

GPTuner: A Manual-Reading Database Tuning System via GPT- Guided Bayesian Optimization

Jiale Lao¹, Yibo Wang¹, Yufei Li¹, Jianping Wang²,
Yunjia Zhang³, Zhiyuan Cheng⁴, Wanghu Chen²,
Mingjie Tang¹, Jianguo Wang⁴

Reporter: Yuxiang Lu

¹ Sichuan University

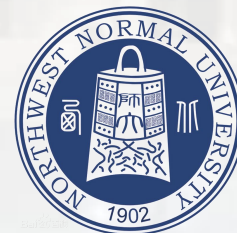
² Northwest Normal University

³ University of Wisconsin-Madison

⁴ Purdue University



IDS Lab
Intelligence & Database System Lab

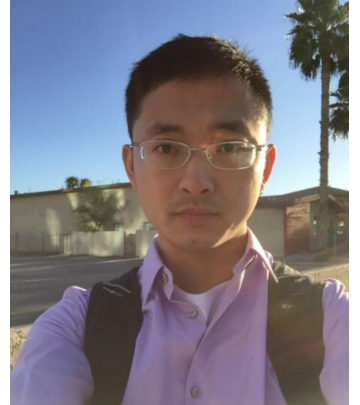


2025.3.21

Author Information

- **Tang Mingjie**

Mingjie Tang is currently working on LLM systems and algorithms. He has broad research interests in RDBMS, distributed machine learning systems, big data computation engines, and distributed deep learning systems.



IDS Lab (Intelligence and Database System Lab) is from the Department of Computer Science at SCU, led by Prof. Mingjie Tang.

The team at IDS Lab focus on the following research areas:

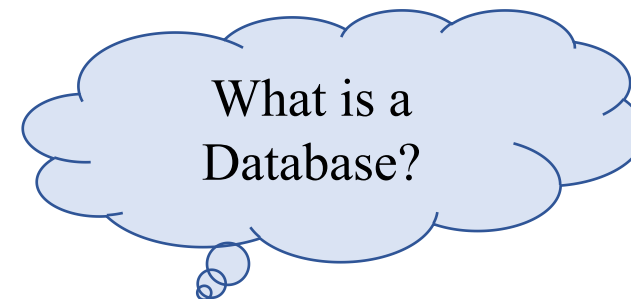
- Machine Learning and Deep Learning Systems
- Databases Systems
- Artificial Intelligence Security



- **Background**
- **Problem**
- **Design**
- **Evaluation**
- **Conclusion**



- **Background**
- Problem
- Design
- Evaluation
- Conclusion



GPTuner: A Manual-Reading **Database Tuning System** via GPT- Guided Bayesian Optimization

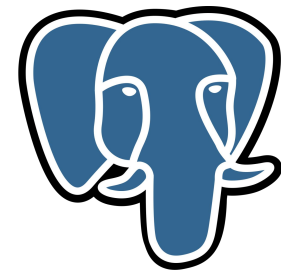
A database is an organized collection of data or
a type of data store based on the use of a
database management system (DBMS)¹.

¹ Wikipedia - <https://en.wikipedia.org/wiki/Database>

Background

Connolly and Begg define database management system (DBMS) as a **"software system that enables users to define, create, maintain and control access to the database."**

Examples of DBMS's include MySQL, MariaDB, PostgreSQL, Microsoft SQL Server, Oracle Database, and Microsoft Access.



PostgreSQL

Background



Modern Database Management Systems (DBMS) expose hundreds of configurable parameters (i.e., **knobs**) to control their runtime behaviours.

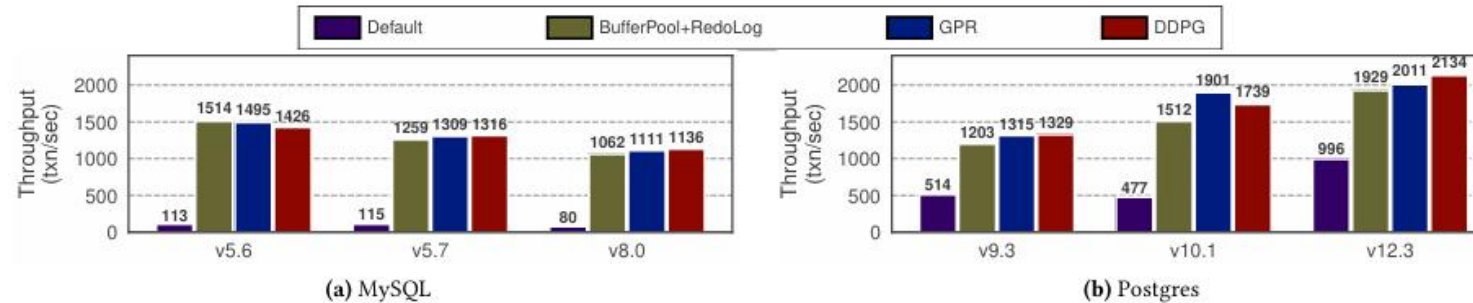
	MySQL	PostgreSQL
Memory Management	innodb_buffer_pool_size	shared_buffers
Query Optimization	optimizer_switch	work_mem
Logging & Durability	sync_binlog	wal_level
Concurrency Control	innodb_thread_concurrency	max_worker_processes



- Background
- **Problem**
- Design
- Evaluation
- Conclusion

Problem

The selection of appropriate values for these knobs is **crucial**.



The DBMS knobs tuning problem is **difficult**!

Increasing number of knobs

Each knob could be in continuous or categorical values

Oracle's Cloud Business Group data reveals that database administrators (DBA) spend an average of **90 percent of their time** on maintenance tasks like knob tuning.

Problem

- **Manual tuning is not acceptable**
- state-of-the-art approaches automate the knob tuning via **Machine Learning (ML) techniques**
- However, the state-of-the-art systems still require **hundreds to thousands** iterations to reach an ideal configuration.

Bayesian Optimization

[SIGMOD'21] **ResTune**: Resource Oriented Tuning Boosted by MetaLearning for Cloud Databases

[VLDB'21] **CGPTuner**: A Contextual Gaussian Process Bandit Approach for the Automatic Tuning of IT Configurations under Varying Workload Conditions

[JMLR'22] **SMAC3**: A Versatile Bayesian Optimization Package for Hyperparameter Optimization

Reinforcement Learning

[SIGMOD'22] **HUNTER**: An Online Cloud Database Hybrid Tuning System for Personalized Requirements

[TODS'21] **SkinnerDB**: Regret-bounded Query Evaluation via Reinforcement Learning

Problem

Increasing number of knobs



fixed subset of knobs or execute workloads numerous times
to identify important knobs

Each knob could be in continuous
or categorical values



use the default value ranges

Knob	shared_buffers	random_page_cost
Default Range	[0.125MB, 8192 GB]	[0, 1.79769×10^{308}]
Guidance	"shared_buffers" can be 25% of the RAM but no more than 40% ... [39]	"random_page_cost" can be 1.x if disk has a speed similar to SSDs ... [41]
DBA	The machine has a 16 GB RAM. Thus we can set "shared_buffers" from 16 GB \times 25% = 4 GB to 16 GB \times 40% = 6.4 GB.	The machine uses SSDs as disks. Thus we can set "random_page_cost" to a value from 1.0 to 2.0.
Improved Range	[4 GB, 6.4 GB]	[1.0, 2.0]

Human DBAs often rely on domain knowledge for tuning, which directly reveals tuning hints!

Those hints from domain
knowledge are valuable in reducing
search space!

Problem



Domain knowledge? seems exclusive to DBAs?

Due to the heterogeneous nature of domain knowledge, it is difficult to apply domain knowledge without manually.

Some approaches utilize the **static rules** summarized by DBAs to tune DBMS.

All workloads
✗

The updates of environments
✗

Problem



Large Language Model (LLM) have impressive in-context learning abilities.

Can we use **pre-trained LLM** to leverage the domain knowledge?

However, this process is **challenging**!

Challenge 1:

How to unify a structured view of the domain knowledge?

Challenge 2:

How to use the structured domain knowledge?


Problem


Challenge 1:


How to unify a structured view of the domain knowledge?

