钮调优。VLDB 汇刊 18, 13 (2026年1月), 5540–5554。https://doi.org/10.14778/3773731.3773732[16] Stratos Idreos、Fabian Groffen、Niels Nes、Stefan Manegold、K Sjoerd Mullender、Martin L Kersten 等。2012年。MonetDB：列式数据库架构的二十年研究。IEEE Data Eng. Bull. 35, 1 (2012), 40–45。[17] Saehan Jo 和 Immanuel Trummer。2024年。Thalamusdb：多模态数据上的近似查询处理。ACM 数据管理论文集 2, 3 (2024), 1–26。[18] Tarun Kathuria 和 S. Sudarshan。2017年。高效且可证明的多查询优化。载于《第36届 ACM SIGMOD-SIGACT-SIGAI 数据库系统原理研讨会论文集》（美国伊利诺伊州芝加哥）（PODS ’17）。计算机协会，美国纽约州纽约市，第 53–67 页。https://doi.org/10.1145/ 3034786.3034792[19] Timo Kersten、Viktor Leis、Alfons Kemper、Thomas Neumann、Andrew Pavlo 和 Peter Boncz。2018年。关于编译型和向量化查询，你一直想知道却又不敢问的一切。VLDB 汇刊 13 (2018), 2209–2222。[20] Abigale Kim、Marco Slot、David G Andersen 和 Andrew Pavlo。2025年。数据库中的无政府状态：数据库管理系统可扩展性的综述与评估。VLDB 汇刊 18, 6 (2025), 1962–1976。[21] Konstantinos Krikellas、Stratis D Viglas 和 Marcelo Cintra。2010年。为整体式查询评估生成代码。载于《2010年 IEEE 第26届国际数据工程会议（ICDE 2010）》，IEEE，第 613–624 页。[22] Jiale Lao 和 Immanuel Trummer。2025年。演示 SQLBarber：利用大语言模型生成定制且逼真的 SQL 工作负载。载于《2025年国际数据管理会议伴随会议论文集》（德国柏林）（SIGMOD/PODS ’25）。计算机协会，美国纽约州纽约市，151 154 10 1145 3722212 3725101 – 。[23] 2026 https://doi.org/ . / . Jiale Lao 和 Immanuel Trummer。 . SQLBarber：一种利用大语言模型生成定制且逼真的 SQL 工作负载的系统。ACM 数据管理会议论文集 4, 1，文章 297（2026年2月），27 页。https://doi.org/10.1145/ 3786699 [24] Jiale Lao、Yibo Wang、Yufei Li、Jianping Wang、Yunjia Zhang、Zhiyuan Cheng、Wanghu Chen、Mingjie Tang 和 Jianguo Wang。2024年。GPTuner：一种通过 GPT 引导的贝叶斯优化实现的手册阅读型数据库调优系统。VLDB 汇刊 17，8（2024年5月），1939–1952。https://doi.org/10.14778/3659437. 3659449[25] Jiale Lao、Yibo Wang、Yufei Li、Jianping Wang、Yunjia Zhang、Zhiyuan Cheng、Wanghu Chen、Mingjie Tang 和 Jianguo Wang。2025年。GPTuner：一种基于大语言模型的数据库调优系统。SIGMOD Rec. 54，1（2025年4月），101–110。https: // doi.org/10.1145/3733620.3733641[26] Jiale Lao、Andreas Zimmerer、Olga Ovcharenko、Tianji Cong、Matthew Russo、Gerardo Vitagliano、Michael Cochez、Fatma Özcan、Gautam Gupta、Thibaud Hottelier、H. V. Jagadish、Kris Kissel、Sebastian Schelter、Andreas Kipf 和 Im- manuel Trummer。2025年。SemBench：语义查询处理引擎基准测试。arXiv:2511.01716 [cs.DB] https://arxiv.org/abs/2511.01716[27] Viktor Leis、Andrey Gubichev、Atanas Mirchev、Peter Boncz、Alfons Kemper 和 Thomas Neumann。2015年。查询优化器究竟有多好？VLDB 汇刊 9，3（2015年），204–215。[28] Haoyang Li、Jing Zhang、Hanbing Liu、Ju Fan、Xiaokang Zhang、Jun Zhu、Renjie Wei、Hongyan Pan、Cuiping Li 和 Hong Chen。2024年。CodeS：致力于构建用于文本到SQL的开源语言模型。ACM 数据管理会议论文集 2，3，文章 127（2024年5月），28页。https://doi.org/10.1145/3654930[29] Xiangyi Li、Wenbo Chen、Yimin Liu、Shenghan Zheng、Xiaokun Chen、Yifeng He、Yubo Li、Bingran You、Haotian Shen、Jiankai Sun、Shuyi Wang、Qunhong Zeng、Di Wang、Xuandong Zhao、Yuanli Wang、Roey Ben Chaim、Zonglin Di、Yipeng Gao、Junwei He、Yizhuo He、Liqiang Jing、Luyang Kong、Xin Lan、Jiachen Li、Songlin Li、Yijiang Li、Yueqian Lin、Xinyi Liu、Xuanqing Liu、Haoran Lyu、Ze Ma、Bowei Wang、Runhui Wang、Tianyu Wang、Wengao Ye、Yue Zhang、Hanwen Xing、Yiqi Xue、Steven Dillmann 和 Han chung Lee。