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

Jiale Lao[1], Yibo Wang[1], Yufei Li[1], Jianping Wang[2], Yunjia Zhang[3], Zhiyuan Cheng[4], Wanghu Chen[2], Mingjie Tang[1], Jianguo Wang[4]
[1]Sichuan University, {laojiale, wangyibo2, evangeline}@stu.scu.edu.cn, tangrock@gmail.com
[2]Northwest Normal University, {2022222119, chenwh}@nwnu.edu.cn
[3]University of Wisconsin-Madison, yunjia@cs.wisc.edu
[4]Purdue University, {cheng443, csjgwang}@purdue.edu

ABSTRACT

Modern database management systems (DBMS) expose hundreds of configurable knobs to control system behaviours. Determining the appropriate values for these knobs to improve DBMS performance is a long-standing problem in the database community. As there is an increasing number of knobs to tune and each knob could be in continuous or categorical values, manual tuning becomes impractical. Recently, automatic tuning systems using machine learning methods have shown great potentials. However, existing approaches still incur significant tuning costs or only yield sub-optimal performance. This is because they either ignore the extensive domain knowledge available (e.g., DBMS manuals and forum discussions) and only rely on the runtime feedback of benchmark evaluations to guide the optimization, or they utilize the domain knowledge in a limited way. Hence, we propose GPTuner, a manual-reading database tuning system that leverages domain knowledge extensively and automatically to optimize search space and enhance the runtime feedback-based optimization process. Firstly, we develop a Large Language Model (LLM)-based pipeline to collect and refine heterogeneous knowledge, and propose a prompt ensemble algorithm to unify a structured view of the refined knowledge. Secondly, using the structured knowledge, we (1) design a workload-aware and training-free knob selection strategy, (2) develop a search space optimization technique considering the value range of each knob, and (3) propose a Coarse-to-Fine Bayesian Optimization Framework to explore the optimized space. Finally, we evaluate GPTuner under different benchmarks (TPC-C and TPC-H), metrics (throughput and latency) as well as DBMS (PostgreSQL and MySQL). Compared to the state-of-the-art approaches, GPTuner identifies better configurations in **16x** less time on average. Moreover, GPTuner achieves up to **30%** performance improvement (higher throughput or lower latency) over the **best-performing** alternative.

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/SolidLao/GPTuner.

1 INTRODUCTION

Modern Database Management Systems (DBMS) expose hundreds of configurable parameters (i.e., knobs) to control their runtime behaviours [38]. The selection of appropriate values for these knobs is crucial to improve DBMS performance, constituting a long-standing challenge within the database community [53]. Given the high dimensionality of the configuration space (e.g., PostgreSQL v14.7 has 351 knobs), and the inherent heterogeneity of these knobs (due to their continuous and categorical domains), database administrators (DBAs) encounter substantial difficulties in identifying promising configurations tailored to specific query workloads. The magnitude of this challenge becomes even more pronounced in the cloud environment, where the underlying physical configurations can significantly vary among distinct DBMS instances [39].

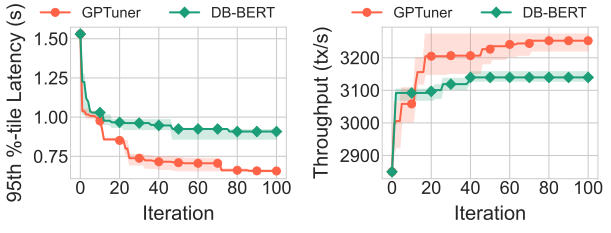
To reduce the manual tuning efforts of DBAs, state-of-the-art approaches automate the knob tuning via Machine Learning (ML) techniques, including Bayesian Optimization [7, 14, 24, 52, 61, 62] and Reinforcement Learning [6, 26, 50, 59]. These ML-based tuning systems follow the main concept of "trial and error" to explore the configuration space iteratively, balancing between the exploration of unseen regions and the exploitation of known space.

While these tuning systems do possess the potential to reach well-performing knob configurations eventually, they still incur significant tuning costs [23, 24]. For example, previous studies [6, 60, 61] have revealed that state-of-the-art systems still require hundreds to thousands iterations to reach an ideal configuration, with each iteration taking several minutes or more to execute the target workload. Such high tuning costs stem from their inefficiency in handling two difficulties: (1) *the large number of knobs that requires tuning* and (2) *the wide search space of possible values for each knob*. For the first difficulty, most approaches either select a fixed subset of knobs [6, 7, 14, 23, 59, 67], sacrificing the flexibility to choose workload-relevant knobs, or execute workloads numerous times to identify the important knobs [22, 52, 60], which is resource-intensive. Regarding the second difficulty, most approaches use the default value ranges provided by DBMS vendors [6, 7, 14, 24, 26, 52, 59, 61, 62]. However, since the default value ranges are excessively broad for flexibility, it complicates the optimization process and introduces the risk of system crashes [39, 53].

In contrast to ML-based tuning approaches that adjust the DBMS solely based on performance statistics, human DBAs often rely on domain knowledge for tuning (e.g., DBMS manuals and discussions from DBMS forums). Unlike performance statistics, external domain knowledge directly reveals tuning hints, including the important knobs and typical value ranges for each knob. First, there are discussions on which knobs significantly impact DBMS performance. For instance, in web forums like *Hacker News*, it is commonly mentioned that parallel knobs (e.g., "`max_parallel_workers_per_gather`") are crucial for OLAP workloads, while I/O-related knobs (e.g., "`max_wal_size`") are important for OLTP workloads [18]. Second, there are typical value ranges summarized for knobs. Table 1 provides two examples of extracting improved value ranges from natural language tuning guidance. For knob "`shared_buffers`", instead

Table 1: Tuning Knowledge Utilization

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% ...[40]	"`random_page_cost`" can be 1.x if disk has a speed similar to SSDs ...[42]
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]



(a) TPC-H Benchmark **(b) TPC-C Benchmark**

Figure 1: GPTuner vs. DB-BERT

of using the default value range [0.125 MB, 8192 GB], the improved value range can be [4 GB, 6.4 GB] since the guidance suggests setting the value between 25% and 40% of the RAM (on a machine with 16 GB of RAM). For knob "`random_page_cost`", we can try out value range [1.0, 2.0] rather than [0, 1.79769×10^{308}] if the machine uses SSDs as disks. We can observe that these hints are highly valuable in reducing the search space of ML-based methods and thus expedite convergence and achieve better performance.

Therefore, to mitigate the high tuning costs of ML-based techniques, it is desirable to design a knob-tuning system leveraging domain knowledge to enhance the optimization process. However, this is non-trivial for the following challenges. *C1. It is challenging to unify a structured view of the heterogeneous domain knowledge while balancing a trade-off between cost and quality.* Domain knowledge typically comes in the form of DBMS manuals and discussions on the web forums. To leverage such knowledge in the ML-based techniques, we need to transfer it into a unified machine-readable format (i.e., structured data). However, preparing such a structured view involves a complex and lengthy workflow: data ingestion, data cleaning, data integration and data extraction [36], and existing approaches cannot meet our trade-off between cost and quality. They either demand domain specific training [45], which is more complicated in our scenario since it requires expert knowledge to annotate DBMS tuning knowledge, or they rely on strong assumptions that lack flexibility (e.g., focusing on specific document format) [11, 30, 31]. *C2. Even with the prepared structured knowledge, we lack a way to integrate the knowledge into the optimization process.* The inherent design of optimization algorithms like Bayesian Optimization [46] and Reinforcement Learning [20] does not support the integration of external domain knowledge directly, necessitating extensive modifications to their standard workflows. For approaches that manually summarize static rules from domain knowledge, the resulting rules cannot capture the nuances of all workloads, and the updates of environments can make them out of date [9, 25]. To address the challenges above, we propose GPTuner, a manual-reading database tuning system that leverages domain knowledge automatically and extensively to enhance the optimization process.

Facing the challenge C1, we develop a Large Language Model (LLM)-based pipeline to collect and refine heterogeneous knowledge, and propose a prompt ensemble algorithm to unify a structured view of the refined knowledge. In light of the brittle nature of LLM and its inevitable hallucination problem (see Section 3 for more details), we design an LLM-based pipeline incorporating two error correction mechanisms (i.e., step 2 is a filter of step 1 and step 4 is both an evaluator and a rewriter of step 3). First, we prepare heterogeneous tuning knowledge from various resources (data ingestion). Second, we filter out noisy knowledge (data cleaning). Third, we summarize the multi-source knowledge by handling the possible conflict in a priority way (data integration). Fourth, we ensure the summarization result is factually consistent with source contents (data correction). Finally, we develop a prompt ensemble algorithm to construct a structured view of the knowledge (data extraction).

Regarding the challenge C2, we use the structured knowledge to (1) design a workload-aware and training-free knob selection strategy, (2) develop a search space optimization technique considering the value range of each knob, and (3) propose a novel knowledge-based optimization framework to explore the optimized space. Before the tuning process, we optimize the search space in terms of the space dimensionality and the values in each dimension. For space dimensionality, we leverage the text analysis ability of LLM to simulate the knob selection process of DBAs, considering the characteristics of DBMS, workload, specific query and knob dependencies. For values in each dimension, we discard meaningless regions, highlight promising space and take special situations into consideration. Next, we develop a novel Coarse-to-Fine Bayesian Optimization Framework. At first, since it is non-trivial to reduce the size of search space while still retaining the potential for optimal results [16], we seek help from domain knowledge to carefully design two search spaces of different granularity. Next, the two spaces are explored sequentially by BO from coarse granularity to fine granularity, with a bootstrap technique to serve as a bridging mechanism. This framework balances between the efficiency of coarse-grained search and the thoroughness of fine-grained search.