(1) Domain knowledge typically comes in the form of DBMS documents and discussions from forums

It involves a **complex and lengthy workflow** to process such heterogeneous and noisy knowledge:
data ingestion, data cleaning, data integration and data extraction

 **Hacker News**
Shared_buffers and *work_mem* are crucial for most workloads. Write-heavy OLTP benefits from WAL (eg. *max_wal_size*) and autovacuum adjustments. Read-only OLAP workloads see gains from Postgres' parallel settings (*max_parallel_workers_per_gather*).

 **POSTGRESQLCO.NF**
Knob: random_page_cost
Set it to 1.x (e.g. "1.2") if your disk technology has a random access profile similar to that of SSDs.

 **PostgreSQL Manual**
Knob: effective_io_concurrency
This setting controls concurrent disk I/O operations in PostgreSQL. Raising it increases parallel I/O requests per session. *Valid range: 1 to 1000 or zero to disable asynchronous I/O.*



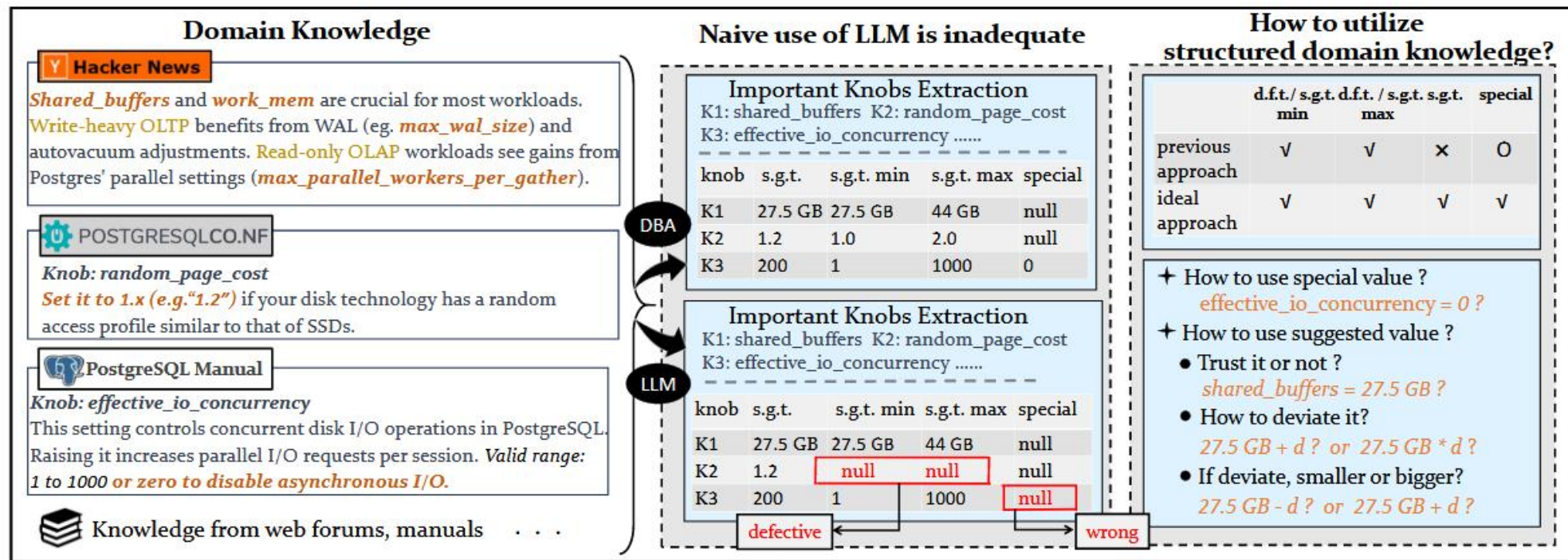
Knowledge from web forums, manuals . . .

Problem

Challenge 1:

How to unify a structured view of the domain knowledge?

(2) The brittle nature and the hallucination problem of LLM



* "s.g.t." means "suggested" and "d.f.t." means "default", "o" means some approach handles this value manually

Problem

Challenge 2: How to use the structured domain knowledge?

DB-BERT

CAN read manuals and use the mined hints

Exhibits rapid convergence

sub-optimal performance and inadequate exploration
of search space

Criterion	DB-BERT	GPTuner
Language Model	BERT	GPT-4
Workload-Aware Knob Selection	no	yes
Fine-Tuning	yes	no
Filter Noise	no	yes
Space Type	Discrete	Heterogeneous
Optimization Algorithm	Reinforcement Learning	Coarse-to-Fine BO
Considered Knowledge	Suggested Value	Suggested Value Bound Constraint Special Cases

Problem



Design a knob-tuning system leveraging domain knowledge

C1

How to unify a structured view of the heterogeneous domain knowledge while balancing a trade-off between cost and quality.



M1 a Large Language Model (LLM)-based pipeline to collect and refine heterogeneous knowledge

M2 a prompt ensemble algorithm to unify a structured view of the refined knowledge

C2

Even with the prepared structured knowledge, how to integrate the knowledge into the optimization process.



M3 a workload-aware and training-free knob selection strategy

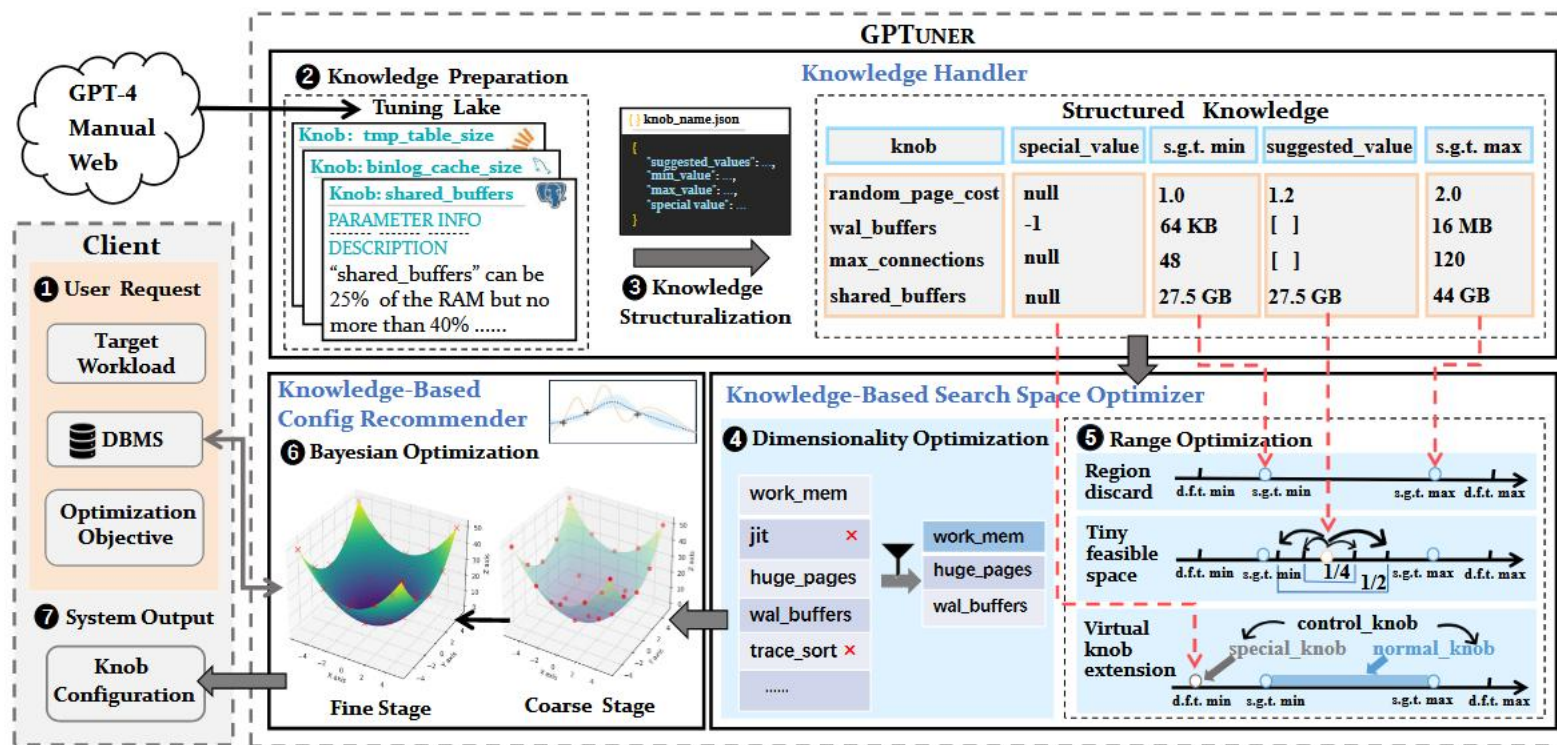
M4 a search space optimization technique considering the value range of each knob

