東南大學
SOUTHEAST UNIVERSITY

λ-Tune: Harnessing Large Language Models for Automated Database System Tuning

SIGMOD 2025

VICTOR GIANNAKOURIS[1], IMMANUEL TRUMMER[1],

[1]Cornell University

汇报人：董兵

2025年 9月 16日

Author

Research Interests

- Large Language Models & Data Systems
- Query optimization

- [1] Large-Scale Multiple Query Optimisation with Incremental Quantum(-Inspired) Annealing. SIGMOD ' 2026
- [2] SwellDB: Dynamic Query-Driven Table Generation with Large Language Models. SIGMOD ' 2025
- [3] Demonstrating SQLBarber: Leveraging Large Language Models to Generate Customized, Constraint-aware, and Realistic SQL. SIGMOD ' 2025

提纲

1 Background

2 Method

3 Evaluation

4 Conclusion

提纲

1 Background

2 Method

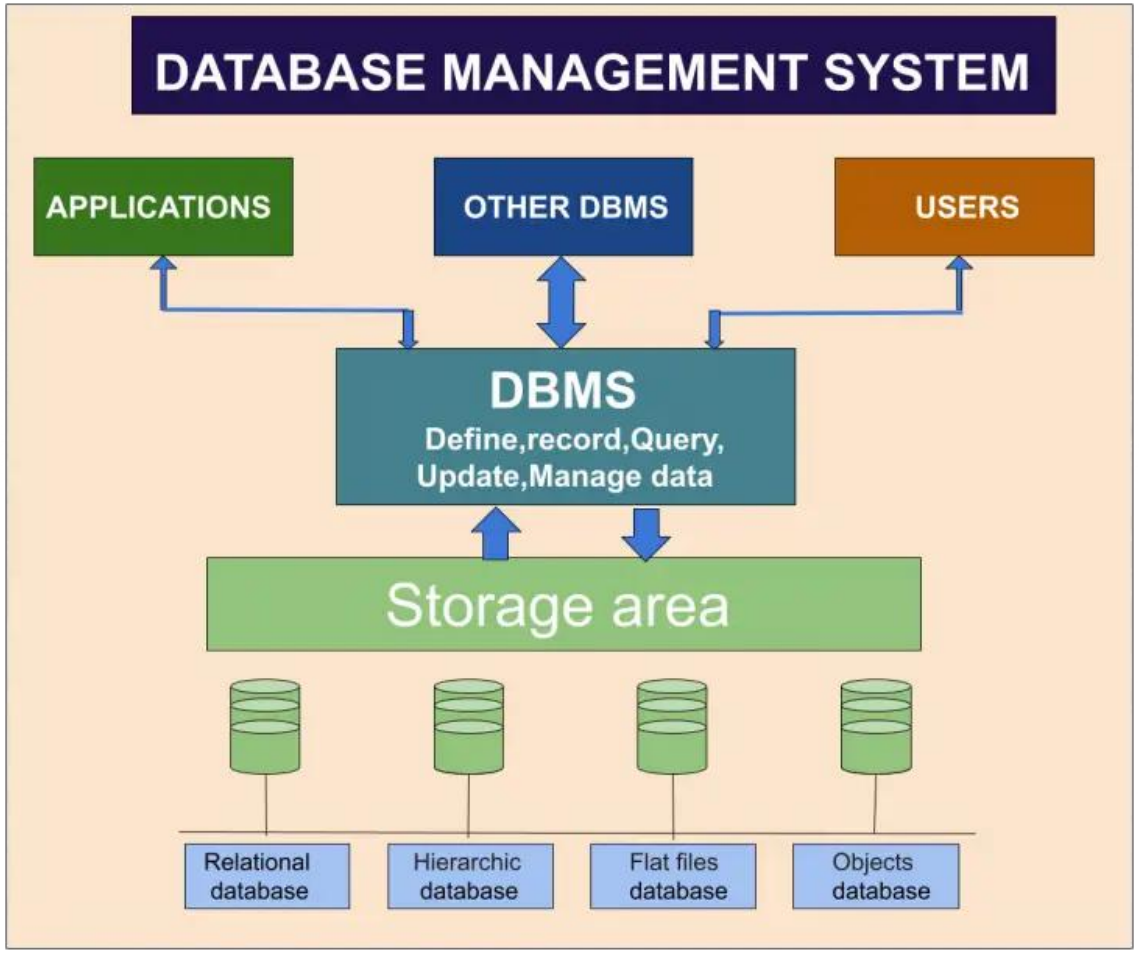
3 Evaluation

4 Conclusion

Background：DBMS

Database management system (DBMS)

- 用户（或应用程序）与实际存储的数据之间，充当一个中间层



Background：DBMS

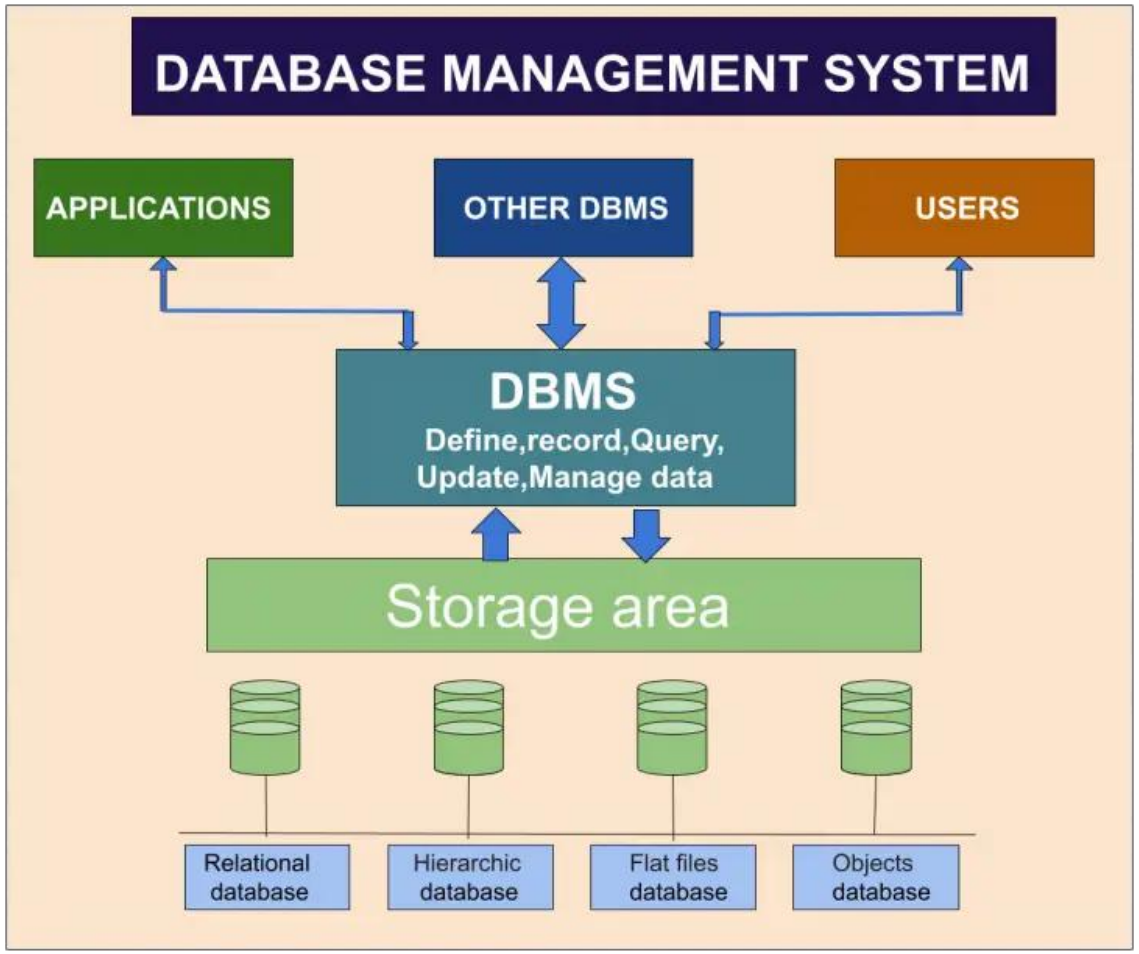
Database management system (DBMS)