We are aware of only one work DB-BERT [50] that utilizes a pre-trained language model to read manuals and uses the mined hints to guide a reinforcement learning algorithm. It exhibits rapid convergence as it benefits from the information gained via text analysis. However, it only yields sub-optimal performance since it utilizes the domain knowledge narrowly and suffers from the inadequate exploration of search space. GPTuner tackles these limitations and thus achieves faster convergence and better performance improvement. As shown in Figure 1, GPTuner significantly outperforms DB-BERT on two representative benchmarks (TPC-H and TPC-C) with different optimization objective (latency and throughput). It is important to note that GPTuner is a distinct approach compared to DB-BERT, and its efficacy does not stem from simply substituting the language model. We verify this by replacing the BERT model in DB-BERT with GPT-4 and compare it with GPTuner again. More details are provided in Section 2.3.

In our experiments, we compare GPTuner against DB-BERT as well as other state-of-the-art approaches that do not use text as input. We consider different benchmarks (TPC-C and TPC-H), metrics (throughput and latency), and DBMS (PostgreSQL and MySQL). Compared to the state-of-the-art approaches, GPTuner

identifies better configurations in **16x** less time on average, achieving up to **30%** performance improvement (higher throughput or lower latency) over the **best-performing** alternative. In addition to performance improvement, we manually prepare, label and open-source two datasets[1], conducting extensive experiments to evaluate GPTuner's robustness and scalability. Moreover, we discuss the overheads introduced in GPTuner and present the experimental statistics using different language models. In summary, we make the following contributions:

- We propose GPTuner, a novel manual-reading database tuning system that leverages domain knowledge automatically and extensively to enhance the knob tuning process.
- We develop an LLM-based pipeline to collect and refine domain knowledge, and propose a prompt ensemble algorithm to unify a structured view of the refined knowledge.
- We design a workload-aware and training-free knob selection strategy, develop an optimization method for the value range of each knob, and propose a Coarse-to-Fine Bayesian Optimization framework to explore the optimized space.
- We open-source the built structured domain knowledge for PostgreSQL and MySQL, two datasets for the evaluation of LLM to enable the further study of broader database community.
- We conduct extensive experiments to demonstrate the effectiveness of GPTuner, considering different benchmarks, metrics and DBMS. Compared with the state-of-the-art methods, GPTuner finds the best-performing knob configuration with significantly less tuning rounds.

2 BACKGROUND AND RELATED WORK

2.1 Database Knob Tuning

DBMS expose tens to hundreds of configurable knobs to control their runtime behaviours [53]. The DBMS knob tuning problem is to select an appropriate value for each knob to optimize the DBMS performance (e.g., throughput or latency) on a certain workload (e.g., a workload is a set of SQL statements). Formally, given a set of configurable knobs $\theta_1, \ldots, \theta_n$ along with their domains $\Theta_1, \ldots, \Theta_n$, the configuration space is defined as $\boldsymbol{\Theta} = \Theta_1 \times \cdots \times \Theta_n$. We want to find a configuration $\boldsymbol{\theta}^* \in \boldsymbol{\Theta}$ to maximize DBMS performance f:

$$\boldsymbol{\theta}^* = \arg\max_{\theta \in \boldsymbol{\Theta}} f(\theta) \tag{1}$$

Finding an optimal solution for this problem is challenging for two main reasons. Firstly, with hundreds of adjustable knobs, each potentially impacting performance, the complexity is immense. Secondly, the knobs themselves vary widely – some have continuous numerical values, while others are categorical, making the problem even more complicated since it is hard to model heterogeneous space. To address these challenges, three kinds of approaches are proposed: heuristic-based, Bayesian Optimization (BO)-based [46] and Reinforcement Learning (RL)-based [20].

• **Heuristic-based.** Heuristic-based methods can be classified into rule-based and search-based. Rule-based methods rely on manually created static rules to explore the space in a predefined way [9, 25]. Search-based methods explore the space according to several heuristics (e.g., avoid to revisit explored regions and explore the nearby regions to improve the current optimum) [4, 67].

• **BO-based.** BO-based methods [7, 14, 24, 52, 61, 62] follow the generic BO framework to search for an optimal configuration: (1) fitting a probabilistic *surrogate model* to map the relation between knob configuration and DBMS performance, (2) selecting the next configuration that maximizes *acquisition function*.

• **RL-based.** RL-based methods explore configuration space with a trial-and-error strategy. The essence is to balance between exploring unexplored space and exploiting known regions, which is achieved by the interactions between an agent (e.g., neural network) and the target environment (e.g., DBMS) [6, 26, 59].

Performance Comparison. An experimental evaluation is conducted to compare these approaches [60]. A BO-based method, Sequential Model-based Algorithm Configuration (SMAC) [28] yields the best performance because it uses Random Forest (RF) as the surrogate to leverage its efficiency in modeling the high-dimensional and heterogeneous configuration space. Since SMAC is evaluated as the best-performing optimizer, *we view it as the current state-of-the-art that does not take text as input and aim to improve it further.*

2.2 Language Models

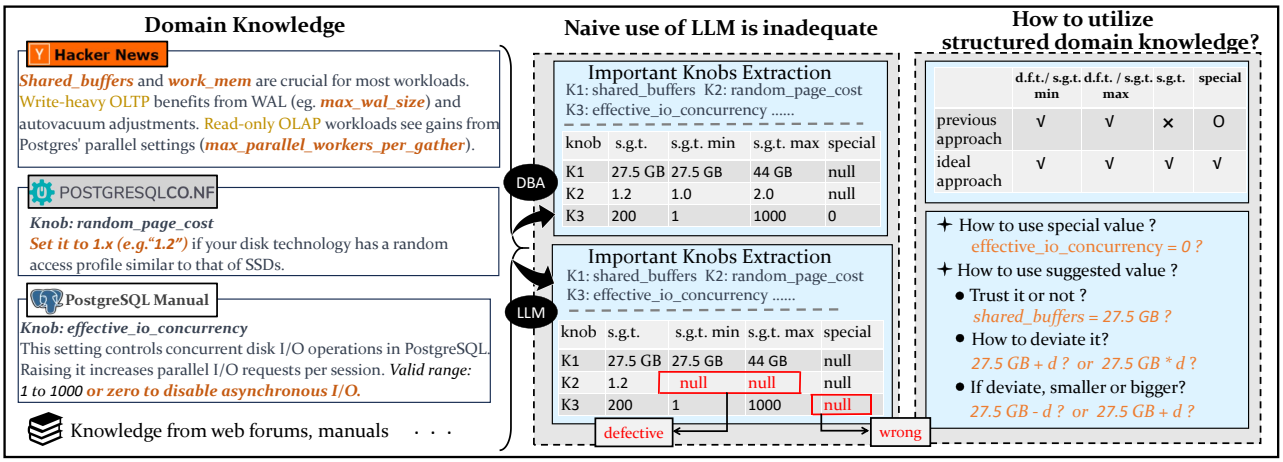
The field of Natural Language Processing (NLP) has undergone a significant transformation with the advent of Large Language Models (LLMs). Starting with the introduction of attention and transformer architecture [54], numerous language models (LMs) have been developed, resulting in a substantial enhancement of language processing capabilities. Notably, BERT, which introduced the encoder-only transformer architecture, revolutionized downstream learning tasks by improving embeddings [12]. More recent additions to the GPT series, such as ChatGPT [32] and GPT-4 [37], have demonstrated greater prowess across a wide range of tasks.

Moreover, the impact of these LLMs extends beyond the scope of NLP. The database community has experienced a surge in the adoption of LLMs to enhance various aspects of DBMS, including data profiling [35, 49, 63], code generation [8, 44, 51, 56] and table-based question answering [21, 55, 64]. When using LLMs in practice, there are three typical design choices: (1) training a language model from scratch, (2) fine-tuning an existing language model, and (3) using a pre-trained language model without parameter modifications. The first two options require a relatively large amount of resources, including both hardware resources and training data. Given the impressive in-context learning capabilities demonstrated by GPT-4, we have chosen the third option and use GPT-4 as our default language model throughout the paper unless noted otherwise.

2.3 Enhancing DBMS Knob Tuning with LMs

With the vast corpus of tuning guidance available on the internet or provided by DBMS vendors, language models can be harnessed to "read the manual" and provide structured tuning hints to enhance the knob tuning approaches. As the state-of-the-art LM-enhanced tuning approach, DB-BERT [50] models the tuning process as a series of "multiple choice question answering problem", and uses Reinforcement Learning to fine-tune a pre-trained BERT model to answer these problems. While DB-BERT converges fast as it benefits from the text analysis, it only yields sub-optimal performance.

[1] https://drive.google.com/file/d/1Ss6EL-B3lhKkwVNBW5vPu-JQ-IeldaUJ



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

Figure 2: Motivating Example

Table 2: Comparison between DB-BERT and GPTuner

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

We summarize the main differences between GPTuner and DB-BERT in Table 2. Firstly, the knob selection of DB-BERT is not workload-aware since it only tunes knobs mentioned in the input documents. GPTuner considers the tuning context (e.g., the characteristic of DBMS, workload, specific query and the knob dependencies) to select which knobs are worth tuning. Secondly, DB-BERT requires fine-tuning of BERT model in both offline training and online tuning stages, which requires a lot of resources. GPTuner adopts in-context learning ability of LLM without modifying the model parameters. Thirdly, DB-BERT relies on the input documents without filtering out noisy contents, while GPTuner tackles this by comparing the contents with the system view from DBMS (e.g., *pg_settings* from PostgreSQL). Fourthly, DB-BERT only relies on the suggested values from documents to construct a discrete space and only explores this limited space. However, such limited utilization of domain knowledge and inadequate exploration of search space are the decisive factors leading to sub-optimal performance. GPTuner, on the other hand, leverages more information from domain knowledge (e.g., suggested value, bound constraint, and special cases), and explores the search space with a novel Knowledge-based Optimization Framework. Finally, we conduct experiments to evaluate the performance of both methods in Section 8.2. Moreover, we investigate the effect the language model in Section 8.4. Whether using the same advanced model (both GPT-4) or a less complex one (i.e., DB-BERT with GPT-4 and GPTuner with GPT-3.5), GPTuner consistently outperforms DB-BERT in both scenarios.