M5 a novel knowledge-based optimization framework to explore the optimized space



- Background
- Problem
- **Design**
- Evaluation
- Conclusion

Key Design

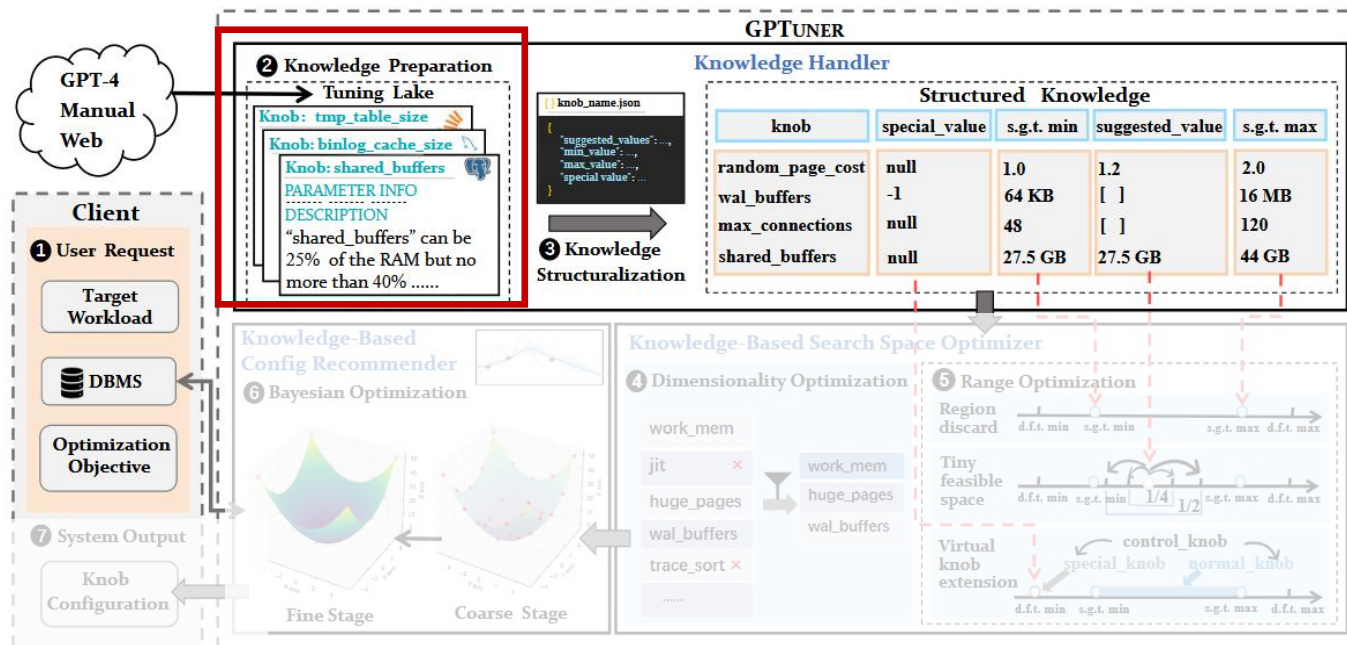


Knowledge Handler

Knowledge-Based Search Space Optimizer

Knowledge-Based Configuration Recommender

Knowledge Handler



Tuning Lake

$$\mathcal{L} = \{d_1, d_2, \dots, d_n\}$$

A set of n texts containing natural language tuning knowledge

Knowledge Handler



Algorithm 1: Knowledge Preparation Algorithm

Input: DBMS manuals \mathcal{D} ; LLM \mathcal{F} ; Rule \mathcal{R} ; Knob Set \mathcal{K} .

Output: Tuning Lake \mathcal{L} .

```
1 Collect tuning knowledge  $\mathcal{D}_{web}$  via a Web Crawler;
2 Extract tuning knowledge  $\mathcal{D}_{LLM}$  from  $\mathcal{F}$ ;
3  $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_{web} \cup \mathcal{D}_{LLM}$ ;
4 Apply  $\mathcal{R}$  and  $\mathcal{F}$  on  $\mathcal{D}$  to obtain legal guidance  $\mathcal{D}_{legal}$ ;
5 Apply  $\mathcal{F}$  on  $\mathcal{D}_{legal}$  to summarize guidance  $d_i$  for  $k_i \in \mathcal{K}$ ;
6 Candidate Tuning Lake  $\mathcal{L} = \{d_i\}$ ;
7 while  $\mathcal{L}$  does not pass Factual Consistency Check by  $\mathcal{F}$  do
8   | Apply  $\mathcal{F}$  to modify  $\mathcal{L}$  based on feedback from check;
9   | Apply  $\mathcal{F}$  on modified  $\mathcal{L}$  to conduct another check;
10 end
11 return  $\mathcal{L}$ ;
```

Step 1 (lines 1-3)
Step 2 (line 4)
Step 3 (lines 5-6)
Step 4 (lines 7-9)

Knowledge Preparation

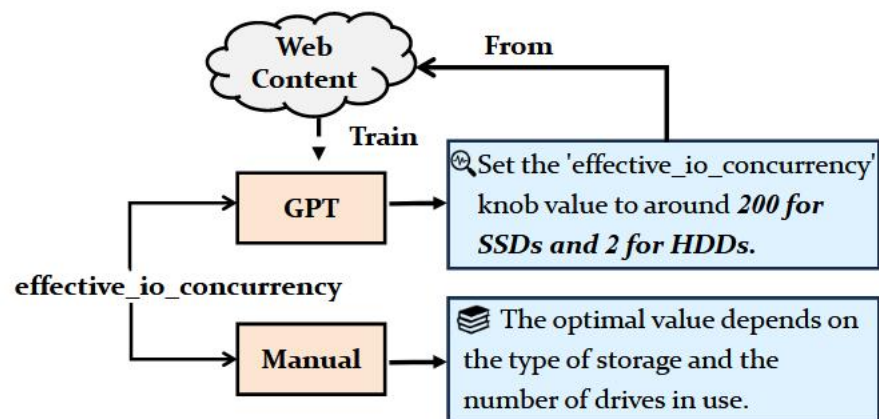
Step 1: Extracting knowledge from LLM

Step 2: Filtering noisy knowledge

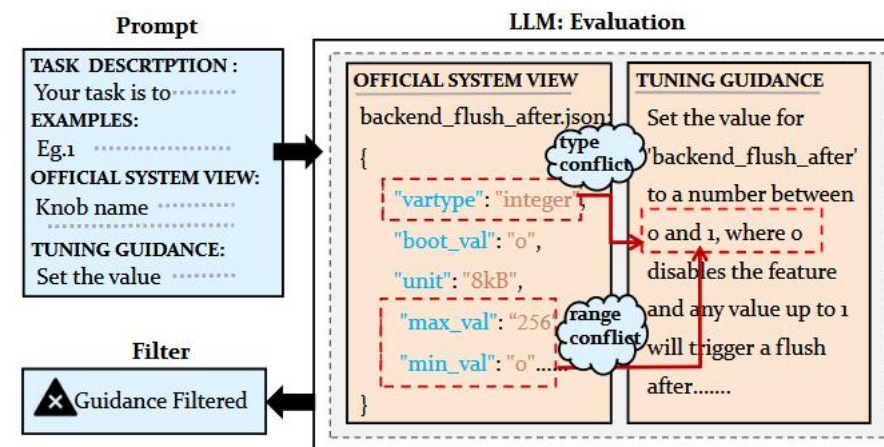
Step 3: Summarizing knowledge from various resources