- 用户（或应用程序）与实际存储的数据之间，充当一个中间层

✓ Define: 创建、修改和删除数据库、数据表以及定义它们之间的关系

✓ Manipulate：Create, Read, Update, Delete

✓ Control：数据的安全性，完整性，并发控制等



Background：DBMS

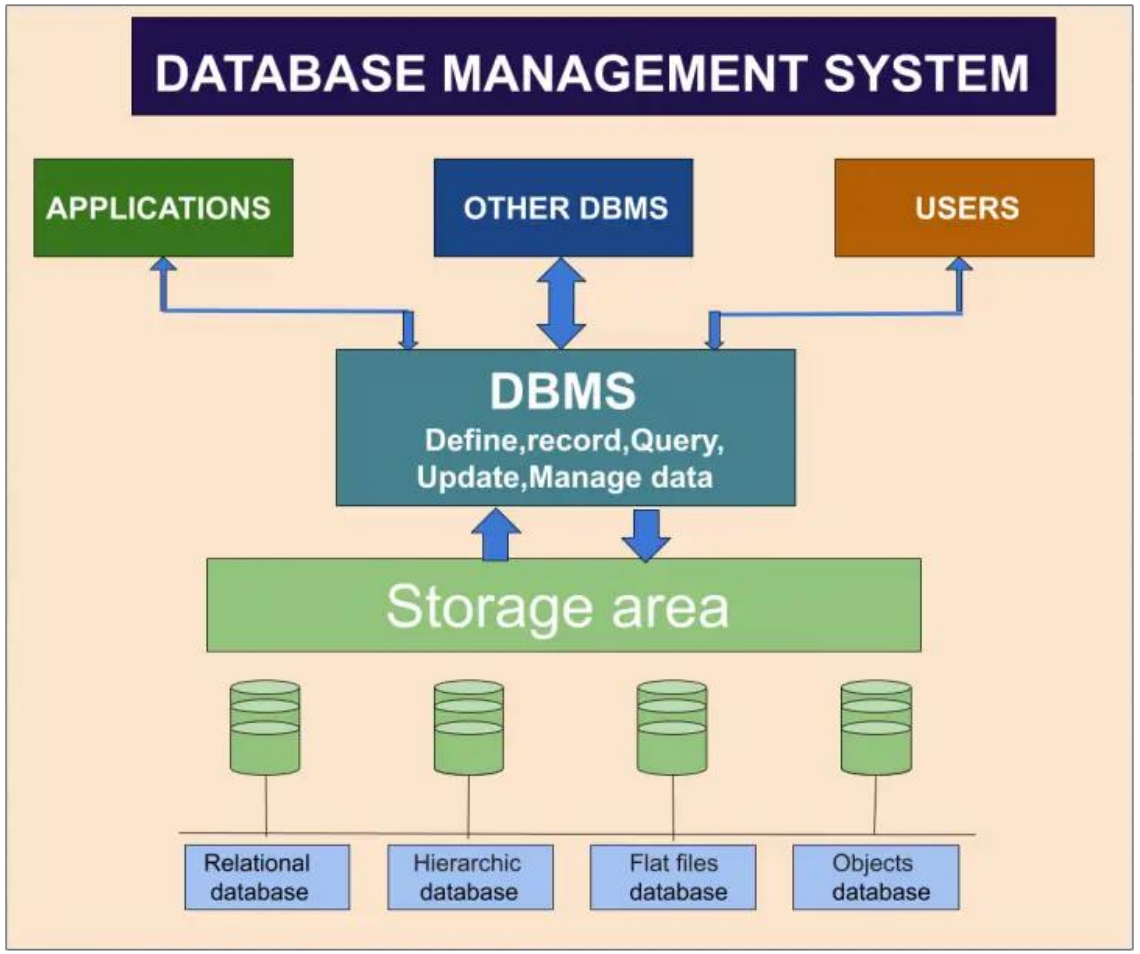
Database management system (DBMS)

- 用户（或应用程序）与实际存储的数据之间，充当一个中间层

✓ Define: 创建、修改和删除数据库、数据表以及定义它们之间的关系

✓ Manipulate：Create, Read, Update, Delete

✓ Control：数据的安全性，完整性，并发控制等



Background：DBMS Tuning

DBMS Tuning：通过调整Knobs，使其在特定硬件上为特定任务（工作负载）发挥出更好的性能。

System Configuration Parameters

- Memory：高速缓存，慢速磁盘
- **Query Optimizer**：选择最高效的查询执行计划，如执行顺序，高效的索引扫描
- I/O and Logging：并发的磁盘I/O请求

Background：DBMS Tuning

DBMS Tuning：通过调整Knobs，使其在特定硬件上为特定任务（工作负载）发挥出更好的性能。

System Configuration Parameters

- Memory：高速缓存空间，慢速磁盘
- **Query Optimizer**：选择最高效的查询执行计划，如执行顺序影响索引创建
- I/O and Logging：并发的磁盘I/O请求

Physical Design Choices

- **Indexing**：为经常被查询的列建立快速查找索引，**避免“全表扫描”**，极大地提升查询速度
- Sorting：改变数据在磁盘上的物理存放顺序，对范围查询比较有效
- Partitioning：对数据表分区，如时间，地区等

Background：DBMS Tuning

DBMS Tuning：通过调整Knobs，使其在特定硬件上为特定任务（工作负载）发挥出更好的性能。

- Indexing：为经常被查询的列建立快速查找索引，避免“全表扫描”，极大地提升查询速度
  - 额外创建，为特定列排好序的、包含指向原始数据行地址的 快速查找的 数据结构
  - 类似于操作系统中的页表

物理地址	user_id	user_name	city
地址-001	102	Bob	Shanghai
地址-002	103	Charlie	Guangzhou
地址-003	101	Alice	Beijing
...	...	...	...

Users Table

CREATE INDEX idx_name ON Users(user_name)

(排好序的)user_name	指向原始数据行的“物理地址”
Alice	地址-003
Bob	地址-001
Charlie	地址-002
...	...

索引文件: idx_name

Background：DBMS Tuning Method

传统的调优方法：基于成本模型

- 依赖数据库内部的“估算器”，通过数学和统计模型来预测一个调优决策（如使用哪个索引）的好坏

1）估算可能不准，导致决策错误；2）很多调优项（如内存参数）没有成本模型可用

Background：DBMS Tuning Method

传统的调优方法：基于成本模型

- 依赖数据库内部的“估算器”，通过数学和统计模型来预测一个调优决策（如使用哪个索引）的好坏

1）估算可能不准，导致决策错误；2）很多调优项（如内存参数）没有成本模型可用

基于机器学习

- 通过大量的实际测试（试运行）获得训练样本，让模型学习出哪个配置更好，如 OtterTune’ SIGMOD 2017，LlamaTune’ VLDB 2022

大量的试验收集训练样本，成本高昂，需要降维或者现有的成本模型来减少训练样本数量

Background：DBMS Tuning Method

基于大语言模型

- 通过解析经验文本文档（例如数据库手册）来为特定参数生成合适的数值建议，减少调优试验次数，如DB-BERT’ SIGMOD 2022，GPTuner’ VLDB 2024

针对特定单个参数生成配置提示，后续要考虑多个参数的组合优化问题

Background：DBMS Tuning Method

基于大语言模型

- 通过解析经验文本文档（例如数据库手册）来为特定参数生成合适的数值建议，减少调优试验次数，如DB-BERT’ SIGMOD 2022，GPTuner’ VLDB 2024
- 局限性：针对特定单个参数生成配置提示，后续要考虑多个参数的组合优化问题

LLM的快速发展（上下文长度等），使用LLM生成完整的配置，避免搜索组合爆炸

Background：DBMS Tuning Method

基于大语言模型

- 通过解析经验文本文档（例如数据库手册）来为特定参数生成合适的数值建议，减少调优试验次数，如DB-BERT’ SIGMOD 2022，GPTuner’ VLDB 2024
- 局限性：针对特定单个参数生成配置提示，后续要考虑多个参数的组合优化问题

LLM的快速发展（上下文长度等），使用LLM生成完整的配置，避免搜索组合爆炸

- **Challenge 1** 提示生成：工作负载（查询）非常庞大，需要考虑LLM输入限制，用户的预算限制

```
Recommend some configuration parameters for postgres to optimize the system's performance. Such
parameters might include system-level configurations, like memory, query optimizer or query-
level configuration, like index recommendations.
Each row in the following list has the following format:
{a join key A}:{all the joins with A in the workload}

aka_title.movie_id: ['movie_info.movie_id', 'title.id']
cast_info.person_id: ['aka_name.person_id', 'name.id']
char_name.id: [‘cast_info.person_role_id']
movie_companies.company_id: ['company_name.id']

The workload runs on a system with the following specs: memory: 61GiB cores: 8
```