3 MOTIVATION

M1: ML-based methods still incur significant tuning costs. State-of-the-art ML-based tuning methods require hundreds to thousands of iterations to converge to a good DBMS knob configuration. Such high tuning expenses stem from the inefficiency of runtime feedback-based optimization algorithms. Specifically, the feedback information is limited (i.e., a few benchmark runs cannot provide a complete picture of the DBMS performance under all conditions) and unstable (i.e., DBMS performance is not guaranteed to improve after each step of knob tuning). Given such weak feedback, it takes a substantial number of observations for ML models to have sufficient confidence in predictions, especially when the space is complex.

M2: Extensive domain knowledge helps, but not well-exploited. Since the early 2000s, when tuning systems take knob settings into consideration, extensive tuning knowledge has continually accumulated in the form of natural language and such knowledge does help in optimizing the search space. As shown in Figure 2 (left part), by analyzing the tuning knowledge, we can identify which knobs are worth tuning and gain insights of the typical value settings for knobs (e.g., suggested value to try, range constraint and special value). However, such wisdom seems exclusive to DBA and is not leveraged to mitigate the expensive tuning costs of ML-based approaches. There are approaches that utilize the static rules summarized by DBA to complete knob tuning. Unfortunately, these rules cannot capture the nuances of all workloads, and the updates of environments can make these static rules out of date. Thus we call for a more advanced approach to utilize the knowledge.

M3: LLM is a notable step forward, but not adequate yet. While it is widely acknowledged that domain knowledge is useful, such wisdom is considered inaccessible to machines due to the barriers in natural language understanding. Recently, the advent of Large Language Model (LLM) makes it possible to leverage such knowledge since we can utilize LLM to transfer the knowledge into a unified machine-readable format (e.g., structured data like JSON

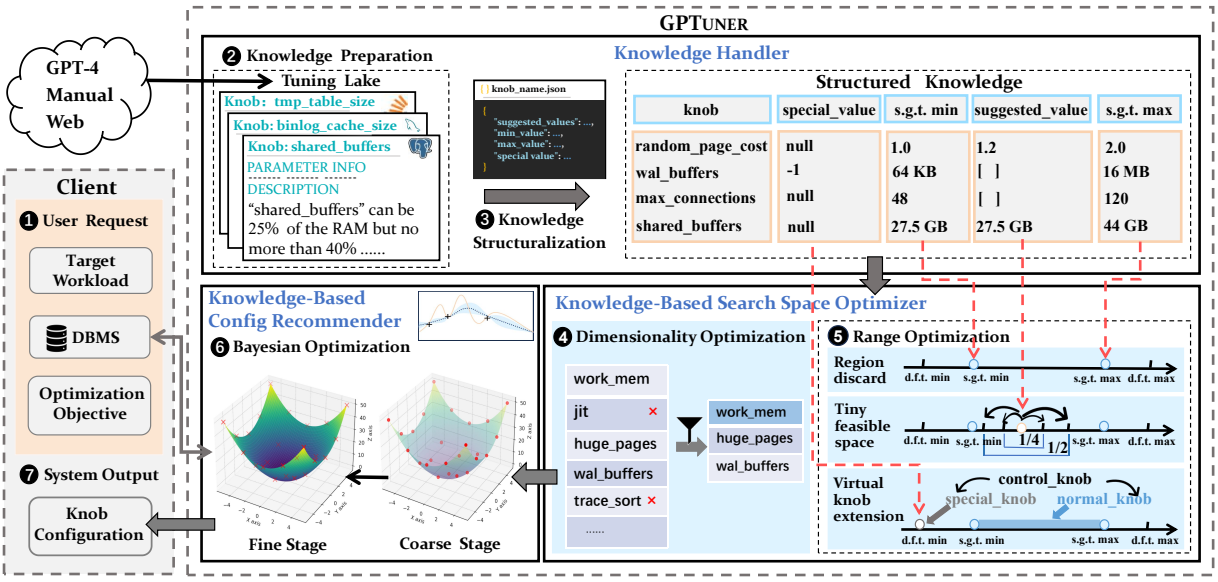


Figure 3: System Overview of GPTuner

and relational table). However, this process is still challenging for two main reasons. First, since domain knowledge typically comes in the form of DBMS documents and discussions from DBMS forums, it involves a complex and lengthy workflow to process such heterogeneous and noisy knowledge: data ingestion, data cleaning, data integration and data extraction. Second, the brittle nature of LLM (i.e. small modifications to the prompt can cause large variations in the model outputs) and the hallucination problem of LLM (i.e., LLM generates answers that seem correct but are factually false) make this challenge even more pronounced. For instance, as shown in Figure 2 (middle part), LLM can generate contents that are defective and even wrong. Thus, a reliable and effective approach to harness the power of LLM is both non-trivial and essential.

M4: Even if structured knowledge is developed, its integration into the optimization process is deficient. To the best of our knowledge, no DBMS tuning framework accounts for structured knowledge. Existing optimization algorithms like BO and RL do not support the integration of external knowledge directly, necessitating extensive modification to their standard workflows. Without modification, the only information that can be used is the range constraint (i.e., lower and upper bounds) recommended by manuals. However, there is much more useful information available (e.g., suggested value and special value). We present some simple ideas to utilize these values in Figure 2 (right part). Unfortunately, it is predictable that such naive ideas cannot serve as an effective solution, especially in the context of DBMS knob tuning where the problem is proven to be NP-hard [47]. Therefore, the current situation calls for an innovative optimization framework that supports the effective utilization of the structured domain knowledge.

Outline. To address the high tuning costs incurred in ML-based approaches (M1), we propose leveraging the invaluable but not well-exploited domain knowledge to enhance their efficiency (M2). However, this process is non-trivial and challenging, due to the limitations of LLM (M3) and the lack of a knowledge-aware optimization framework (M4). Therefore, we make the following technical contributions. To handle M3, we develop (1) an LLM-based pipeline with two error correction mechanisms in Section 5.1, (2) a prompt ensemble algorithm in Section 5.2. To address M4, we develop (3) an LLM-based knob selection strategy in Section 6.1, (4) three range optimization techniques in Section 6.2, and (5) a Coarse-to-Fine Bayesian Optimization Framework in Section 7. Finally, we use an ablation study to reveal the benefit of each module in Section 8.5.

4 OVERVIEW OF GPTUNER

Workflow. GPTuner is a manual-reading database tuning system to suggest satisfactory knob configurations with reduced tuning costs. Figure 3 presents the tuning workflow that involves seven steps. ❶ User provides the DBMS to be tuned (e.g., PostgreSQL or MySQL), the target workload, and the optimization objective (e.g., latency or throughput). ❷ GPTuner collects and refines the heterogeneous knowledge from different sources (e.g., GPT-4, DBMS manuals and web forums) to construct *Tuning Lake*, a collection of DBMS tuning knowledge. ❸ GPTuner unifies the refined tuning knowledge from *Tuning Lake* into a structured view accessible to machines (e.g., JSON). ❹ GPTuner reduces the search space dimensionality by selecting important knobs to tune (i.e., fewer knobs to tune means fewer dimensions). ❺ GPTuner optimizes the search space in terms of the value range for each knob based on structured knowledge. ❻ GPTuner explores the optimized space via a novel Coarse-to-Fine Bayesian Optimization framework, and finally ❼ identifies satisfactory knob configurations within resource limits (e.g., the maximum optimization time or iterations specified by users).

Components. GPTuner consists of three components: *Knowledge Handler*, *Knowledge-Based Search Space Optimizer* and *Knowledge-Based Configuration Recommender* and they work as follows:

Knowledge Handler. *Knowledge Handler* uses a DBMS tuning knowledge-oriented data pipeline to unify a structured view of the heterogeneous domain knowledge. At first, we propose an LLM-based pipeline to balance between cost and quality in Section 5.1. Next, we propose an *LLM-based Prompt Ensemble Algorithm* to

transfer the refined knowledge into a structured format such that it can be utilized by machines in Section 5.2.

Knowledge-Based Search Space Optimizer. The optimizer optimizes the search space from two aspects. Firstly, it reduces the size of the search space in terms of dimensionality. Specifically, we propose a workload-aware and training-free approach to select important knobs in Section 6.1. Secondly, we optimize the search space by focusing on the value range of each knob. We propose *Region Discard*, *Tiny Feasible Space* and *Virtual Knob Extension* methods to discard meaningless regions, highlight promising space and handle special situations, respectively (Section 6.2).

Knowledge-Based Configuration Recommender. The recommender uses a novel Coarse-to-Fine BO Framework to compute optimal configurations. In the first stage, BO only explores a discrete subspace of the whole heterogeneous space. This subspace is small in size but promising to contain good configurations since we generate it based on the reliable domain knowledge. In the next stage, in order to avoid the overlooking problem of coarse-grained search (i.e., it is inevitable to lose some useful configurations for any space reduction technique), BO explores the heterogeneous space thoroughly with the optimizations in Section 7. After the two stages, the recommender outputs the best-performing knob configurations found within the budget limits specified by users.