Step 4: Checking factual inconsistency

Knowledge Handler

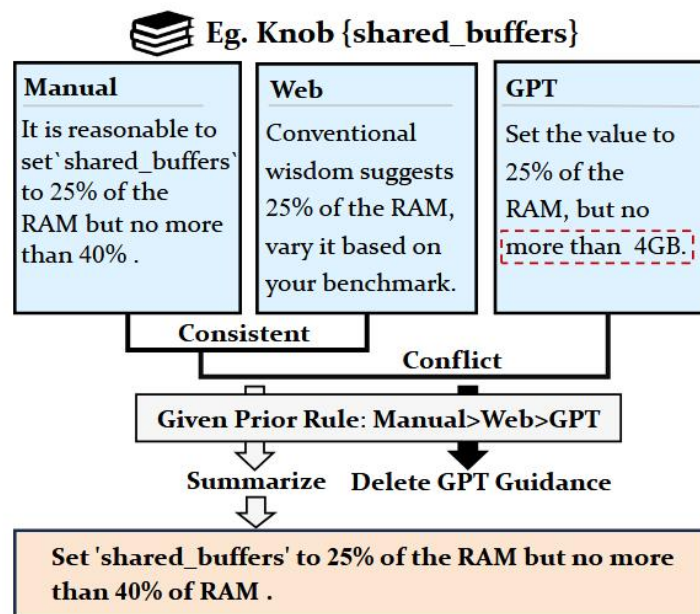


Step 1: Extracting knowledge from LLM

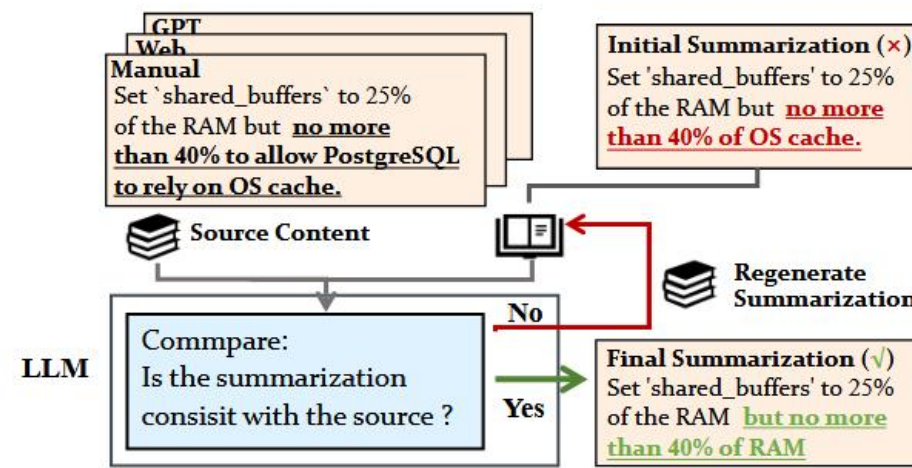


Step 2: Filtering noisy knowledge

Knowledge Handler

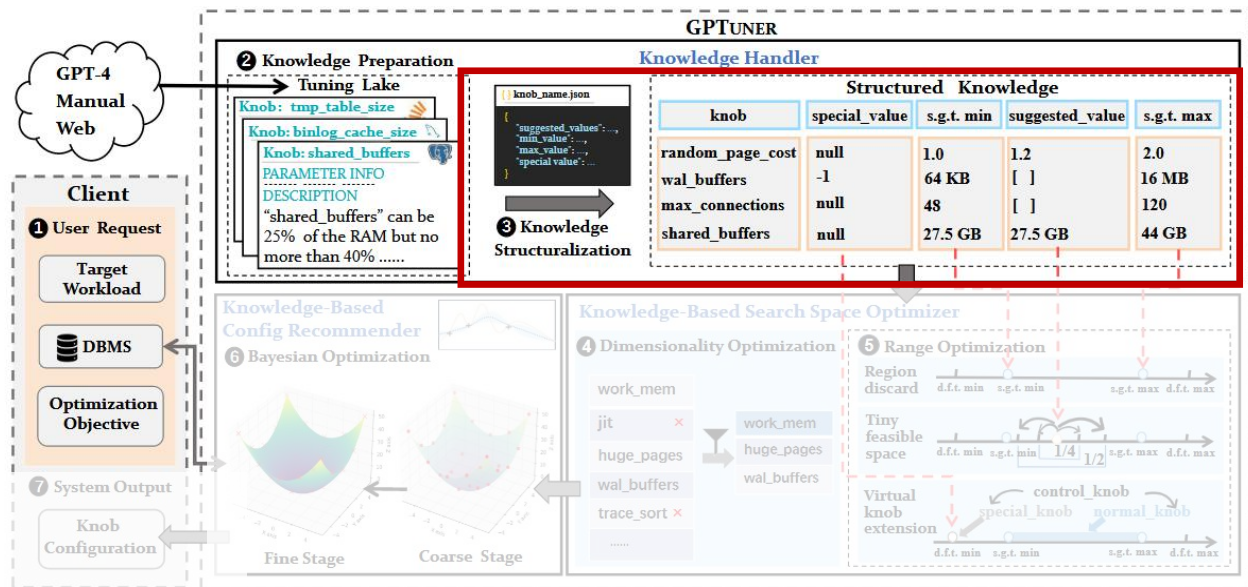


Step 3: Summarizing knowledge from various resources



Step 4: Checking factual inconsistency

Knowledge Handler



Knowledge Transformation

Structured Knowledge

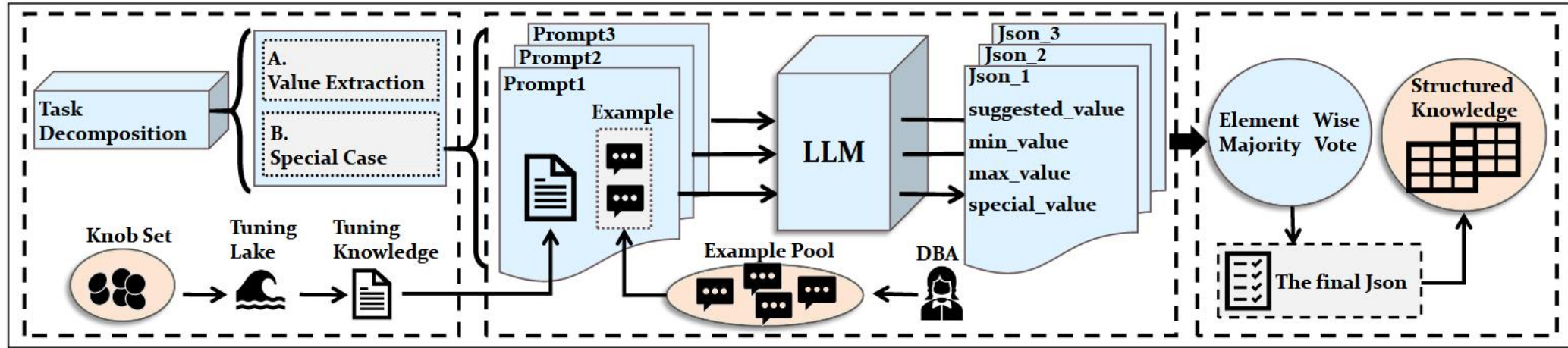
Structured Knowledge \mathcal{S} maintains a structured view s_i for each tuning knowledge d_i from Tuning Lake \mathcal{L} .

s_i is defined by attributes $A = \{a_1, a_2, \dots, a_n\}$ and corresponding values $V = \{v_1, v_2, \dots, v_n\}$

Determining the Attributes

Four types of attributes: `suggested_values`, `min_value`, `max_value` and `special_value`.

Knowledge Handler



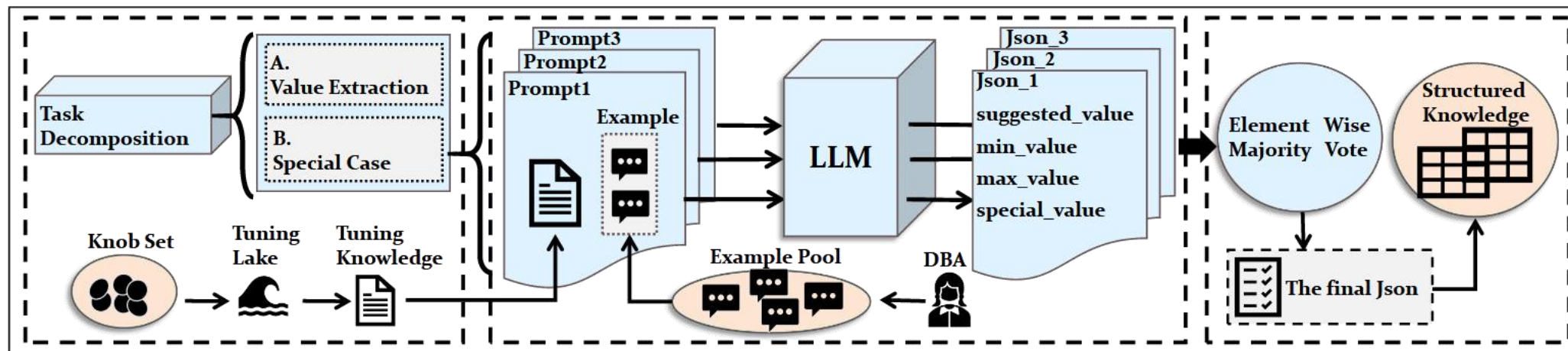
Determining the Attribute Values

Step 1: model the transformation task as a series of information extraction problems.