2026年。SkillsBench：评估智能体技能在多样化任务中的表现。arXiv:2602.12670 [cs.AI] https://arxiv.org/abs/ 2602 12670 。Yinan Li、Bailu Ding、Ziyun Wei、Lukas M Maas、Momin Al-Ghosien、Spyros Blanas、Nicolas Bruno、Carlo Curino、Matteo Interlandi、Craig Peeper 等。2025年。突破GPU内存容量限制，扩展GPU加速数据库。VLDB 汇刊 18，11（2025年），4518–4531。[31] Meiyu Lin、Haichuan Zhang、Jiale Lao、Renyuan Li、Yuanchun Zhou、Carl Yang、Yang Cao 和 Mingjie Tang。2025年。基于文本到SQL的大语言模型安全吗？通过后门攻击探索SQL注入。ACM 数据管理会议论文集 3，6，文章 297（2025年12月），27页。https://doi.org/10.1145/3769762[32] Chunwei Liu、Matthew Russo、Michael Cafarella、Lei Cao、Peter Baile Chen、Zui Chen、Michael Franklin、Tim Kraska、Samuel Madden、Rana Shahout 等。2025年。Palimpzest：通过声明式查询处理优化AI驱动的分析。载于创新数据库研究会议论文集（CIDR）。2。[33] Hanwen Liu、Qihan Zhang、Ryan Marcus 和 Ibrahim Sabek。2025年。SEFRQO：一种自我演进的基于微调RAG的查询优化器。ACM 数据管理会议论文集 3, 6, 361 2025 27 10 1145 3769826 文章（2025年12月），页数。[34] 2025 https://doi.org/ . / Jie Liu 和 Barzan Mozafari。 . GenRewrite：通过大语言模型进行查询重写。arXiv: 2403 09060 [ ] 2403 09060 . cs.DB https://arxiv.org/abs/ . Jiayi Liu、Yunan Zhang、Chenzhe Jin、Aditya Gupta、Shige Liu 和 Jianguo Wang。2026年。PostgreSQL中的快速向量搜索：一种解耦式方法。载于创新数据系统研究会议（CIDR）。[36] Samuel Madden、Michael Cafarella、Michael Franklin 和 Tim Kraska。2024年。数据库无界：借助人工智能查询全世界的字节。VLDB 汇刊 17，12（2024年），4546–4554。[37] Elio Masciari、Vincenzo Moscato、Enea Vincenzo Napolitano、Gian Marco Or- lando、Marco Perillo 和 Diego Russo。2026年。大语言模型准备好使用TOON了吗？对新型结构化输出格式中的结构正确性与可持续性权衡进行基准测试。arXiv预印本 arXiv:2601.12014（2026年）。[38] Bruce Momjian。2001年。PostgreSQL：介绍与概念。第192卷。Addison-Wesley，纽约。[39] Thomas Neumann。2011年。面向现代硬件，高效编译高效查询计划。VLDB 汇刊 4，9（2011年），539–550。[40] Thomas Neumann 和 Michael J Freitag。2020年。Umbra：一种具有内存性能的基于磁盘的系统。载于 CIDR，第20卷。29。[41] Melissa Z Pan、Mert Cemri、Lakshya A Agrawal、Shuyi Yang、Bhavya Chopra、Rishabh Tiwari、Kurt Keutzer、Aditya Parameswaran、Kannan Ramchandran、Dan Klein 等。2025年。多智能体系统为何会失败？载于 ICLR 2025 构建可信语言模型与应用研讨会。[42] Liana Patel、Siddharth Jha、Melissa Pan、Harshit Gupta、Parth Asawa、Carlos Guestrin 和 Matei Zaharia。2025年。语义算子及其优化：在 Lotus 中实现具有准确性保证的基于大语言模型的数据处理。VLDB 汇刊 18，11（2025年），4171–4184。[43] pgvector。2025年。pgvector。https://github.com/pgvector/pgvector。访问日期：2026年2月27日。[44] Pinecone Systems Inc. 2026年。Pinecone。https://www.pinecone.io/。访问日期：2026年2月27日。[45] Mark Raasveldt 和 Hannes Mühleisen。2019年。Duckdb：一种可嵌入的分析型数据库。载于2019年国际数据管理会议论文集。1981–1984。