5 KNOWLEDGE HANDLER

5.1 Knowledge Preparation

In this section, we discuss how *Knowledge Handler* completes Knowledge Preparation task. As outlined in Algorithm 1, this task is to collect tuning knowledge from various resources (data ingestion, Line 1-3), filter out noisy contents (data cleaning, Line 4), summarize the legal parts (data integration, Line 5-6) and make sure the summarization is factual consistent with source contents (data correction, Line 7-10). The output is a *Tuning Lake* defined as follows:

Definition 1 (Tuning Lake). *Tuning Lake* $\mathcal{L} = \{d_1, d_2, \dots d_n\}$ is a set of n texts, where n is the number of configurable knobs and d_i is the natural language tuning knowledge for i-th knob. For example, "set `shared_buffers` to 25% of the RAM" (denoted as d_i) is the tuning knowledge for knob "`shared_buffers`" (the i-th knob).

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: Extracting knowledge from LLM. Except for the common tuning knowledge sources (e.g., manuals and web contents), we propose utilizing LLM as a knowledge source as well. Since GPT is trained on a vast corpus related to database [3], GPT itself is an informative manual and allows us to retrieve the knowledge through prompt. In practice, we surprisingly find that GPT can give reasonable suggestions that are not included in the manuals. Such suggestions come from web contents summarized by DBAs and were used as training data for GPT. Since it is impossible to provide all web contents to any system and GPT already knows much of it, it is reasonable to use GPT as a complementary source of knowledge. In case that GPT gives nonsense suggestions, we utilize LLM to handle such abnormal situation in the next step.

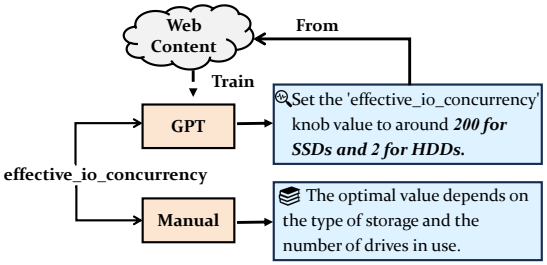


Figure 4: Knowledge Extraction from GPT

Example 1. Figure 4 depicts the process of extracting knowledge from GPT for knob "`effective_io_concurrency`". While the manual provides no useful suggestions, GPT recommends value 200 for SSDs and 2 for HDDs, which is available from a blog [34].

Step 2: Filtering noisy knowledge. The tuning knowledge comes from various sources and its quality cannot be guaranteed. Thus we filter out noisy knowledge by modeling this process as a "binary classification problem" and utilize LLM to solve it. We provide LLM with the candidate tuning knowledge for a knob and an official system view for this knob (e.g., *pg_settings* from PostgreSQL and *information_schema* from MySQL). Moreover, we give a few examples in the prompt to utilize the in-context learning ability of LLM [10]. LLM evaluates whether the tuning knowledge conflicts with the system view and we discard any knowledge that does conflict.

Example 2. Figure 5 illustrates the process of filtering out noisy knowledge for knob "`backend_flush_after`". The official system view categorizes it as an "integer" knob with a range from 0 to 256. However, the tuning knowledge recommends a value "between 0 and 1". Since this recommendation contradicts the official system view, we classify it as noisy and exclude it.

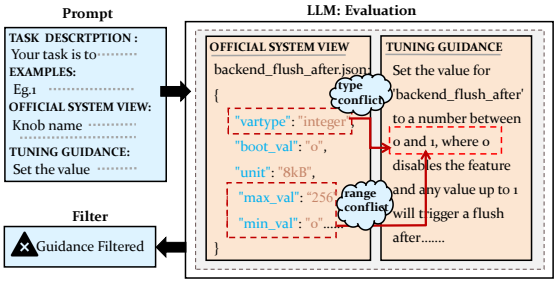


Figure 5: Filter Noisy Knowledge

Step 3: Summarizing knowledge from various resources. There can be multiple tuning knowledge for a knob. While such tuning guidance obey the official system view (Step 2), they could conflict with each other (e.g., different manuals provide distinct recommended values for the same knob). We handle this by manually setting priority for each information source based on its reliability. For example, official manuals are authoritative and thus have the highest priority, while LLM has the lowest priority due to its hallucination problem. We summarize the non-contradictory guidance and delete the content with low priority for the contradictory parts.

Example 3. Figure 6 presents the summarization process for knob "`shared_buffers`". All sources consistently recommend a value as 25% of the RAM. However, there is a conflict in the upper bound: GPT advises "no more than 4 GB" while manual recommends "no more than 40% of RAM". Since manual has a higher priority, the GPT suggestion is omitted, which is appropriate since "not more than 4 GB" seems specific to certain machines and lacks generality.

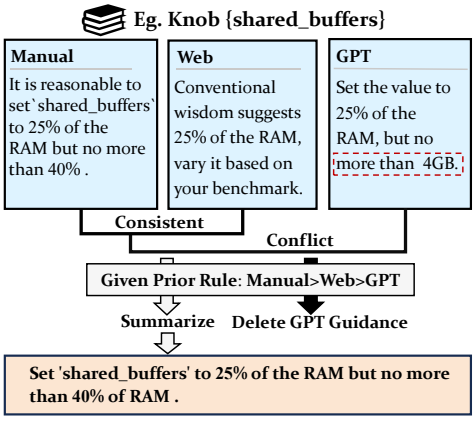


Figure 6: Knowledge Summary

Step 4: Checking factual inconsistency. In the last step, the summary task is completed by LLM and the generated summaries may be factually inconsistent (i.e., the summary includes information that does not appear in the source contents or even contradicts it). Since GPT outperforms previous methods as a factual inconsistency evaluator [32], we utilize GPT to check this inconsistency. For each knob, the summarization and the source contents are included in prompt for GPT to determine whether there is an inconsistency. If an inconsistency is detected, GPT is prompted to recreate the summarization. This newly generated summary, along with its source contents, are once again provided to GPT. This process is repeated until GPT identifies no errors, producing the final summarization.

Example 4. Figure 7 presents the process of inconsistency check for knob "`shared_buffers`". The inconsistency occurs since the summarization "no more than 40% of OS cache" contradicts the source content "no more than 40% of the RAM". Next, GPT learns from the mistake and regenerates the summarization. Finally, the newly generated summarization passes the inconsistency check.

Robustness Study. It is important to note that the hallucination problem still remains a challenge in the NLP field, and thus we propose two error correction mechanisms (i.e., step 2 and 4 are

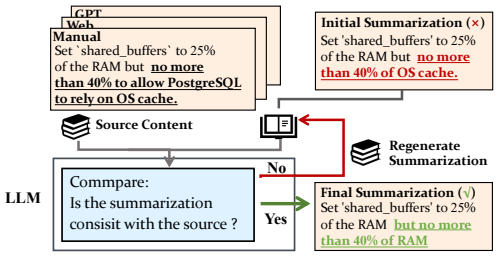


Figure 7: Inconsistency Check

designed to correct step 1 and 3, respectively) to minimize such impact as much as possible. To quantify their reliability, we manually prepare and open-source two datasets, and conduct experiments to measure the effectiveness of the two mechanisms. Moreover, we study the effect of domain knowledge quality (e.g., outdated and incorrect knowledge), and the LLM ability on GPTuner's tuning performance. More details are provided in Section 8.4.

5.2 Knowledge Transformation

In this section, we build a structured view (namely *Structured Knowledge*) of the *Tuning Lake* such that it can be utilized by ML. Formally,

Definition 2 (Structured Knowledge). Structured Knowledge $\mathcal{S}$ maintains a structured view s_i for each tuning knowledge d_i from Tuning Lake $\mathcal{L}$, where s_i is defined by a set of attributes $A = \{a_1, a_2, \ldots, a_n\}$ (e.g., $a_1 = suggested\_values$) with corresponding values $V = \{v_1, v_2, \ldots, v_n\}$ (e.g., $v_1 = 4\ GB$).

• **Determining the Attributes.** In the context of DBMS knob tuning, we primarily consider four types of attributes: *suggested_values*, *min_value*, *max_value* and *special_value*. Firstly, the most typical hint to consider is the recommended values for a knob since they performed well in the past practice and can serve as good starting points for new scenario. Secondly, we take into account the minimum and maximum values suggested in the tuning knowledge. This is because the default value ranges provided by DBMS vendors are excessively broad, complicating the optimization process (e.g., there will be a large number of values whose effect on the DBMS performance is unknown and this means ML models require much more iterations to converge to good configurations) and introducing the risk of system crashes (e.g., set memory-related knobs to a value higher than the available RAM can crash the DBMS). Finally, there are knobs with "special values" and these values are hard to be modeled by ML methods since they lead to distinct behaviours of DBMS. Thus we use *special_value* to handle such special situations. The techniques to utilize these values are detailed in Section 6.2.