Step 2: vary the prompts by changing the examples provided for few-shots learning.

Step 3: aggregate the results via a majority vote strategy.

Knowledge Handler



Step 1: model the transformation task as a series of information extraction problems.

Step 2: vary the prompts by changing the examples provided for few-shots learning.

TASK DESCRIPTION :

Extract $\{target\ values\}$ from the given tuning knowledge.

<STEP>

Step 1: Check whether it is useful.

Step 2: Check whether it uses hardware info.

...

Step 7: Return JSON result.

</STEP>

<EXAMPLES>

Sample examples from the Example Pool.

</EXAMPLES>

<QUESTION>

KNOB: $\{knob\}$

JSON RESULT TEMPLATE:

```
{
  "target_values": []
}
```

</QUESTION>

Let's think step by step and give me the final JSON result.



Knob: *random_page_cost*
Set it to 1.x (e.g. "1.2") if your disk technology has a random access profile similar to that of SSDs.

TUNING KNOWLEDGE:

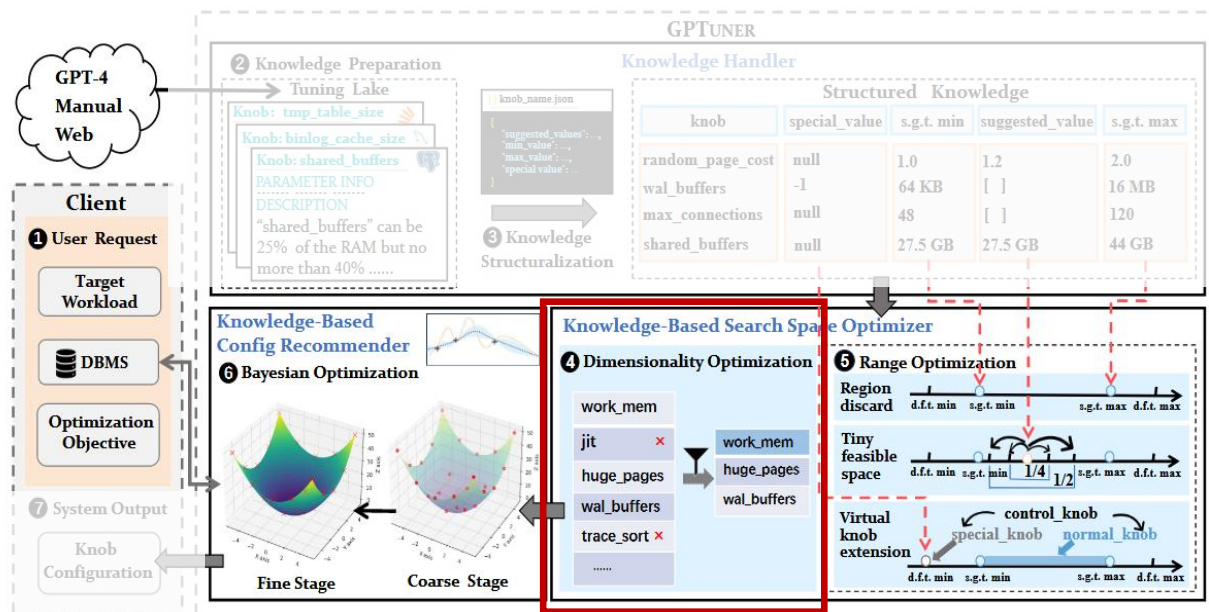
$\{knowledge\}$ ←

HARDWARE INFO:

$\{hardware_info\}$

Step 3: aggregate the results via a majority vote strategy.

Search Space Optimizer



Dimensionality Optimization

- (1) System-Level selects knobs based on the specific DBMS product.
- (2) Workload-Level selects knobs based on the workload type.
- (3) Query-Level selects knobs based on the bottleneck of queries.
- (4) Knob-Level complements interdependent knobs to a given target knob set.

Search Space Optimizer



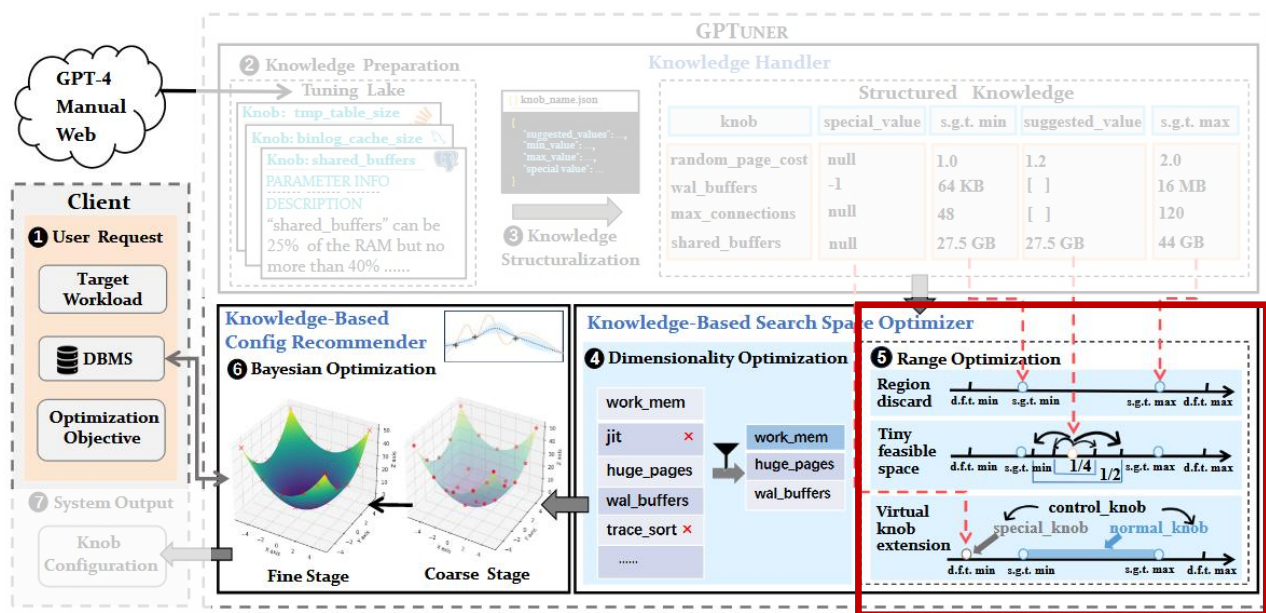
Algorithm 1: LLM-based Knob Selection

Input: Knob Set \mathcal{K} ; LLM \mathcal{F} ; DBMS \mathcal{D} ; Workload \mathcal{W} ;
Tuning Lake \mathcal{L} .