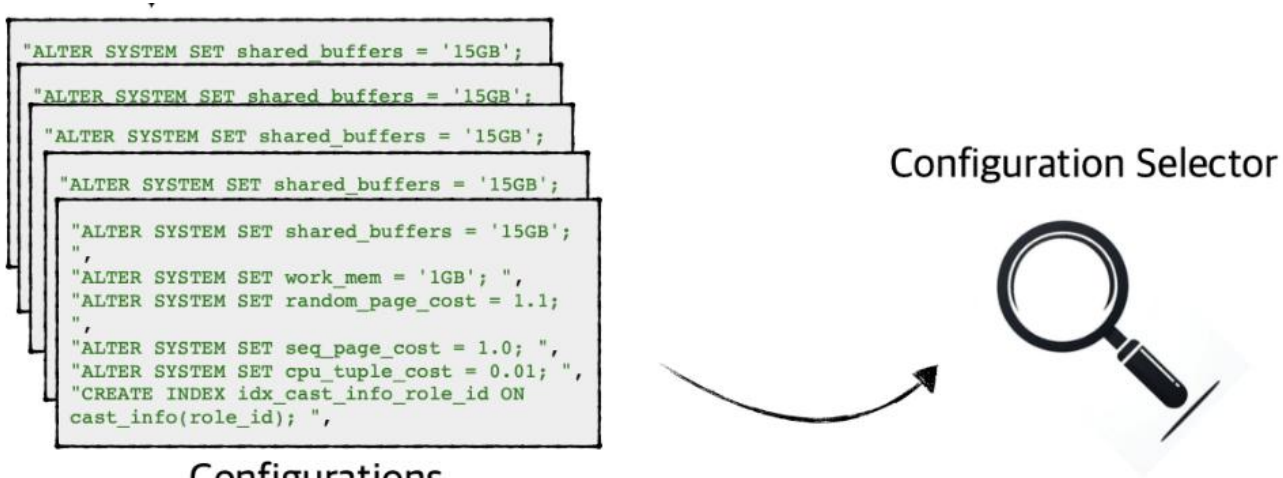
Background：DBMS Tuning Method

基于大语言模型

- 通过解析经验文本文档（例如数据库手册）来为特定参数生成合适的数值建议，减少调优试验次数，如DB-BERT’ SIGMOD 2022，GPTuner’ VLDB 2024
- 局限性：针对特定单个参数生成配置提示，后续要考虑多个参数的组合优化问题

LLM的快速发展（上下文长度等），使用LLM生成完整的配置，避免搜索组合爆炸

- **Challenge 2** 配置选择：LLM生成的配置质量可能“参差不齐“，按顺序评估这些配置会因为特别慢的配置而导致巨大的开销



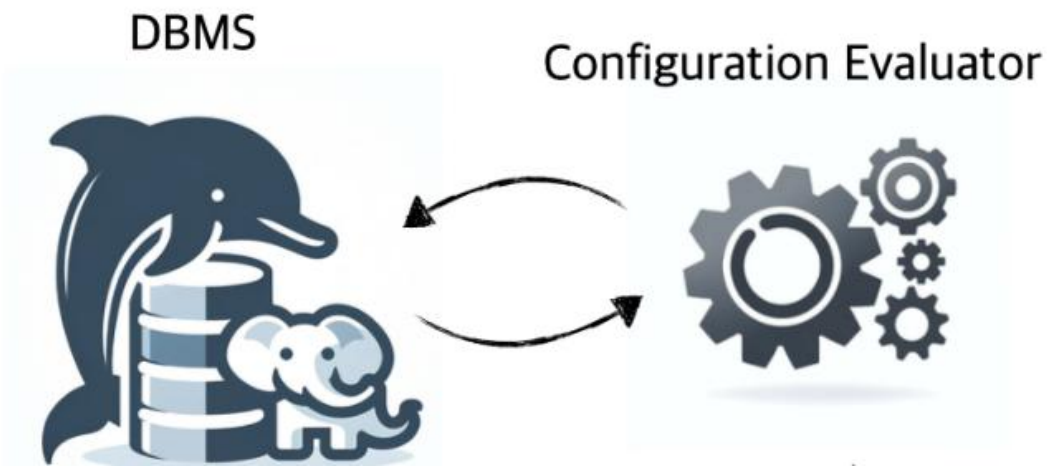
Background：DBMS Tuning Method

基于大语言模型

- 通过解析经验文本文档（例如数据库手册）来为特定参数生成合适的数值建议，减少调优试验次数，如DB-BERT’ SIGMOD 2022，GPTuner’ VLDB 2024
- 局限性：针对特定单个参数生成配置提示，后续要考虑多个参数的组合优化问题