• **Determining the Attribute Values.** Extracting specific knob values for certain attributes from given texts is challenging due to the brittle nature of LLM (i.e., small modifications to the prompt can cause large variations in the LLM outputs [27]). Since it is useful to acquire a more reliable result by aggregating multiple imperfect but effective results [5, 43], we develop a *Prompt Ensemble Algorithm* to determine the attribute values effectively and its workflow is summarized in Figure 8. Specifically, it involves three steps: modeling the extracting task as a Natural Language Problem such that it can be answered by LLM, varying the prompts to prepare multiple results, and aggregating these results to produce the final result:

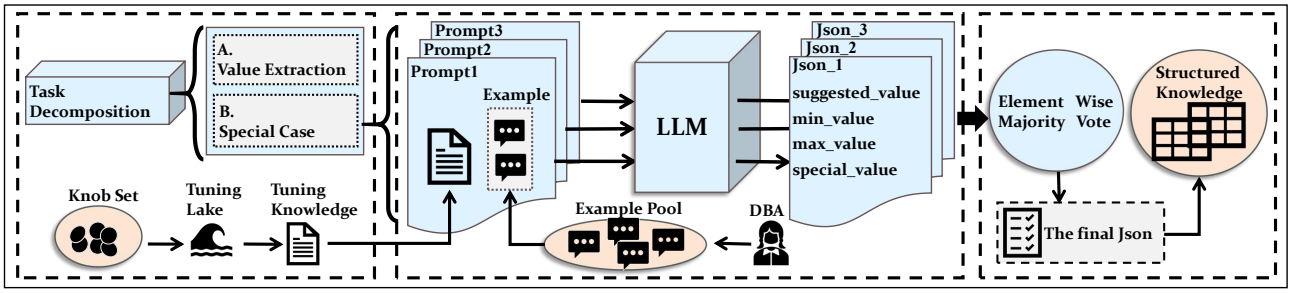


Figure 8: Knowledge Transformation Workflow

Step 1: we model the transformation task as a series of information extraction problems. At first, we decompose the transformation task into two subtasks of extracting (1) *suggested_values*, *min_value*, *max_value* and (2) *special_value*, respectively. We tackle *special_value* separately because *special_value* has its own context, which will be discussed in Section 6.2. Next, we prepare the prompt for each subtask. The {*target_values*} is a placeholder to be determined by task type (e.g., it is replaced with the definition of special values if we want to extract *special_value* from the {*knowledge*}). We also include examples in the prompt to utilize the in-context learning of LLM.

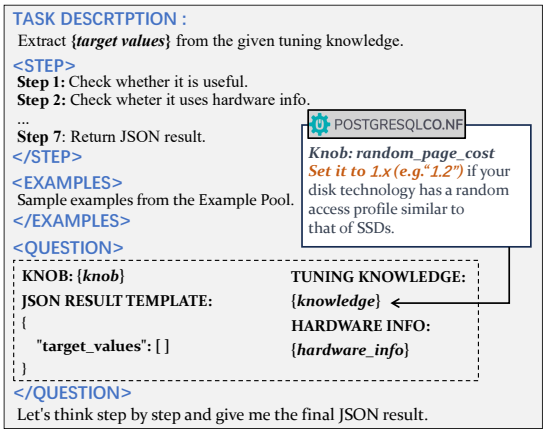


Figure 9: QA Prompt Template

Step 2: we vary the prompts by changing the examples provided for few-shots learning. Specifically, we manually prepare K examples and store them in an *Example Pool*. Then we randomly sample n examples for each prompt ($n \leq K$). In our experiments, we manually prepare 10 examples (i.e., $K = 10$) since 10 examples are empirically good enough to provide the diversity requirement of ensemble algorithms [2]. More examples can be added if domain experts are available. Regarding the n, more sampled examples might provide better results. However, this leads to a longer prompt with more tokens to be processed, which subsequently results in longer processing time and higher monetary costs. Moreover, too many examples might lead to long-context errors, and thus we select $n = 3$ as suggested in [5].

Step 3: we aggregate the results via a majority vote strategy. LLM generates a candidate JSON for each prompt and we aggregate the results by taking an *Element-Wise Majority Vote* strategy, where *Element* refers to our target attribute. For each attribute, we rank the values extracted based on occurrence frequency and choose the value of the highest frequency as the final result for that attribute. The resulting JSON is stored in the Structured Knowledge $\mathcal{S}$.

6 SEARCH SPACE OPTIMIZER

6.1 Dimensionality Optimization

In this section, we identify knobs that have a significant impact on the DBMS performance and only tune these important knobs.

Motivation. While there are hundreds of knobs in DBMS (e.g. PostgreSQL v14.7 has 351 knobs), not all of them significantly impact DBMS performance, and it is infeasible to take all knobs into account due to the curse of dimensionality. Therefore, we only select important knobs to tune. Recent studies have shown that tuning a small number of knobs is enough to yield near-optimal performance while significantly reducing tuning costs [22, 53]. Existing approaches rely on ML-based algorithms to select important knobs and this requires hundreds to thousands of evaluations on DBMS under different workloads and configurations [22, 52, 60]. Differently, DBAs seek help from manuals to select which knobs are worth tuning and this yields better performance than ML-based algorithms [53]. However, it takes significant burden for DBAs to handle hundreds of knobs. Thus, we hope to remove the tedious burden by proposing a workload-aware and training-free approach via LLM. Specifically, we prompt LLM to simulate DBA's empirical judgement in real-world scenarios by considering the four factors:

(1) System-Level selects knobs based on the specific DBMS product. After years of tuning practice, it is empirically known which knobs are important for a certain DBMS product. For instance, it is widely discussed that we can gain substantial performance improvement by tuning "`shared_buffers`" and "`max_wal_size`" for PostgreSQL, "`innodb_buffer_pool_size`" and "`innodb_log_file_size`" from MySQL [53]. More importantly, we find such wisdom is included in the corpus of GPT-4 [37] and we can extract it by prompting GPT-4 to recommend knobs to tune based on the DBMS product.

(2) Workload-Level selects knobs based on the workload type. The workload type is informative for selecting important knobs since different workload types have distinct requirements on DBMS resources, which are regulated by knobs. For example, one typical scenario for OLTP workload is I/O-intensive, where write operations compete for the limited disk I/O. Thus we should consider disk-related knobs like "`effective_io_concurrency`". For analytical queries from OLAP workload, we should tune planning-related

and system resource-related knobs to handle their complex structure and resource-intensive nature [65].

(3) Query-Level selects knobs based on the bottleneck of queries. It is useful for DBAs to delve into the execution plan of the time-consuming and frequently executed queries, where an experienced DBA can diagnose performance bottlenecks and focus on related knobs. Equipped with the analysis ability of LLM [66], we can include the execution plan in the prompt such that LLM can choose the bottleneck-aware knobs. For example, if LLM detects "Sequential Scan" in the execution plan and the target table contains a large number of rows, LLM will recommend adjusting scan-related knobs like "`random_page_cost`" since it influences the PostgreSQL query planner's bias towards index scans or sequential scans.

(4) Knob-Level complements interdependent knobs to a given target knob set. One important reason why ML techniques outperform DBA is that they can handle the dependencies between knobs [53]. However, if the given knob set contains the important knobs but excludes the interdependent knobs, such ability is wasted. Since many dependencies are explicitly mentioned in the manuals, we can leverage the text analysis ability of LLM to read the manuals and capture such knobs. For example, the official PostgreSQL document suggests "Larger settings for '`shared_buffers`' usually require a corresponding increase in '`checkpoint_segments`' " [41], indicating that we should consider the two knobs at the same time.

Algorithm 2: 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 $C \leftarrow FILTER(\mathcal{K})$;
`// 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 $\quad \mathcal{E}_i \leftarrow EXECUTE(\mathcal{D}, q_i)$;
$\quad$ `// Get execution plan for query` q_i `from` $\mathcal{D}$
7 $\quad C_{q_i} \leftarrow \mathcal{F}(C, \mathcal{E})$;
8 $\quad C_q \leftarrow C_q \cup C_{q_i}$;
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** $\mathcal{T}$;

Based on the above four factors, we develop the **LLM-based Knob Selection** as Algorithm 2. It aspires to utilize LLM $\mathcal{F}$ to select important knobs $\mathcal{T}$ from the Knob Set $\mathcal{K}$ for a DBMS $\mathcal{D}$ under a specific workload $\mathcal{W}$. The algorithm starts by preparing a configurable knob set C, which excludes knobs related to debugging, security and path-setting (Line 1). Next, it selects knobs from four different levels: System Level, Workload Level, Query Level and Knob Level. In the System Level Selection, knobs hugely influencing the DBMS performance are identified, producing the set C_s (Line 2). During the Workload Level Selection, we provide LLM with the type of the workload (e.g., OLTP or OLAP) and the optimization target (e.g., throughput or latency) to prepare knob set C_w (Line 3). The Query Level Selection delves into each query q_i within the workload (Line 4-9). Specifically, the execution plan $\mathcal{E}_i$ of each query is retrieved, and LLM $\mathcal{F}$ extracts influential knobs set C_q by diagnosing the performance bottlenecks of each query and identifying knobs related to the bottleneck. Finally, we leverage LLM $\mathcal{F}$ to replenish interdependent knobs to the union of C_s, C_w and C_q, resulting in the final Target Knob Set $\mathcal{T}$(Line 10).

6.2 Range Optimization

In this section, given each knob's unique semantics and associated tuning knowledge, we optimize the value range for each knob.

Region Discard. We utilize *min_value* and *max_value* to discard some regions for the following cases. (1) The regions are unlikely to result in promising performance. For knob "`random_page_cost`", the value range regulated by DBMS is $[0, 1.79 \times 10^{308}]$. Given the large value range, the algorithm is likely to sample very large values and this will lead to poor DBMS performance since it is suggested to set it to "1.x" if the disk has a random access profile similar to that of SSDs [42]. Equipped with this prior knowledge, we release the burden for optimizers to trial vast but meaningless space. (2) The regions could seize too many system resources. For resource-related knobs like `shared_buffers`, a value close to the maximum system resource could be detrimental to other services that are running on the same machine. (3) The regions that can make the DBMS crash. For resource-related knobs, setting a value that exceeds the resource limits could prevent the DBMS from starting (e.g., we cannot set memory-related knobs to a value higher than the available RAM). Finally, the recommended range [*min_value*, *max_value*] is much more narrow than the range provided by DBMS vendors.