Output: Target Knob Set \mathcal{T} .

```
1 Configurable Knob Set  $C \leftarrow \text{FILTER}(\mathcal{K})$ ;  
  // Filter out knobs that are related to  
  // debugging, security and path-setting  
  System Level Selection:  
2  $C_s \leftarrow \mathcal{F}(C, \mathcal{D})$ ;  
  Workload Level Selection:  
3  $C_w \leftarrow \mathcal{F}(C, \mathcal{W})$ ;  
  Query Level Selection:  
4  $C_q \leftarrow \emptyset$ ;  
5 for query  $q_i$  in  $\mathcal{W}$  do  
6    $\mathcal{E}_i \leftarrow \text{EXECUTE}(\mathcal{D}, q_i)$ ;  
   // Get execution plan for query  $q_i$  from  $\mathcal{D}$   
7    $C_{q_i} \leftarrow \mathcal{F}(C, \mathcal{E})$ ;  
8    $C_q \leftarrow C_q \cup C_{q_i}$ ;  
9 end  
  Knob Level Selection:  
10 Target Knob Set  $\mathcal{T} \leftarrow \mathcal{F}(\mathcal{L}, C_s \cup C_w \cup C_q)$ ;  
11 return  $\mathcal{T}$ ;
```

Search Space Optimizer



Tiny Feasible Space

$$\Omega(k) = \{\alpha \cdot V | \alpha \in M\}$$

$$\alpha = 1 + \frac{\beta}{V} (U - V), \beta \in \{r_1, r_2, \dots, r_n | r_i \in [0, 1]\}$$

Range Optimization

Region Discard

Knob	shared_buffers	random_page_cost
Default Range	[0.125MB, 8192 GB]	[0, 1.79769 × 10 ³⁰⁸]
Guidance	"shared_buffers" can be 25% of the RAM but no more than 40% ... [39]	"random_page_cost" can be 1.x if disk has a speed similar to SSDs ... [41]
DBA	The machine has a 16 GB RAM. Thus we can set "shared_buffers" from 16 GB × 25% = 4 GB to 16 GB × 40% = 6.4 GB.	The machine uses SSDs as disks. Thus we can set "random_page_cost" to a value from 1.0 to 2.0.
Improved Range	[4 GB, 6.4 GB]	[1.0, 2.0]

- (1) The regions are unlikely to result in promising performance
- (2) The regions could seize too many system resources
- (3) The regions that can make the DBMS crash

Search Space Optimizer



Virtual Knob Extension

Algorithm 2: Virtual Knob Extension

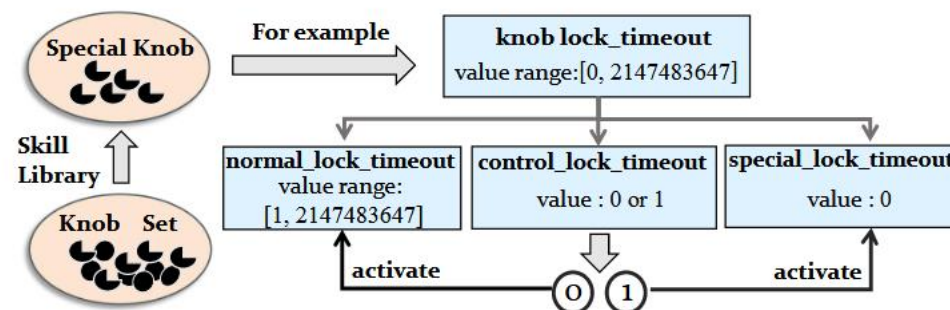
Input: Knob Set \mathcal{K} ; Skill Library \mathcal{S} ; Optimizer O .

Virtual Knobs Generation:

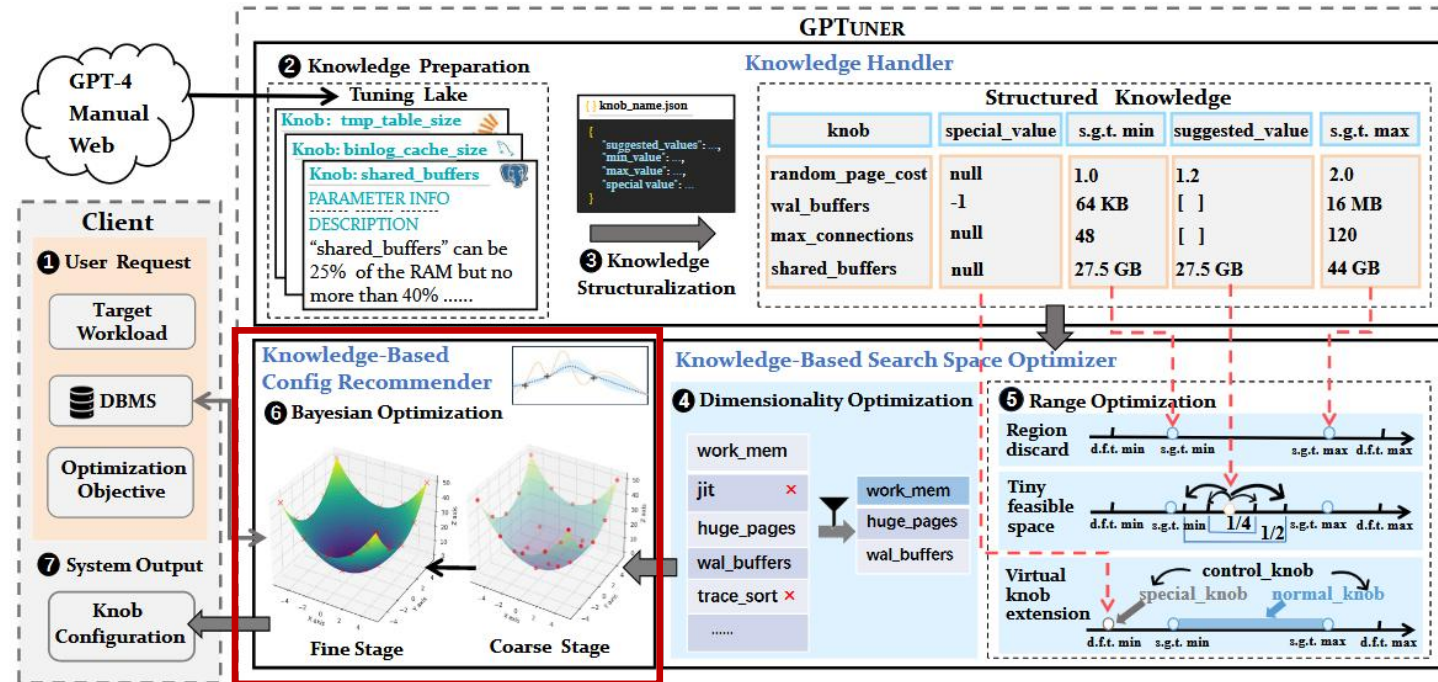
```
1 Use  $\mathcal{S}$  to identify special knobs  $\mathcal{R} \subset \mathcal{K}$ ;  
2 for  $r \in \mathcal{R}$  do  
3   Create virtual knobs  
   {control_knob, normal_knob, special_knob} for  $r$ ;  
4 end
```

Virtual Knobs Utilization:

```
5 for round = 1 to number of iterations do  
6    $O$  chooses a value from  $\{0, 1\}$  for control_knob;  
7   if control_knob is 0 then  
8     Activate normal_knob in  $O$ ;  
9   else  
10    Activate special_knob in  $O$ ;  
11  end  
12 end
```



Knowledge-Based Configuration Recommender



Configuration Recommender

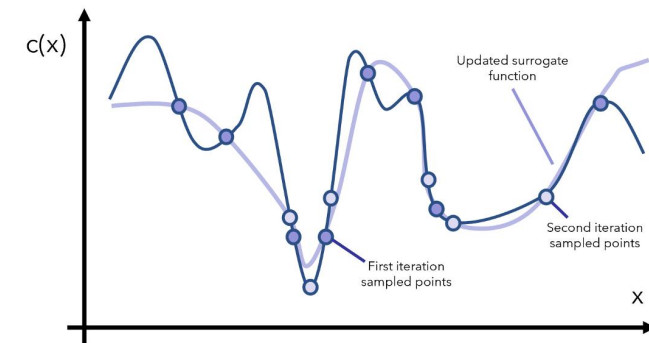
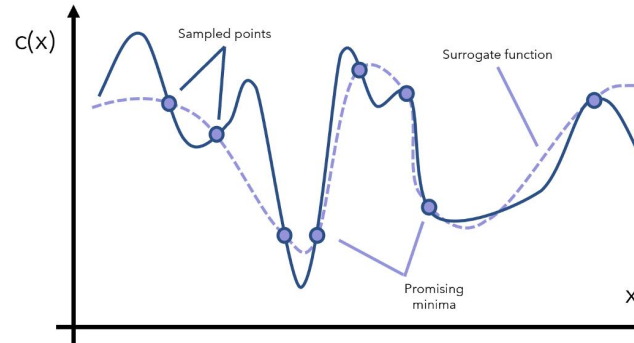
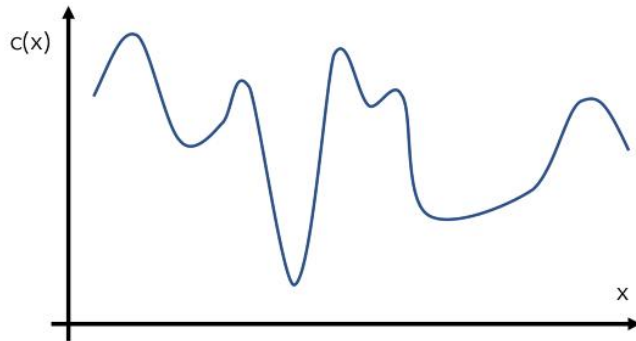
Knowledge-Based Configuration Recommender

Bayesian Optimization

sequential model-based algorithm

Surrogate model

Acquisition function



Knowledge-Based Configuration Recommender

Limitation of Bayesian Optimization

Resource-intensive and Time-consuming



Integrate the domain knowledge into the optimization process

Coarse-to-Fine Bayesian Optimization Framework

Algorithm 3: Coarse-to-Fine BO Framework. Blue underlined text highlights differences to original BO algorithm.