LLM的快速发展（上下文长度等），使用LLM生成完整的配置，避免搜索组合爆炸

- **Challenge 3** 配置评估：对选择的配置进行测试评估时，如何高效评估查询，如顺序，索引创建等



提纲

1 Background

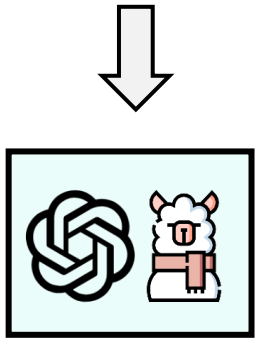
2 Method

3 Evaluation

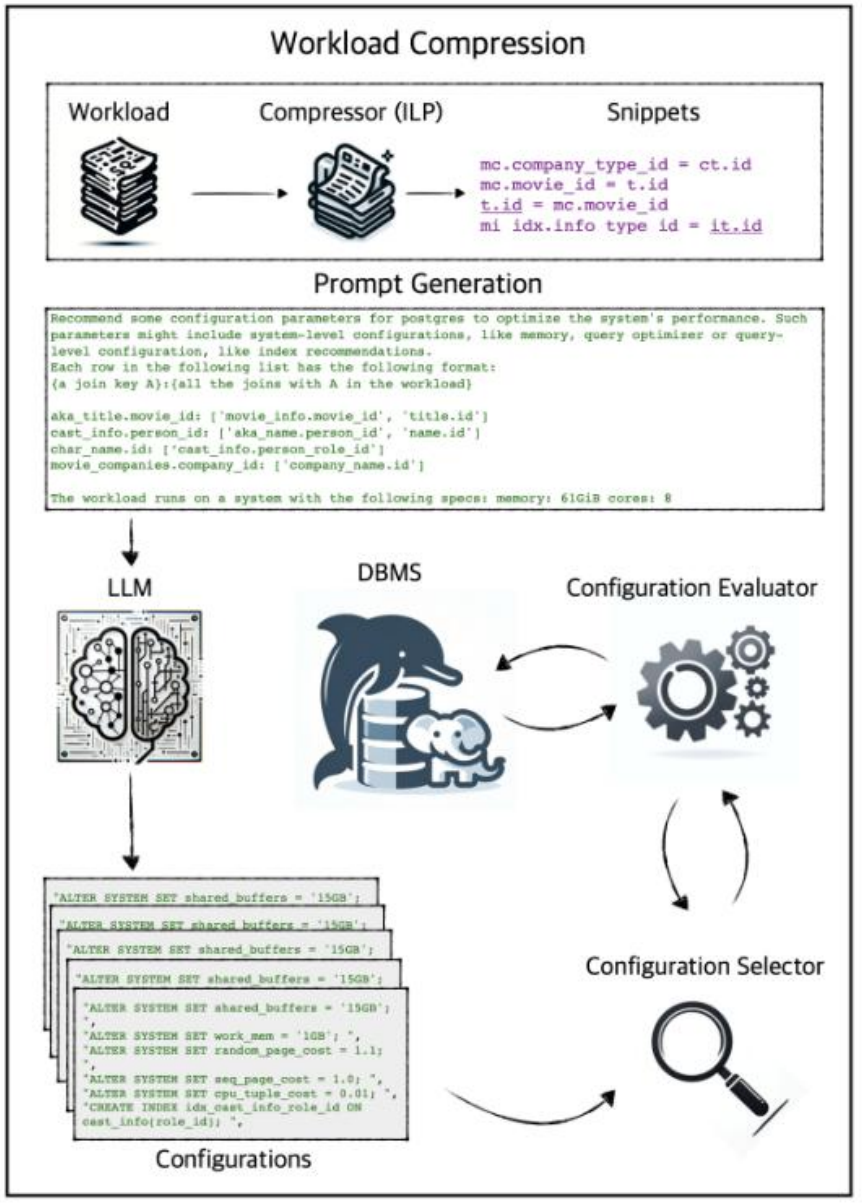
4 Conclusion

Method: Overview of λ-Tune

- Prompt Generation
  - ✓ Workload Compression

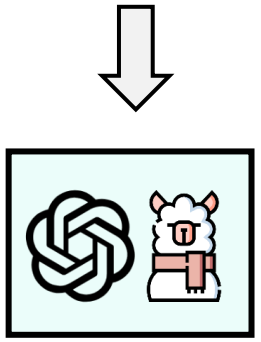


- Configurations
  - ✓ Configuration Selector
- Configuration Evaluator

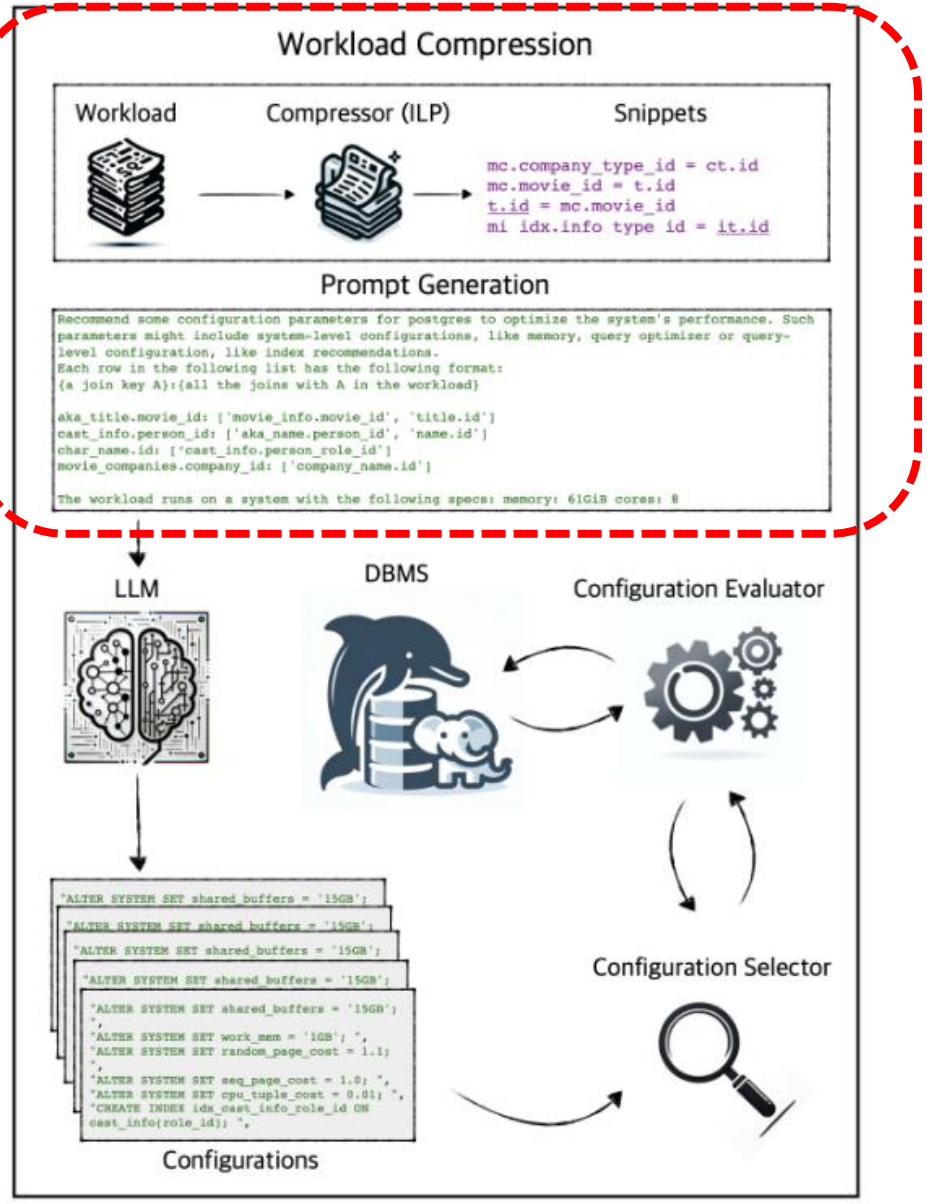


Method: Overview of λ-Tune

- Prompt Generation
  - ✓ Workload Compression



- Configurations
  - ✓ Configuration Selector
- Configuration Evaluator



Method：Prompt Generation

- Prompt Template

```
Recommend some configuration parameters for ${DBMS}$ to
optimize the system's performance. Parameters might
include system-level configurations, like memory,
query optimizer or physical design configurations,
like index recommendations.

Each row in the following list has the following format:
{a join key A}:{all the joins with A in the workload}
${COMPRESSED_WORKLOAD}$

The workload runs on a system with the following specs:
memory: ${MEMORY}
cores: ${CORES}
```

- ✓ DBMS：要调优的目标数据库管理系统的名称（例如，PostgreSQL或MySQL）
- ✓ **Workload Compression**：专注于优化查询描述中的join连接操作
- ✓ 系统硬件属性：内存大小和CPU核心数

Method：Prompt Generation

- **Challenge 1** 提示生成：工作负载（查询）非常庞大，需要考虑LLM输入限制，用户的预算限制

  - ✓ **Workload Compression**：描述应侧重于数据不同部分之间的二元关系，如同一查询中表的共现，或出现在相同连接（join）条件中的列对之间的联系， 这类信息对于数据分区、索引和复制等调优决策非常重要。
  - ✓ 连接（join）通常是开销最大的算子之一，压缩连接开销的描述信息


```
SELECT Users.Name, Orders.Product
FROM Users
JOIN Orders ON Users.UserID = Orders.UserID;
```

Method： Prompt Generation

- **Challenge 1** 提示生成：工作负载（查询）非常庞大，需要考虑LLM输入限制，用户的预算限制

  ✓ **Workload Compression**： 描述应侧重于数据不同部分之间的二元关系，如同一查询中表的共现，或出现在相同连接（join）条件中的列对之间的联系， 这类信息对于数据分区、索引和复制等调优决策非常重要。

  ✓ 连接（join）通常是开销最大的算子之一 ， 专注查询中连接的描述信息压缩

Naïve：将连接集合都表示为一个独立的队列 <A, B>, <A, C>, <A, D>

合并公共列：A：B，C，D -> join snippets

Method： Prompt Generation

- **Challenge 1** 提示生成：工作负载（查询）非常庞大，需要考虑LLM输入限制，用户的预算限制

  - ✓ **Workload Compression**： 描述应侧重于数据不同部分之间的二元关系，如同一查询中表的共现，或出现在相同连接（join）条件中的列对之间的联系， 这类信息对于数据分区、索引和复制等调优决策非常重要。
  - ✓ 连接（join）通常是开销最大的算子之一 ， 专注查询中连接的描述信息压缩

✗ Naïve：将连接集合都表示为一个独立的队列 <A, B>, <A, C>, <A, D>

✓ 合并公共列：A：B，C，D -> join snippets

LLM输入限制，用户预算限制：仍然无法表示具有多样化连接条件的大型工作负载

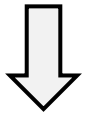
✓ 并非所有join都同等重要，优先选择开销较大，更耗时的join信息输入

Method： Prompt Generation

- **Challenge 1** 提示生成：工作负载（查询）非常庞大，需要考虑LLM输入限制，用户的预算限制

  - LLM输入限制，用户预算限制：仍然无法表示具有多样化连接条件的大型工作负载
    - ✓ 并非所有join都同等重要，优先选择开销较大，更耗时的join信息输入

基于成本模型选择：利用 EXPLAN工具 估计每个join片段进行成本得到 $V\ (p)$



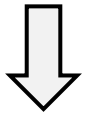
给定token预算，选择一组join片段，使得所有片段的总成本$V\ (p)$最高

Method： Prompt Generation

- **Challenge 1** 提示生成：工作负载（查询）非常庞大，需要考虑LLM输入限制，用户的预算限制

  - LLM输入限制，用户预算限制：仍然无法表示具有多样化连接条件的大型工作负载
    - ✓ 并非所有join都同等重要，优先选择开销较大，更耗时的join信息输入

基于成本模型选择：利用 EXPLAN工具 估计每个join片段进行成本得到 $V\ (p)$



给定token预算，选择一组join片段，使得所有片段的总成本$V\ (p)$最高

01背包问题（01 knapsack problem）：一共有N件物品，第i（i从1开始）件物品的重量为w[i]，价值为v[i]。在总重量不超过背包承载上限W的情况下，能够装入背包的最大价值是多少?

Method： Prompt Generation

- **Challenge 1** 提示生成：工作负载（查询）非常庞大，需要考虑LLM输入限制，用户的预算限制

  - 基于成本模型选择：利用 EXPLAN工具 估计每个join片段进行成本得到 $V\ (p)$
    - ✓ 给定token预算，选择一组join片段，使得这组片段的总成本$V\ (p)$最高

Variable		Semantics
$R_{\langle c_1,c_2\rangle} \in \{0,1\}$	A：B，C，D	Binary variable which denotes if c_2 appears on the right-hand side of c_1.
$L_c \in \{0,1\}$		Binary variable which denotes whether the left-hand side column c is included in the prompt.