Tiny Feasible Space. We use *suggested_values* from Structured Knowledge to define a discrete space for each knob. Such values are valuable since they performed well in the past and can serve as good starting points for new scenario. However, they may not be suitable for all cases, as the optimal knob setting depends on the specific environment, which can be diverse. Instead of relying on these values only, we can apply a set of multiplicators for each suggested value of all numerical knobs. The intuition behind is to deviate the suggested value in different directions (smaller or bigger) with different extents. Formally, given a set of multiplicators $M = \{m_1, m_2, \ldots, m_n\}$ and a suggested value V for a knob k, the search space Ω for this knob is defined as:

$$\Omega(k) = \{\alpha \cdot V |\ \alpha \in M\} \tag{2}$$

However, this heuristic approach ignores the value ranges of knobs and the multiplication results could be useless. For example, knob "`checkpoint_timeout`" has a value range from 30 to 86400 with 90 as a recommended starting point [15]. Given the maximum factor used in DB-BERT [50], which is 4, the maximum value to try is $90 \times 4 = 360$ and this is much smaller than 86400, which means a lot of promising values are ignored. Thus we develop the following algorithm to address this limitation by considering the value range and calculating the multiplicators dynamically. For knob k, we denote its maximum (minimum) value as U, the multiplicator is

calculated by the following formula:

$$\alpha = 1 + \frac{\beta}{V}(U - V),\ \beta \in \{r_1, r_2, \ldots, r_n \mid r_i \in [0, 1]\} \quad (3)$$

where β is a scaling coefficient with a value from 0 to 1, and the n candidate values r_i are predefined by users. The choice of U determines the deviation direction (e.g., maximum makes value bigger while minimum makes it smaller) and β controls the changing extends. Specifically, V remains the same or is extended to the maximum (minimum) when β is set to 0 and 1, respectively. For β value between 0 and 1, suggested value V moves proportionally to its maximum (minimum). In our experiments, $\beta \in \{0, 0.25, 0.5\}$. We conduct this deviation process for all numerical knobs and the resulting discrete space is our *Tiny Feasible Space*, where *Tiny* means the possible number of values for knobs is significantly reduced, and *Feasible* indicates the chosen values are promising.

Algorithm 3: Virtual Knob Extension

Input: Knob Set $\mathcal{K}$; Skill Library $\mathcal{S}$; Optimizer $\mathcal{O}$.
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 $\mathcal{O}$ chooses a value from $\{0, 1\}$ for *control_knob*;
7 **if** *control_knob is 0* **then**
8 Activate *normal_knob* in $\mathcal{O}$;
9 **else**
10 Activate *special_knob* in $\mathcal{O}$;
11 **end**
12 **end**

Virtual Knob Extension. This extension is utilized to handle knobs that use a special value to do something different from what the knobs normally do [23, 39]. For example, knob "`lock_timeout`", with a value range from 0 to 2147483647, controls the maximum allowed duration of any wait for a lock. When it is set to zero, the timeout function is disabled and this makes "0" a special value. However, optimizers may never trial this value (even though it could be optimal) since the likelihood of sampling it is extremely low [23] and the DBMS performance will be modeled to degrade as the knob value converges to zero [39]. Thus we develop a *Virtual Knob Extension* technique to handle these knobs with special values, as outlined in Algorithm 3. Firstly, we utilize *Structured Knowledge* to select which knobs have the special values (Line 1). Such information is explicitly provided in official documents [41] and thus can be understood and summarized by *Knowledge Handler* (Section 5.1). Secondly, we add "virtual knobs" (*control_knob*, *normal_knob* and *special_knob*) for each knob with special value (Line 3). *control_knob* is a knob with a value of zero or one to determine which value range will be used (Line 6). *normal_knob* and *special_knob* represent the normal and the special value range of the original knob, respectively. Based on the value of *control_knob*, only one of *normal_knob* and *special_knob* will be activated (Line 7-11). Specifically, if *control_knob* is set to zero (one), only *normal_knob* (*special_knob*) will be activated. Above technique makes the special values of knobs to be considered by optimizers.

Example 5. Figure 10 presents the Virtual Knob Extension for knob "`lock_timeout`". It has a value range from 0 to 2147483647 with 0 as its special value. It is extended to *control_lock_timeout* with a value of 0 or 1, *normal_lock_timeout* with a value from 1 to 2147483647, and *special_lock_timeout* with a value of 0. If *control_lock_timeout* is set to 0 (1), only *normal_lock_timeout* (*special_lock_timeout*) will be activated.

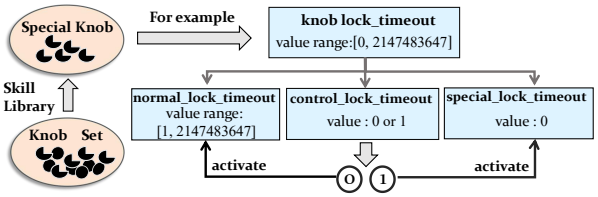


Figure 10: Virtual Knob Extension

7 CONFIGURATION RECOMMENDER

In this section, we discuss how to explore the optimized space to recommend well-performing configurations.

Basic Idea of Bayesian Optimization. Bayesian Optimization is a sequential model-based algorithm [46]. It consists of two core components: the *surrogate model* and the *acquisition function*. A surrogate model takes a configuration as input and predicts the DBMS performance. The acquisition function evaluates the utility of candidate configurations (e.g., Probability of Improvement (PI) or Expected Improvement (EI)), and we choose the next configuration with the maximum utility. After the surrogate model is initialized, BO works iteratively as follows: (1) selecting the next configuration by maximizing acquisition function, (2) evaluating the configuration on DBMS and updating the surrogate model with the new observation. This process is repeated until it runs out of resources.

Limitation of Bayesian Optimization. The key to the success of BO-based methods is its surrogate model. Recent works utilize random forest as the surrogate model to leverage its efficiency in modeling high-dimensional and heterogeneous search space, achieving the state-of-the-art performance [48, 60]. However, the number of samples required to have sufficient confidence in predictions could still be significant [23, 24]. Specifically, it requires hundreds of iterations to achieve satisfactory performance, which is resource-intensive and time-consuming. The key observation of this work is that such iteration cost could be significantly reduced if we integrate the domain knowledge into the optimization process, which motivates the following novel optimization framework.

Coarse-to-Fine Bayesian Optimization Framework. The intuition behind is to take advantages of the efficiency of coarse-grained search (i.e., we aim to acquire non-optimal but effective solutions with low expenses) and the thoroughness of fine-grained search (i.e., we aim to find the possible optimal solutions by exploring the space thoroughly). Algorithm 4 describes the workflow and highlights the main differences from the basic BO algorithm.

Algorithm 4: 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 $\quad \vec{x}_i \leftarrow \arg\max_{\vec{x} \in \mathcal{T}} \mathcal{A}(\vec{x})$;
7 $\quad$ Evaluate $\vec{x}_i$ on $\mathcal{D}$ to get performance y_i;
8 $\quad$ 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 $\quad \vec{x}_i \leftarrow \arg\max_{\vec{x} \in \mathcal{P}''} \mathcal{A}(\vec{x})$;
15 $\quad$ Evaluate $\vec{x}_i$ on $\mathcal{D}$ to get performance y_i;
16 $\quad$ Update $\mathcal{M}$ with $(\vec{x}_i,\ y_i)$;
17 **end**
18 $\mathcal{X} \leftarrow \arg\max_{\vec{x}_i} y_i$;
19 **return** $\mathcal{X}$;

Coarse-grained Stage. In the first stage, we only explore part of the whole space. It is a widely adopted approach to discretize the value ranges of parameters evenly for coarse-grained search. However, we will lose too many promising solutions because such space reduction technique is inherently imprecise and non-adaptive. We seek help from knowledge to reduce the space while still retaining the potential for optimal results. Specifically, we explore the *Tiny Feasible Space* (Line 1) defined in Section 6.2, which is small but reliable because it comes from manuals. Following previous works [14, 23, 60], we initialize the surrogate model with ten samples ($n = 10$) generated by Latin Hypercube Sampling (LHS) [33], which distributes samples evenly across the whole space. Instead of sampling points from the whole space $\mathcal{P}$, we sample from the Tiny Feasible Space $\mathcal{T}$ (Line 2). After the samples are evaluated on $\mathcal{D}$ (Line 3) and the surrogate model is initialized (Line 4), we explore $\mathcal{T}$ with the BO algorithm for C iterations (Line 5-8). After the coarse-grained stage, we try out $n + C$ configurations and this already yields non-optimal but promising results in practice, owing to the guidance of domain knowledge.

Fine-grained Stage. Since it is inevitable to lose some important configurations for any space reduction technique, we explore the space thoroughly until we run out of resource limits or reach expected performance improvement (Line 13-17). However, this process could be exhausted, especially when there are hundreds of knobs to tune. Thus we make optimizations to enhance the search: (1) we bootstrap BO with the samples from the first stage (Line 10), (2) we narrow down space $\mathcal{P}$ with the *Region Discard* technique (Line 11), and (3) we take into account the knobs with special values with the *Virtual Knob Extension* technique (Line 12). Since (2) and (3) are discussed in Section 6, we focus on (1) bootstrap next.