Input: DBMS \mathcal{D} ; Surrogate Model \mathcal{M} ; Acquisition Function \mathcal{A} ; Workload \mathcal{W} ; Structured Knowledge \mathcal{S} ; Whole Space \mathcal{P} ; Coarse Threshold C ; Initial Number n .

Output: Knob Configuration \mathcal{X} .

```
1 Generate Tiny Feasible Space  $\mathcal{T}$  from  $\mathcal{S}$ ;
2 Generate  $n$  samples  $p_i \in \mathcal{T}$  with space-filling design (LHS);
3 Evaluate samples on  $\mathcal{D}$  to get performance  $y_i$ ;
4 Update  $\mathcal{M}$  with observations  $\{(p_i, y_i)\}$ ;
   Coarse Stage:
5 for  $i = 1$  to  $C$  do
6    $\vec{x}_i \leftarrow \arg \max_{\vec{x} \in \mathcal{T}} \mathcal{A}(\vec{x})$ ;
7   Evaluate  $\vec{x}_i$  on  $\mathcal{D}$  to get performance  $y_i$ ;
8   Update  $\mathcal{M}$  with  $(\vec{x}_i, y_i)$ ;
9 end
   Fine Stage:
10 Reuse the surrogate model  $\mathcal{M}$  from Coarse Stage;
11 Apply Region Discard on  $\mathcal{P}$  to get  $\mathcal{P}'$ ;
12 Apply Virtual Knob Extension on  $\mathcal{P}'$  to get  $\mathcal{P}''$ ;
13 while not stopping condition do
14    $\vec{x}_i \leftarrow \arg \max_{\vec{x} \in \mathcal{P}''} \mathcal{A}(\vec{x})$ ;
15   Evaluate  $\vec{x}_i$  on  $\mathcal{D}$  to get performance  $y_i$ ;
16   Update  $\mathcal{M}$  with  $(\vec{x}_i, y_i)$ ;
17 end
18  $\mathcal{X} \leftarrow \arg \max_{\vec{x}_i} y_i$ ;
19 return  $\mathcal{X}$ ;
```



- Background
- Problem
- Design
- **Evaluation**
- Conclusion

Experimental Setup



Workloads

TPC-H (OLAP type) with scale factor 1 and TPC-C (OLTP type) with factor 200 as our benchmarks

Hardware

cloud server with a 24-core Intel Xeon E5-2676 v3 CPU, 110 GB of RAM and a 931 GB WD SSD.

GeForce RTX 2080 Ti with 22 GB of memory.

Baselines

SMAC GP DB-BERT DDPG++

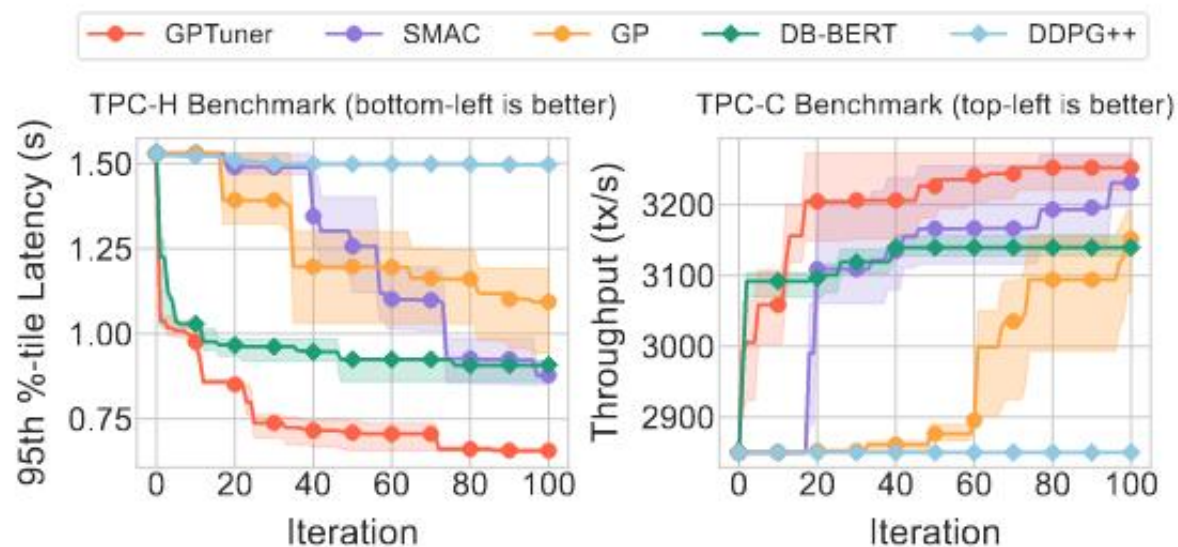
optimize **throughput** for **OLTP** workload
and **95-th percentile latency** for **OLAP**
workload