Constraint		Semantics
$R_{\langle c_1,c_2\rangle} \leq L_{c_1}$	先有左侧，才能有右侧	：B，C，D ✗
$L_{c_1} \leq \sum_{\langle c_1,c_2\rangle \in P} R_{\langle c_1,c_2\rangle}$	避免“空行“	A： ✗
$R_{\langle c_1,c_2\rangle} + R_{\langle c_2,c_1\rangle} < 2$	避免“重复“	A：B，B：A ✗

Method： Prompt Generation

- **Challenge 1** 提示生成：工作负载（查询）非常庞大，需要考虑LLM输入限制，用户的预算限制

  - 基于成本模型选择：利用 EXPLAN工具 估计每个join片段进行成本得到 $V\ (p)$
    - ✓ 给定token预算，选择一组join片段，使得这组片段的总成本$V\ (p)$最高

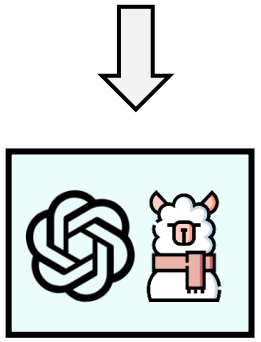
$$\text{Maximize:} \quad \sum_{p \in P} V(p) R_p$$

$$\text{Subject to:} \quad \sum_{\langle c_1, c_2 \rangle \in P} H_{c_2} \cdot R_{\langle c_1, c_2 \rangle} + \sum_{c \in C} H_c \cdot L_c \leq \mathcal{B}$$

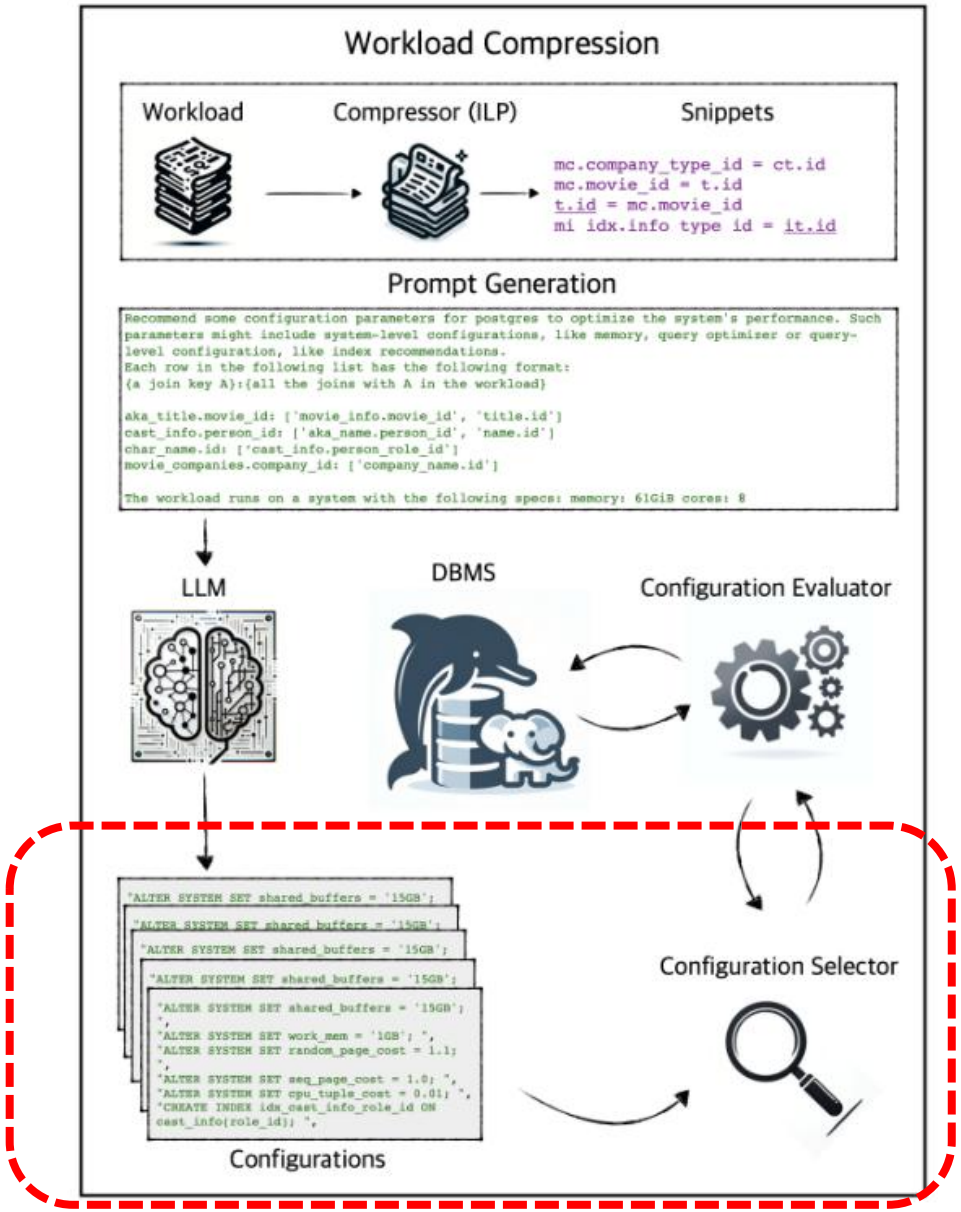
- **背包容量**： prompt固定的token预算 B
- **物品**： 每一个可能的join片段（如 A：B，C，D）
- **物品的重量**： 每个片段占用的token数量（H）
- **物品的价值**： 每个片段的估算开销（V）

Method: Overview of λ-Tune

- Prompt Generation
  - ✓ Workload Compression



- Configurations
  - ✓ Configuration Selector
- Configuration Evaluator



Method：Configuration Selector

- **Challenge 2** 配置选择：LLM生成的配置质量可能参差不齐，按顺序评估这些配置会因为特别慢的配置而导致巨大的开销

Incremental Timeouts：分轮次进行评估，并在每轮为每个配置设置一个超时时间

Method：Configuration Selector

- **Challenge 2** 配置选择：LLM生成的配置质量可能参差不齐，按顺序评估这些配置会因为特别慢的配置而导致巨大的开销

Incremental Timeouts：分轮次进行评估，并在每轮为每个配置设置一个超时时间

- 第一轮：给所有配置一个很短的初始超时时间，比如10分钟。10分钟后，强制停止所有配置的测试。
  - ✓ 结果：那些差的配置可能一个查询都没跑完，在这些配置上只浪费了10分钟。而表现好的配置可能已经跑完了一部分查询。
- 第二轮：将超时时间加倍（乘以某个因子α），比如延长到20分钟。让所有配置从上一轮停止的地方继续跑。
- 后续轮次：超时时间继续以倍数（例如10, 20, 40, 80分钟...）增加，直到至少有一个最优配置率先完成了所有的查询任务。

Method： Configuration Selector

- **Challenge 2** 配置选择：LLM生成的配置质量可能参差不齐，按顺序评估这些配置会因为特别慢的配置而导致巨大的开销

Incremental Timeouts：分轮次进行评估，并在每轮为每个配置设置一个超时时间

为什么没有多配置并行评估?

- 测试结果准确性：数据库系统调优是为了找到在特定硬件上表现最好的配置，每个配置都应该能完全利用这台机器的所有资源。
- 高昂的重配置开销：切换配置，创建索引是非常耗时的操作，开销甚至超过查询本身的执行时间

Method： Configuration Selector

- **Challenge 2** 配置选择：LLM生成的配置质量可能参差不齐，按顺序评估这些配置会因为特别慢的配置而导致巨大的开销

Incremental Timeouts：分轮次进行评估，并在每轮为每个配置设置一个超时时间

Avoiding Redundancy：为每个配置跟踪已完全处理的查询，前面轮次执行完的查询不用再处理

Best Configuration：当找到一组率先完成所有查询的配置时，使用**这个最佳配置的执行时间**，减去该配置**已用于完全评估查询（没有被中断）**的时间，作为新的超时时间（ **best.time - time**）

Method： Configuration Selector

- **Challenge 2** 配置选择：LLM生成的配置质量可能参差不齐，按顺序评估这些配置会因为特别慢的配置而导致巨大的开销

Incremental Timeouts：分轮次进行评估，并在每轮为每个配置设置一个超时时间

Avoiding Redundancy：为每个配置跟踪已完全处理的查询，前面轮次执行完的查询不用再处理

Best Configuration：当找到一组率先完成所有查询的配置时，使用**这个最佳配置的执行时间**，减去该配置**已用于完全评估查询（没有被中断）**的时间，作为新的超时时间（ **best.time - time**）