Effectiveness of our Bootstrap. While using good starting points to initialize the surrogate model can enhance BO extensively [17, 58], it is still challenging to prepare such high-quality samples. Existing approaches rely on historical results to complete this task (i.e., initialize the surrogate model with the results from the most similar workloads) [52, 53, 61]. However, it is expensive to prepare such results. For example, it takes "over 30k trials per DBMS" to bootstrap OtterTune [52] and we have to re-build such results from scratch for changes in hardware components and software versions [7]. Moreover, even if we can prepare such historical results, it is still challenging to transfer knowledge from previously seen workloads [6, 60]. Therefore, we omit the preparation procedure and difficult transfer process, and generate samples for new tuning tasks on the fly (coarse-grained stage). This stage of the process is beneficial and introduces no additional overhead. For instance, the target space for the search is small and manageable because it is derived from manual recommendations.

8 EXPERIMENTAL EVALUATION

8.1 Experimental Setup

Workloads. Following the setup of DB-BERT [50], we use TPC-H (OLAP type) with scale factor 1 and TPC-C (OLTP type) with scaling factor 200 as our benchmarks. TPC-C uses ten terminals with unlimited arrival rate and the implementations of benchmarks are from BenchBase [13].

Hardware. We conduct our experiments using a cloud server with a 24-core Intel Xeon E5-2676 v3 CPU, 110 GB of RAM and a 931 GB WD SSD. We utilize a GeForce RTX 2080 Ti with 22 GB of memory.

Baselines. GPTuner is implemented with SMAC3 library [29] and uses OpenAI completion API of GPT-4 [37]. We compare GPTuner with the following state-of-the-art methods:

- **DDPG++.** DDPG is a Reinforcement Learning algorithm used in CDBTune [59] and improved in [53] (DDPG++). We use the implementation from the authors [53].
- **GP.** GP is a Gaussian Process-based optimizer used in iTuned [14] and OtterTune [52]. We implement it using SMAC3 library [29].
- **SMAC.** SMAC is the best-performing BO-based method with random forest as its surrogate [60]. We use the author version [29].
- **DB-BERT.** DB-BERT is a database tuning tool [50] that uses BERT for text analysis to guide Double Deep Q-Networks [19], a Reinforcement Learning algorithm. We use the author version [50].

Tuning Settings. We run experiments with PostgreSQL v14.9 and MySQL v8.0. Following previous works [23, 59, 60], all algorithms tune the same 60 and 40 knobs of PostgreSQL and MySQL respectively, and the generic optimizations like the clone schema from HUNTER [6] as well as the space projection and bucketization from LlamaTune [23] are not used for any method. We run three tuning sessions for each method, with each session consisting of 100 iterations and each iteration requires a stress test for the target workload. Aligning with the latest experimental evaluation [60], we optimize throughput for OLTP workload and 95-th percentile latency for OLAP workload, reporting the median and quartiles of

the best performances. We restart the DBMS each iteration since the change of many knobs requires it. For a configuration that makes DBMS crash, we assign the performance twice worse than the worst configuration ever seen to make optimizers learn such faults, as suggested in [53]. For BO-based methods, we follow the setting of iTuned [14] and OtterTune [52] by executing 10 configurations generated by Latin Hypercube Sampling (LHS) [33] to initialize the surrogate model. For RL-based methods, we follow recent works [6, 50] and do not train the neural network since it is evaluated that the trained network suffers from over-fitting [60].

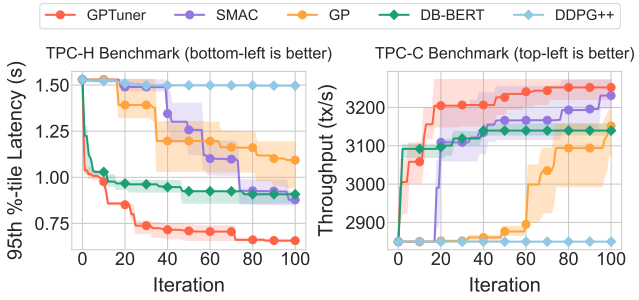


Figure 11: Best performance over iterations on PostgreSQL

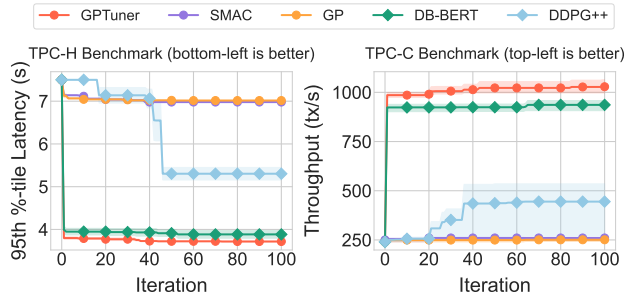


Figure 12: Best performance over iterations on MySQL

8.2 Performance Comparison

Optimizing for PostgreSQL. As shown in Figure 11, GPTuner finds the best performing knob configuration with significantly less iterations in both benchmarks. For example, GPTuner rapidly achieves significant performance improvement and reaches near-optimal latency (44.4% less latency) with only 20 iterations in terms of TPC-H benchmark. Moreover, GPTuner continues to achieve a 12.7% reduction of latency after the 20-th iteration. While DB-BERT presents similar performance in the first 20 iterations (37.5% less latency), it fails to further reduce the latency (only 3.4% reduction). For methods that do not use domain knowledge, they converge much slower than GPTuner (26x slower on average) and fail to find a configuration comparable to GPTuner (35.6% worse on average). The results on TPC-C are similar to that on TPC-H, where GPTuner converges 12.6x faster and reaches the highest throughput.

Optimizing for MySQL. The results on MySQL are presented in Figure 12, which are similar to that on PostgreSQL. In terms of TPC-H, GPTuner significantly reduces the latency by 48.5% at the very beginning, surpassing the best performance achieved by all other baselines within 100 rounds. For TPC-C, GPTuner achieves the highest throughput of 1029 tx/s over the default 240 tx/s, which is 10% and 131.7% better than DB-BERT and DDPG++, respectively. Notably, GP and SMAC fail to have considerable performance improvement on both benchmarks. This is because the default value ranges provided by DBMS vendors are excessively broad, making the optimizers struggle to explore the vast search space without the guidance of domain knowledge.

Summary. GPTuner consistently outperforms all state-of-the-art methods on different DBMS, benchmarks and metrics. This is because GPTuner significantly benefits from the reliable Tuning Lake, the ease-of-use Structured Knowledge, the optimized search space and the efficient Coarse-to-Fine BO framework. To explain how GPTuner produces the best performance, we conduct an ablation study to understand the benefits of each module in Section 8.5.

8.3 Scalability Study

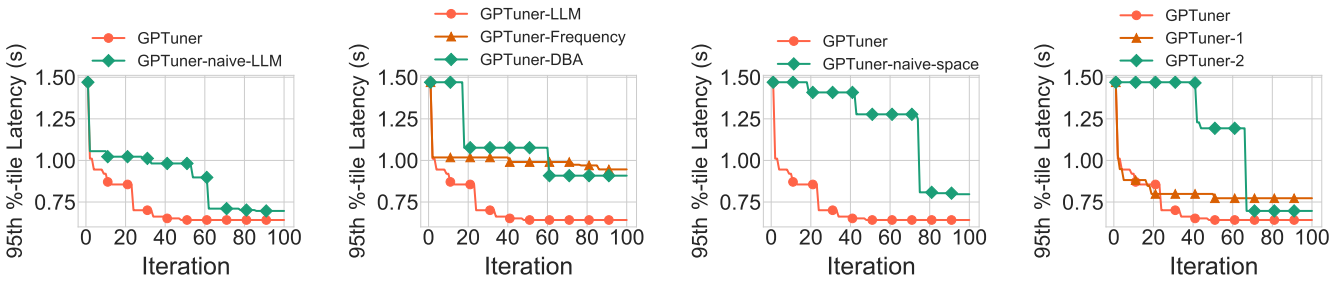
Database Size. We study the impact of database size on tuning performance by varying the scale factor of TPC-H from 1 to 10 and 50. As shown in Figure 14, compared to other methods, GPTuner finds better configurations in much fewer iterations in all database sizes. When the factor is 50, GPTuner identifies a configuration better than any other methods just in the 30-th iteration and finally achieves a 42.5% reduction in latency. An interesting observation is that knowledge-enhanced approaches (i.e., GPTuner and DB-BERT) are affected by the increasing of database sizes slightly, while other methods suffer from it. This can be attributed to the fact that the complexity of modeling the relationship between configurations and DBMS performance is higher in the larger database sizes, since more performance bottlenecks are revealed. GPTuner learns such experience directly from domain knowledge rather than through iterative trial and error, and thus showcases superior performance, a result similar to previous work [50].

Search Space Dimensionality. We study the impact of space dimensionality on tuning performance by varying the number of target knobs from 50 to 100 and 150. Note that DB-BERT is excluded in this experiment since it relies on a frequency-based selection strategy to tune only a fixed set of knobs mentioned in the input documents (e.g., it tunes 22 knobs when tuning TPC-H for PostgreSQL), making it infeasible to manually control the number of target knobs [50]. As shown in Figure 16, GPTuner consistently showcases the best performance in all space sizes. While other approaches perform well in low-dimensional case, their performance deteriorates in high-dimensional cases. This is because although the space dimensionality is fixed for all methods, GPTuner still benefits from its Range Optimization as discussed in Section 6.2.

8.4 Robustness Study

Effect of Error Correction Mechanisms. We measure the effectiveness of the two error correction mechanisms in our Knowledge Preparation stage with two manually prepared datasets.