[46] RAPIDS 开发团队。2024年。RAPIDS cuDF：GPU 数据帧库。https://github.com/rapidsai/cudf。访问日期：2026年2月28日。[47] RAPIDS 开发团队。2024年。RAPIDS RMM：RAPIDS 内存管理器。https://github.com/rapidsai/rmm。访问日期：2026年2月28日。[48] Robert Schulze、Tom Schreiber、Ilya Yatsishin、Ryadh Dahimene 和 Alexey Milovidov。2024年。ClickHouse：人人可用的闪电般快速分析。*VLDB* 汇刊 17，12（2024年），3731–3744。[49] Chunhui Shen、Qianyu Ouyang、Feibo Li、Zhipeng Liu、Longcheng Zhu、Yujie Zou、Qing Su、Tianhuan Yu、Yi Yi、Jianhong Hu 等。2023年。Lindorm 时序数据库：面向大规模监控系统的云原生时间序列数据库。*VLDB* 汇刊 16，12（2023年），3715–3727。[50] Snowflake。 [无日期。]。数据智能体。https://www.snowflake.com/en/product/use-cases/data-agents/。访问日期：2026年2月28日。[51] Yuyang Song、Hanxu Yan、Jiale Lao、Yibo Wang、Yufei Li、Yuanchun Zhou、Jianguo Wang 和 Mingjie Tang。2026年。QUITE：一种超越规则的、基于大语言模型智能体的查询重写系统。arXiv: 2506 07675 [ ] 2506 07675 . cs.DB [52] 2025 https://arxiv.org/abs/ . Pavel Sulimov、Claude Lehmann 和 Kurt Stockinger。 . GenJoin：一种能够从子计划提示中学习的条件生成式计划到计划查询优化器。*ACM* 数据管理会议论文集 3，4，文章 247（2025年9月），25页。https://doi.org/10. 1145/3749165[53] Jie Tan、Kangfei Zhao、Rui Li、Jeffrey Xu Yu、Chengzhi Piao、Hong Cheng、Helen Meng、Deli Zhao 和 Yu Rong。2025年。大语言模型能否成为关系型数据库的查询优化器？*ACM* 数据管理会议论文集 3，6，文章 306（Dec. 2025 28 10 1145 3769771 ），页数。 [54] 2025 https://doi.org/ . / Timescale。 . TimescaleDB。https://github.com/timescale/timescaledb。访问日期：2026年2月27日。[55] TPC理事会。 [无日期。]。TPC-H主页。https://www.tpc.org/tpch/ [56] Immanuel Trummer。2022年。CodexDB：利用 GPT-3 Codex 根据自然语言指令合成查询处理代码。*VLDB* 汇刊 15，11（July 2022 2921 2928 10 14778 3551793 3551841 ），– 。https://doi.org/ . / . [57] 2022 Immanuel Trummer。 . DB-BERT：一种“阅读手册”的数据库调优工具。载于*2022*年国际数据管理会议论文集 数据 (美国宾夕法尼亚州费城) *(SIGMOD '22)*。美国纽约：计算机协会， 190 203 10 1145 3514221 3517843 – 。 [58] 2024 https://doi.org/ 。/ 。Immanuel Trummer。 。为大语言模型的高成本效益提示生成简洁的数据库模式描述。*VLDB Endow.* 论文集 17, 11 2024 3511 3523 10 14778 3681954 3682017 (7月)， – 。 [59] 2025 https://doi.org/ 。/ 。Immanuel Trummer。 。通过大语言模型定制 SQL 处理的算子实现。*IEEE*计算机学会数据工程技术委员会季刊 (2025年)。[60] 美国证券交易委员会。 [不适用]。财务报表数据集。https://www.sec.gov/files/aqfs.pdf。访问日期：2026-02-20。[61] Alexander van Renen、Dominik Horn、Pascal Pfeil、Kapil Vaidya、Wenjian Dong、Murali Narayanaswamy、Zhengchun Liu、Gaurav Saxena、Andreas Kipf 和 Tim Kraska。2024年。为何 TPC 还不够：对 Amazon Redshift 集群的分析。*VLDB Endow.* 论文集 17, 11（2024年8月），3694–3706。https://doi.org/10.14778/ 3681954.3682031[62] Jianguo Wang、Xiaomeng Yi、Rentong Guo、Hai Jin、Peng Xu、Shengjun Li、Xiangyu Wang、Xiangzhou Guo、Chengming Li、Xiaohai Xu 等。2021年。Milvus：专为特定用途构建的向量数据管理系统。载于*2021*年国际数据管理会议论文集。2614–2627。[63] Bobbi Yogatama、Yifei Yang、Kevin Kristensen、Devesh Sarda、Abigale Kim、Adrian Cockcroft、Yu Teng、Joshua Patterson、Gregory Kimball、Wes McKin- ney 等。2025年。重新思考 GPU时代的分析处理。*arXiv* 预印本 *arXiv:2508.04701* (2025年)。[64] Yan Zhou、Chunwei Liu、Bhuvan Urgaonkar、Zhengle Wang、Magnus Mueller、Chao Zhang、Songyue Zhang、Pascal Pfeil、Dominik Horn、Zhengchun Liu、Davide Pagano、Tim Kraska、Samuel Madden 和 Ju Fan。2025年。PBench：基于真实统计数据的云分析基准测试工作负载合成器。*VLDB Endow.* 论文集 18, 11 2025 3883 3895 10 14778 3749646 3749661 (7月)， – 。 [65] 2005 https://doi.org/ 。/ 。Marcin Zukowski、Peter A Boncz、Niels Nes 和 Sándor Héman。 。MonetDB/X100——CPU 缓存中的数据库管理系统。*IEEE* 数据工程简报 28, 2 （ 2005 年），17–22。