每个配置的评估记录：configMeta

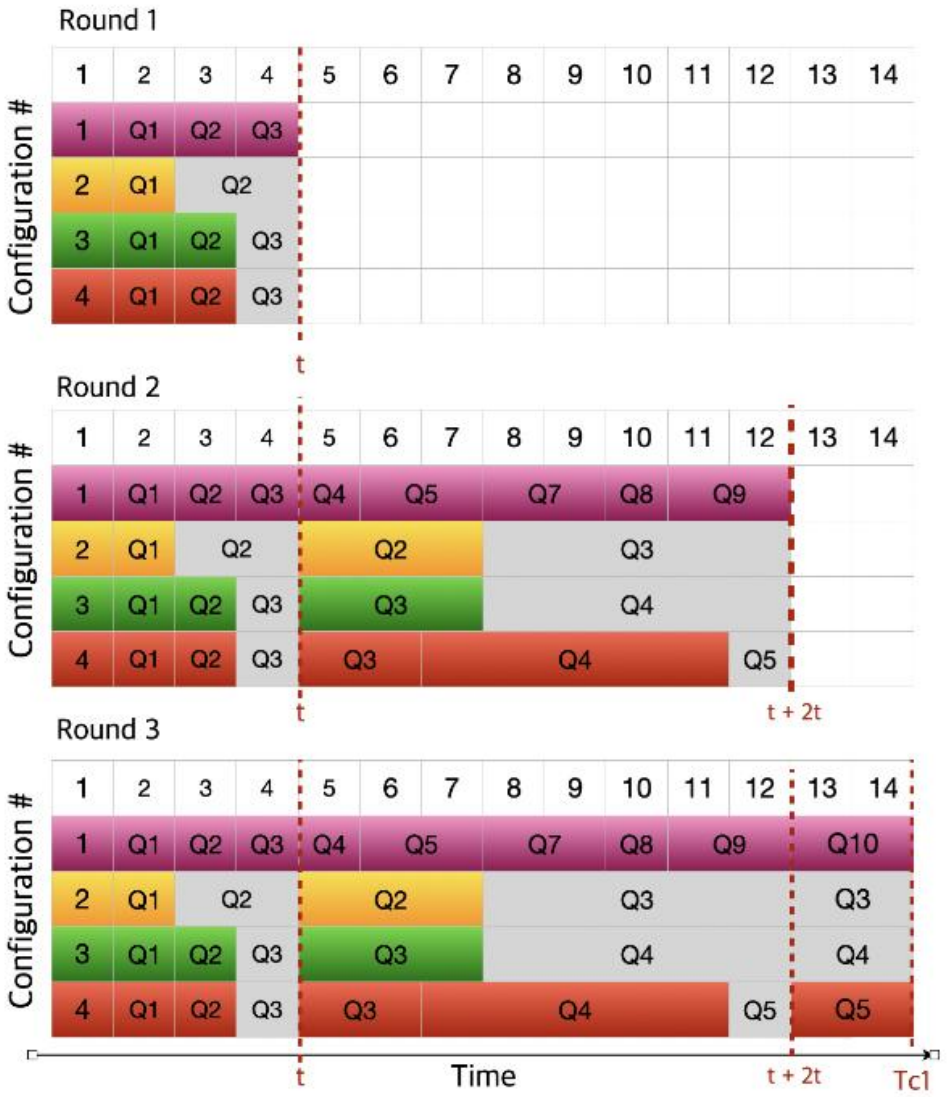
time：已经完成的查询所消耗的时间

Completed Queries：添加已经处理完的查询，因超时而中断的查询不会被添加进来

isComplete：配置是否完成

indexTime：索引创建的时间

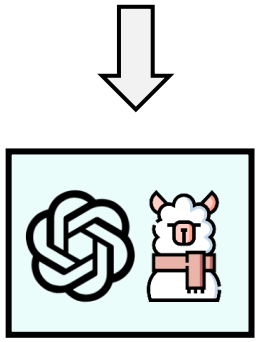
Method： Configuration Selector



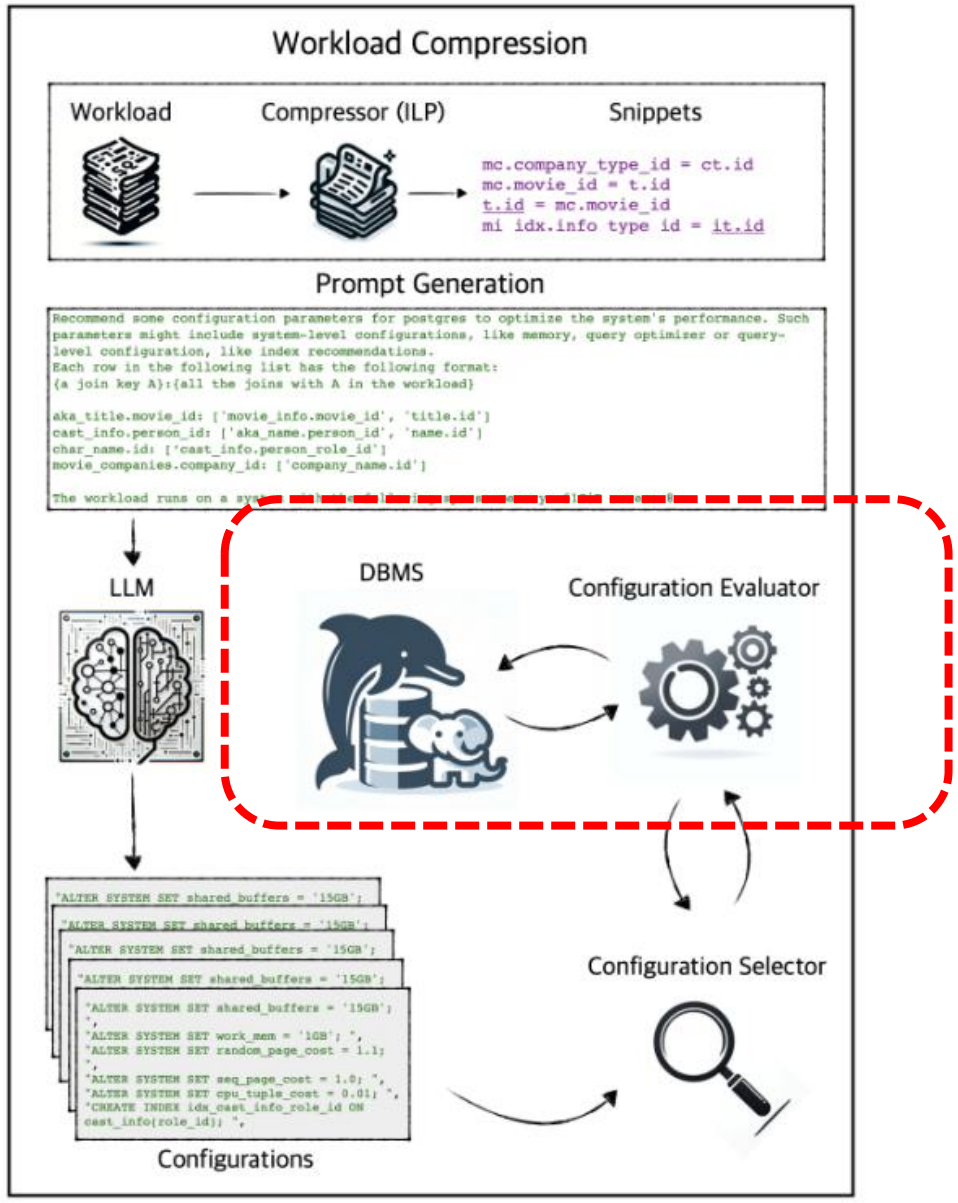
- Initialize：
  - W: workload
  - C: configuration sets
  - best.time = ∞
  - Timeout = t
  - α: 2
- Until first configuration finishes：
  - $Update\ (c, W\ , configsMeta, t, best)$
  - best.time = Tc1
- Check remaining configurations：
  - t = Tc1 – 已完成查询的时间
  - $Update\ (c, W\ , configsMeta, t, best)$