Tuning Settings

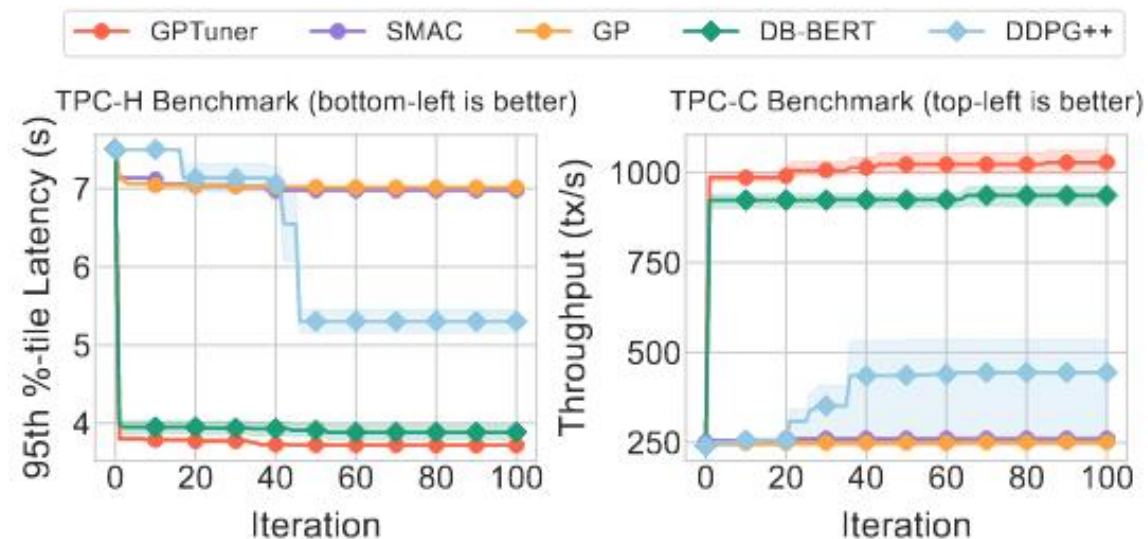
PostgreSQL v14.9 and MySQL v8.0

tune the same 60 and 40 knobs of PostgreSQL and MySQL

Performance Comparison

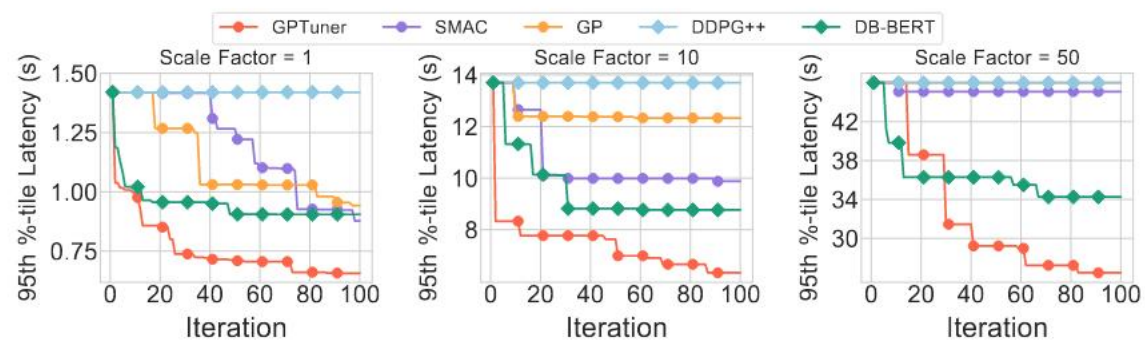


Optimizing for PostgreSQL

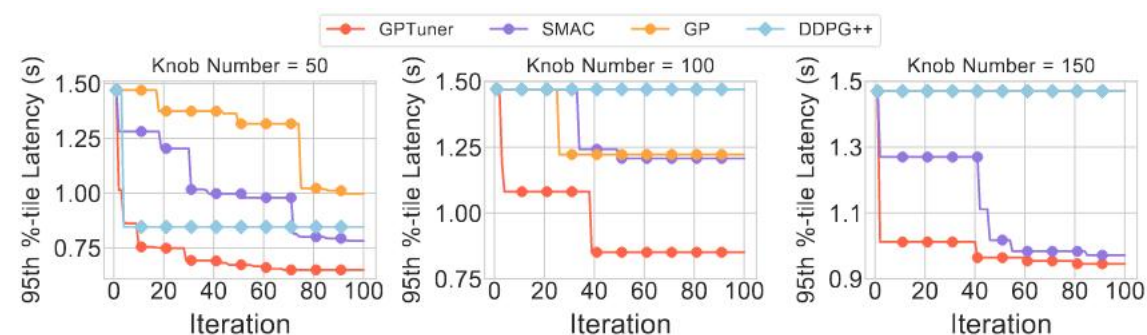


Optimizing for MySQL

Scalability Study



Database Size



Search Space Dimensionality

Robustness Study

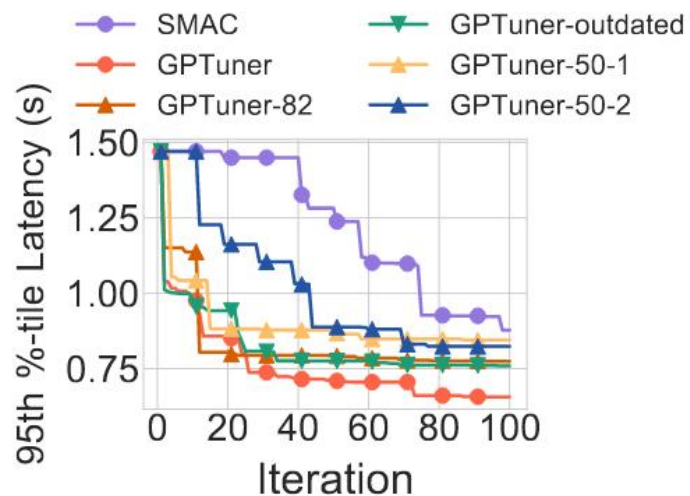


Effect of Error Correction Mechanisms

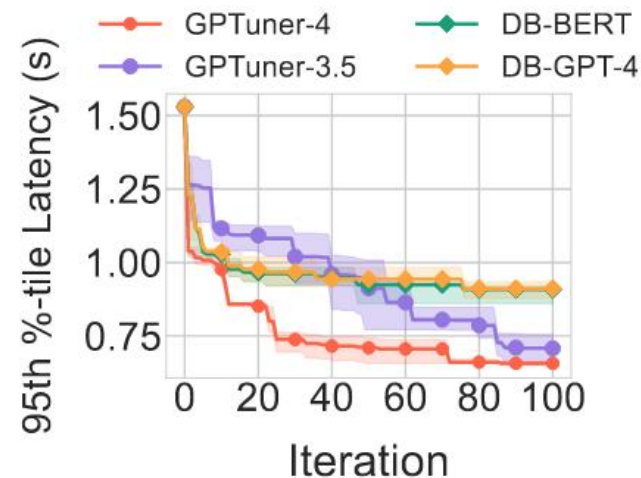
Equipped with step 2 as a filter of noisy knowledge, GPTuner improve the accuracy of input knowledge to 94%

GPT-4 successfully detects 78% of the factual inconsistent, and rewrites all of them correctly with an accuracy of 100%, which means it is feasible to utilize GPT-4 as a corrector

Effect of Domain Knowledge Quality



Effect of Different Language Models





(1) Initial Profiling Overhead

(2) Runtime Overhead

**NO extra runtime overhead compared
to the BO-based optimizer**

Table 3: Initial Profiling Overheads Statistics

Complex.	Knob #n	Tuning Lake									Structured Knowledge								
		Token (k)			Money (USD)			Time (s)			Token (k)			Money (USD)			Time (s)		
		3.5	4	4-turbo	3.5	4	4-turbo	3.5	4	4-turbo	3.5	4	4-turbo	3.5	4	4-turbo	3.5	4	4-turbo
O (n)	50	124.4	160.1	148.8	0.2	6.0	2.1	833.0	4155.0	3023.0	558.0	578.0	558.4	0.9	18.7	5.9	1183.0	4888.0	1805.0
	100	254.6	324.1	300.4	0.4	12.2	4.2	1679.0	7939.0	6021.0	1121.0	1162.7	1105.5	1.7	37.8	11.7	2371.0	9738.0	4985.0
	150	379.5	482.7	443.9	0.6	18.2	6.2	2450.0	11742.0	8626.0	1690.9	1738.4	1657.7	2.6	56.4	17.5	3574.0	13958.0	6729.0
	average	2.5	3.2	3.0	0.003	0.1	0.04	16.3	78.3	57.5	11.3	11.6	11.1	0.02	0.4	0.1	23.8	93.1	44.9



- Background
- Problem
- Design
- Evaluation
- **Conclusion**



GPTuner, an automatic manual-reading database tuning system

- Leveraging LLMs to Integrate Domain Knowledge
- Multi-Level Search Space Optimization
- Coarse-to-Fine Bayesian Optimization Framework
- Significant Experimental Performance Improvement



➤ 这个paper有什么问题，基于这个paper还能做什么？

- 文章中只针对MySQL和PostgreSQL两个数据库进行了实验，对于其他的数据库系统的效果如何？

比如如果是图数据库的话，能否沿用GPTuner的思路，使用LLM整合领域知识？
如果是更复杂的多模型数据库呢？

- GPTuner是在离线状态根据一个DBMS进行调优之后在部署到实际运行环境中的，那么有没有可能基于此实现在线调优呢？



➤ 这个paper提到的idea能不能用在自己的方向/project上面?

(1) 知识转化过程很有参考价值，其中的数据去噪、结合这些流程还有对于特殊值、建议值的处理可以参考，此外使用LLM的生成作为领域知识感觉也是一个可以添加的点

(2) 参数调优这一块，GPTuner因为是数据库调优所以选择一种基于BO的空间搜索的算法，我的项目中在调优这一块是否也可以设计成搜索算法来找到最优的配置?

➤ 这个paper能不能泛化?

- 利用LLM结合领域知识进行优化这一思想可以应用比如资源调度管理这一问题上，使用LLM指导调度算法参数的配置
- 动态参数选择以及范围裁剪特殊值处理的思想也可以有广泛的应用，通过使用范围裁剪和特殊值处理降低最优配置的搜索空间

The background of the slide is a faded, light-colored image of a large, classical-style building with a prominent dome and a portico supported by columns. In the foreground, to the left, is a circular fountain with water spraying upwards. The overall tone is soft and professional.

Thank you!