Method: Overview of λ-Tune

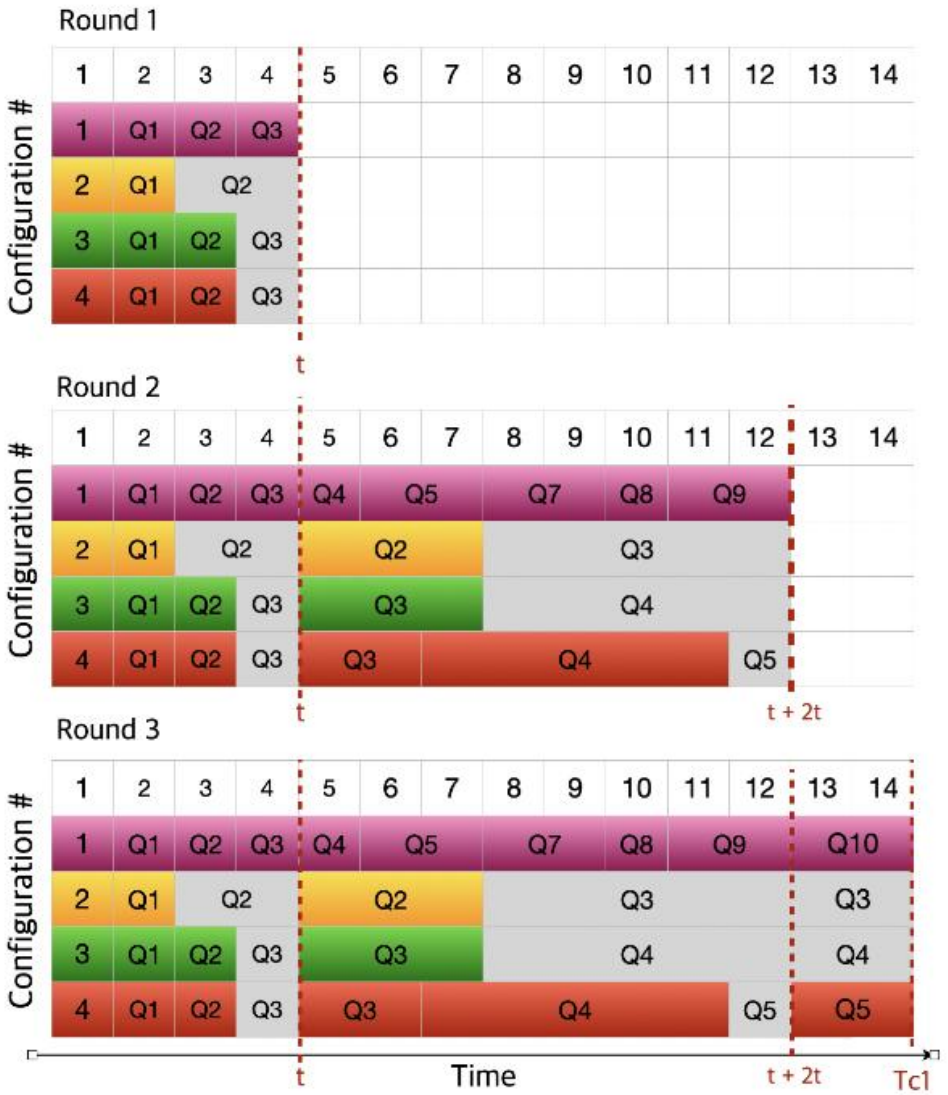
- Prompt Generation
  - ✓ Workload Compression



- Configurations
  - ✓ Configuration Selector
- Configuration Evaluator



Method： Configuration Evaluator



- Initialize：
  - W: workload
  - C: configuration sets
  - best.time = ∞
  - Timeout = t
  - α: 2
- Until first configuration finishes：
  - $Update\ (c, W\ , configsMeta, t, best)$
  - best.time = Tc1
- Check remaining configurations：
  - t = Tc1 – 已完成查询的时间
  - $Update\ (c, W\ , configsMeta, t, best)$

Evaluate Workload

Method： Configuration Evaluator

- **Challenge 3** 配置评估：对选择的配置进行测试评估时，如何高效评估查询，如顺序，索引创建等

creates indexes lazily：只创建那些与下一个查询可能相关的索引

- 创建索引是非常耗时的操作，开销甚至超过查询本身的执行时间
- 由于存在超时中断，如果那些后续的查询没有被执行，那么为它们创建相关的索引是一种浪费。

Method： Configuration Evaluator

- **Challenge 3** 配置评估：对选择的配置进行测试评估时，如何高效评估查询，如顺序，索引创建等

Cost Model for Query Scheduling：计算创建索引的期望成本，得到最优的查询执行顺序

Method： Configuration Evaluator

- **Challenge 3** 配置评估：对选择的配置进行测试评估时，如何高效评估查询，如顺序，索引创建等

Cost Model for Query Scheduling：计算创建索引的期望成本，得到最优的查询执行顺序

- 处理查询的顺序可能对创建索引的开销有很大影响，尤其是中断会影响后面的查询成本

示例：假设处理两个查询，q1和q2。每个查询只能使用一个其特有的索引，**为q1创建索引的成本是1，为q2创建索引的成本是5**。假设两个查询的运行时间相同，且无论哪个查询先执行，它**有50%的概率被中断**。

- 如果采用**q1−q2**的顺序，创建索引的**期望成本**是 $1 + 0.5 \cdot 5 = 3.5$。
- 反之，如果采用**q2−q1**的顺序，**期望成本**是 $5 + 0.5 \cdot 1 = 5.5$。

Method： Configuration Evaluator

- **Challenge 3** 配置评估：对选择的配置进行测试评估时，如何高效评估查询，如顺序，索引创建等

Cost Model for Query Scheduling：计算创建索引的期望成本，评估查询顺序好坏

- 处理查询的顺序可能对创建索引的开销产生重大影响，尤其是中断会影响后面的查询成本

$$1/n \cdot \sum_{1 \le k \le n} \sum_{1 \le j \le k} z_{i_j}(\{q_{i_1}, \ldots, q_{i_{j-1}}\})$$

假设在每个查询之后发生中断的概率是均等的，1/n

- 外层求和，表示遍历了所有可能中断的情况（如k=2，表示执行完第2个查询后就中断）
- 内层求和，计算在中断点 k发生时，已经花费的创建索引的期望成本
- Z，计算增量成本，如果 j 所需的索引已被创建，成本就为 0

Method： Configuration Evaluator

- **Challenge 3** 配置评估：对选择的配置进行测试评估时，如何高效评估查询，如顺序，索引创建等

Optimizing Query Order：通过动态规划算法快速得到一组最优的查询评估顺序

- **对于一组n个查询，有 n! 种顺序，直接暴力搜索评估效率太低**

最优性原则

- 如果前 k 个查询已经排好序，那么添加下一个查询（从n-k个剩余查询中选择）的成本与前 k 个查询的内部顺序无关。
- 改变前k个查询的顺序以降低k个查询的成本，不会使整体成本变得更差
- DP：维护查询子集的最优成本，将每一个查询q作为某个子集合的结尾

$$C(S) = \min_{q \in S}\{C(S - \{q\}) + \text{cost}(q, S - \{q\})\}$$

Method： Configuration Evaluator

Algorithm 4: DP Query Scheduling

```
1  Function ComputeOrderDP(W, I):
       /* W: The set of queries
       /* I: A hashmap containing the indexes for each query
2      dpCost ← {}
3      dpOrder ← {}
4      for q ∈ W do
5          indexes ← I[q]
6          dpOrder[{q}] ← [q]
7          dpTable[{q}] ← cost(indexes)
8      for i = 2 .. i ≤ n do
           /* Enumerate all subsets of size i
9          for subset ⊆ W : |subset| = i do
10             dpCost[subset] ← ∞
11             for query ∈ subset do
12                 subset′ ← subset − query
13                 queryCost ← computeCost(query, subset′)
14                 c ← dpTable[subset′] + queryCost
15                 if c < dpCost[subset] then
16                     dpCost[subset] ← c
17                     dpOrder[subset] ← dpOrder[subset′] ∘ query
18     return dpOrder[set(W)]
```

- Initialize（4-7）：
  - 子集合为1：等于自身

Method： Configuration Evaluator

Algorithm 4: DP Query Scheduling

```
1  Function ComputeOrderDP(W, I):
       /* W: The set of queries
       /* I: A hashmap containing the indexes for each query
2      dpCost ← {}
3      dpOrder ← {}
4      for q ∈ W do
5          indexes ← I[q]
6          dpOrder[{q}] ← [q]
7          dpTable[{q}] ← cost(indexes)
8      for i = 2 .. i ≤ n do
           /* Enumerate all subsets of size i
9          for subset ⊆ W : |subset| = i do
10             dpCost[subset] ← ∞
11             for query ∈ subset do
12                 subset′ ← subset − query
13                 queryCost ← computeCost(query, subset′)
14                 c ← dpTable[subset′] + queryCost
15                 if c < dpCost[subset] then
16                     dpCost[subset] ← c
17                     dpOrder[subset] ← dpOrder[subset′] ∘ query
18     return dpOrder[set(W)]
```

- Initialize（4-7）：
  - 子集合为1：等于自身
- 动态规划（8-17）：
  - 子集合从2 到 n
  - 计算子集合中以 q 为结尾的索引创建期望成本
  - 维护更新子集合的最低成本
- 循环完所有的子集合后，dpOrder中包含了针对所有查询的最优评估顺序

Method： Configuration Evaluator

- **Challenge 3** 配置评估：对选择的配置进行测试评估时，如何高效评估查询，如顺序，索引创建等

Query Clustering：动态规划算法具有指数级复杂度，通过聚类分组减少输入的规模

工程上的实现问题

- 将查询向量化：每个索引都被分配一个唯一的编号 i，而每个查询 q 则被表示为一个二进制向量，其中第 i 个元素表示 q 是否引用了索引 i（如果是则为1，否则为0）。
  - Q1 需要索引A和C => [1, 0, 1]
- 欧几里得距离作为聚类距离度量 $d(I, I') = \sqrt{\sum_i^n (I_i - I'_i)^2}.$ ，使用K-Means对查询进行聚类，同组的看作一个整体，如[{A}，{B,C}]

提纲

1 Background

2 Method

3 Evaluation

4 Conclusion

Evaluation：Setup

实验平台: EC2 p3.2xlarge（AWS）

Baselines:

- 基于LLM：GPTuner （ VLDB’ 24）、DB-BERT（SIGMOD’ 22）
- 基于强化学习：UDO （ VLDB’ 21）
- 基于降维：LlamaTune （ VLDB’ 22）
- 专用索引选择工具：Dexter，DB2（ VLDB’ 23）
- 专用优化器参数调优系统：ParamTree（PostgreSQL，SIGMOD’ 23）

Benchmark：

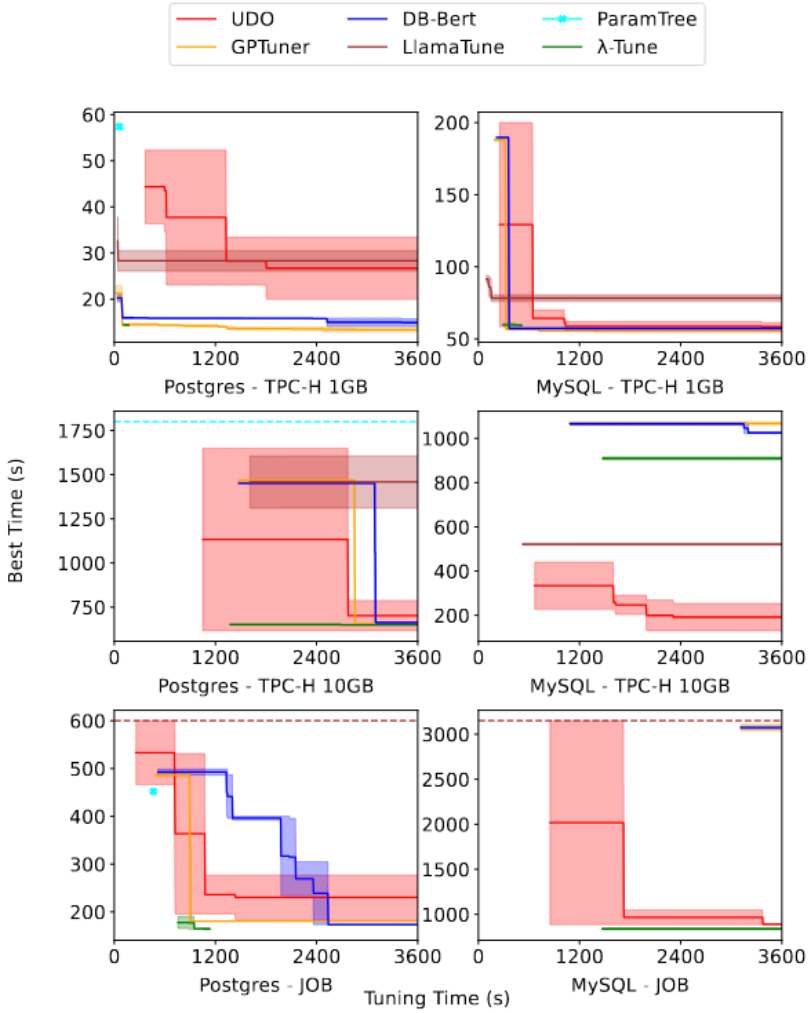
- TPC-DS：模拟了一个批发供应商的业务环境（8张数据表）
- TPC-H：模拟了一个大型零售企业的业务（24张表），更加复杂
- Join Order Benchmark (JOB)：专门为测试数据库查询优化器中Join排序

Evaluation: Tuning Time

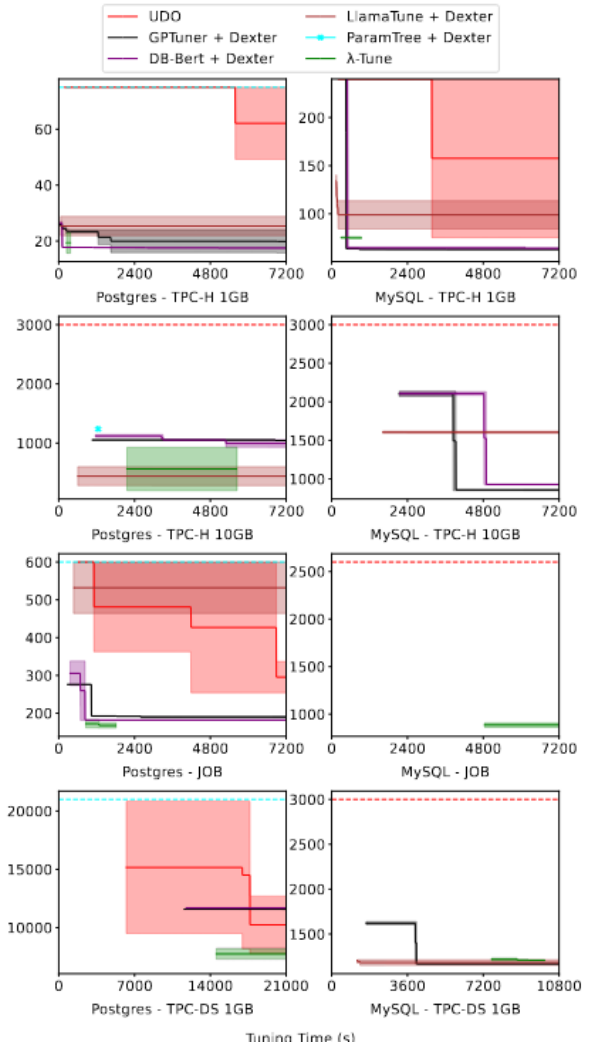
Benchmark	DBMS	Initial Indexes	λ-Tune	UDO	DB-Bert	GPTuner	LlamaTune	ParamTree
TPC-H 1GB	PG	Yes	1.07	1.96	1.13	1	2.08	3.23
TPC-H 1GB	MS	Yes	1.06	1	1.02	1.73	1.39	3.24
TPC-H 10GB	PG	Yes	1.03	1	1.05	1.04	2.38	3.18
TPC-H 10GB	MS	Yes	4.98	1	5.16	5.84	2.86	15.2
JOB	PG	Yes	1	1.32	1.05	1.1	3.48	3.48
JOB	MS	Yes	1	1.07	3.69	3.69	3.22	3.22
TPC-H 1GB	PG	No	1.05	3.76	1	1.06	1.43	4.24
TPC-H 1GB	MS	No	1.2	2.83	1.02	1	1.61	3.64
TPC-H 10GB	PG	No	1.65	1.54	2.45	2.52	1	1.54
TPC-H 10GB	MS	No	1.04	3.2	1.09	1	1.88	3.2
JOB	PG	No	1	1.69	1.08	1.13	3.09	3.26
JOB	MS	No	1	3.07	3.07	3.07	3.07	3.07
TPC-DS	PG	No	1	1.37	1.67	1.66	3.33	3.33
TPC-DS	MS	No	1.79	3.25	1	1.03	1.05	3.25
Average			**1.41**	**2.00**	**1.82**	**1.91**	**2.27**	**4.07**

Evaluation： Tuning Time

- not Create Indexes (Pure Parameter Tuning)



- Create Indexes



提纲

1 Background

2 Method

3 Evaluation

4 Conclusion

Conclusion

- 使用LLM解析经验文档并生成完整的配置，避免了搜索组合爆炸问题，主要针对 join 操作 和 索引创建
- Prompt Generation
  - Workload Compression：合并join中的共享列，如A：B,C,D
  - 背包问题：在给定token预算下，计算每个join片段所需的评估开销
- Configuration Selector：
  - Incremental Timeouts：为避免在评估差配置上花费过多时间，评估分轮次进行，并在每轮为每个配置设置一个超时时间
- Configuration Evaluator
  - Cost Model for Query Scheduling：计算索引创建的期望成本（中断概率都是 1 / n）
  - Optimizing Query Order：找到最优的查询评估顺序

Thinking

- 能否提高
  - 实现迭代式调优：将评估结果反馈给LLM，进行多轮的“对话式”调优。
  - 结合RAG

- 泛化：serverless中的参数调优，内存分配，Batch Size，超时设置

- 用到我们的idea中：并没有与LLM结合很深，主要是配置选择，查询顺序的 选择算法优化

東南大學
SOUTHEAST UNIVERSITY

恳请各位老师与同学批评指正！