(1) Evaluating step 2 as a filter of step 1. Firstly, we utilize GPT-4 to generate domain knowledge for 150 knobs from PostgreSQL, and then conduct a survey with a team of five database experts to annotate the generated knowledge. Among all pieces of knowledge, 123 of them are consistent with the *system_view* of PostgreSQL and the remaining 27 are not. Next, we use GPT-4 to filter out noisy knowledge as discussed in Section 5.1. GPT-4 performs well in this task by identifying 20 pieces of noisy knowledge out of 27 pieces with a recall rate of 74%, and classifying 109 pieces of correct



(a) Enhanced Knowledge (b) Knob Selection (c) Space Optimization (d) Coarse-to-Fine BO Framework

Figure 13: Ablation study of GPTuner on TPC-H benchmark (bottom-left is better)

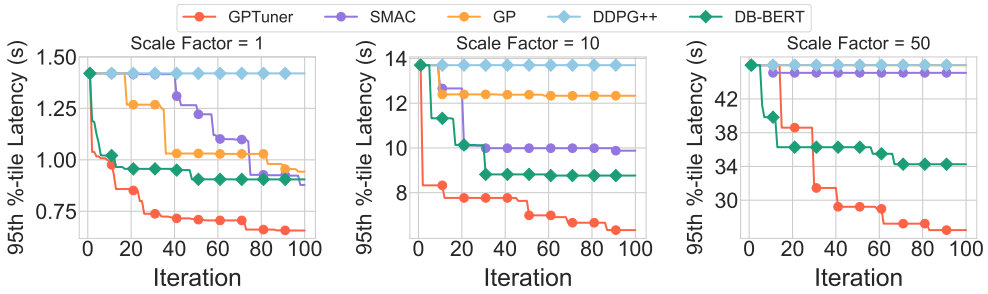


Figure 14: Effect of Database Size on Tuning Performance (bottom-left is better)

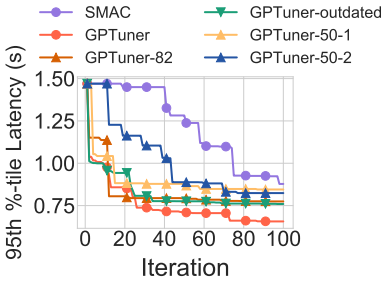


Figure 15: Effect of Knowledge Quality (bottom-left is better)

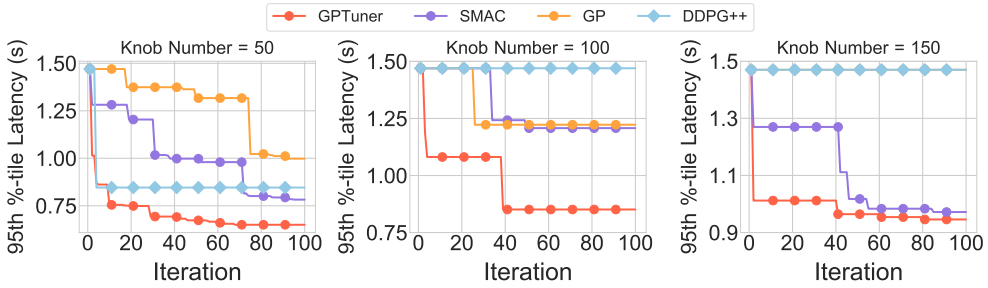


Figure 16: Effect of Space Dimensionality on Tuning Performance (bottom-left is better)

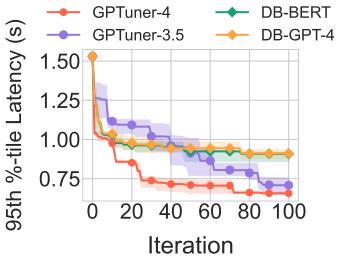


Figure 17: Effect of Language Models (bottom-left is better)

knowledge out of 123 pieces with a specificity rate of 88.6%. In summary, equipped with our step 2 as a filter of noisy knowledge, we improve the accuracy of input knowledge from 82% to 94%.

(2) Evaluating step 4 as a corrector of step 3. At first, we prepare domain knowledge from GPT-4, manuals and web for 150 knobs from PostgreSQL. Next, we use GPT-3.5 and GPT-4 to complete step 3 and step 4, where they detect 7 and 4 inconsistencies respectively and rewrite all of them correctly. Since the number of inconsistencies is low and we want to further evaluate the model's reliability, we additionally prepare a dataset containing 50 summaries that are factual inconsistent with the original knowledge. Among the 50 incorrect pieces of knowledge, GPT-4 detects 39 of them are factual inconsistent with an accuracy of 78%, and rewrites all of them correctly with an accuracy of 100%. The degradation of accuracy can be attributed to the complexity and diversity of the manually prepared knowledge, which challenges language models. Given that GPT-4 detects and corrects all inconsistencies in the real dataset, it is feasible to utilize GPT-4 as a corrector of step 3.

Effect of Domain Knowledge Quality. We study how GPTuner's performance is affected by the quality of input domain knowledge.

(1) Outdated Knowledge. We only use PostgreSQL manual of version 9.1 as the knowledge input for GPTuner to tune PostgreSQL v14.9, and this version is denoted as "GPTuner-outdated".

(2) Inaccurate Knowledge. Firstly, we use the domain knowledge with an accuracy of 82% and 94% from the "Evaluating step 2 as a filter of step 1" part, which are denoted as "GPTuner-82" and "GPTuner" respectively. Next, we use the knowledge from the "Evaluating step 4 as a corrector of step 3" part. Specifically, among all the target knobs, we randomly select half of them and assign the factual inconsistent knowledge to them. This procedure is repeated for five times and we present two representative results, which are denoted as "GPTuner-50-1" and "GPTuner-50-2", respectively.

As shown in Figure 15, the higher the knowledge quality, the better the tuning performance. It is noteworthy that outdated knowledge only affects tuning performance slightly, while inaccurate knowledge impacts tuning performance with different degrees. This is related to our motivation in Section 6.1, highlighting that some

knobs are crucial in determining tuning performance and GPTuner benefits a lot if the knowledge about these important knobs is correct. On one hand, in production environment, we encourage users to double check such knowledge. On the other hand, given that GPTuner under the noisy knowledge inputs is still better than the best optimizer without knowledge input (i.e., SMAC) as evaluated in [60], we believe GPTuner is robust enough to learn from mistakes (i.e., Bayesian Optimization) and achieve satisfactory performance optimization ultimately.

Effect of Different Language Models. We analyze the effect of language models on tuning performance. Specifically, DB-BERT is upgraded to use GPT-4 ("DB-GPT-4"), and GPTuner is tested with "gpt-3.5-turbo" and "gpt-4", named "GPTuner-3.5" and "GPTuner-4", respectively. As shown in Figure 17, DB-BERT does not benefit from the improvement of language model, implying its design is less dependent on the quality of language model. Both versions of GPTuner outperform DB-BERT methods. Moreover, GPTuner significantly benefits from GPT-4, demonstrating its ability to leverage more sophisticated language model to enhance the tuning process.

8.5 Ablation Study

Effect of Knowlege Preparation and Transformation. To verify the effectiveness of the pipeline in Section 5.1 and the *prompt ensemble algorithm* in Section 5.2, we additionally prepare functionally complete prompts and exclude the two techniques to implement a naive approach denoted as "GPTuner-naive-LLM". As shown in Figure 13a, GPTuner significantly outperforms GPTuner-naive-LLM in both convergence speed (4x speedup) and latency reduction.

Effect of Knob Selection. To demonstrate the effectiveness of the LLM-based Knob Selection algorithm in Section 6.1, we compare GPTuner with different target knobs: (1) knobs selected by DBA, (2) knobs selected by occurrence frequency (used in DB-BERT), and (3) knobs selected by our Knob Selection algorithm, which are denoted as "GPTuner-DBA", "GPTuner-Frequency" and "GPTuner-LLM", respectively. As shown in Figure 13b, GPTuner-LLM significantly outperforms GPTuner-Frequency and GPTuner-DBA in both convergence speed and latency reduction (20% better on average).

Effect of Space Optimization. We employ *Region Discard*, *Tiny Feasible Space* and *Virtual Knob Extension* techniques to optimize the search space in Section 6.2. To evaluate the impact of space optimization, we compare the performance of GPTuner with and without the optimization, where GPTuner without space optimization is denoted as "GPTuner-naive-space". As shown in Figure 13c, GPTuner-naive-space struggles to search the vast search space, resulting in prolonged convergence and sub-optimal performance. In contrast to that, GPTuner yields superior performance since it is guided by the invaluable domain knowledge.

Effect of Coarse-to-Fine Bayesian Optimization Framework. In this part, we compare GPTuner with two stages against itself with only the first stage and the second stage, which are denoted as "GPTuner-1" and "GPTuner-2", respectively. As shown in Figure 13d, GPTuner exhibits the fastest convergence speed and the lowest latency. GPTuner-1 achieves fast convergence since it benefits from the Tiny Feasible Space defined in Section 6.2. However, it gets stuck in local-optimum due to the coarse granularity of the Tiny Feasible Space. While GPTuner-2 achieves lower latency than GPTuner-1 since it benefits from the *Region Discard* and *Virtual Knob Extension* techniques, it still requires a lot of iterations to explore the optimized space, which demonstrates the effectiveness of our bootstrap strategy discussed in Section 7.

Summary. Above ablation study reveals the benefits of all our technical contributions: (1) the proposed LLM-based pipeline is crucial to prepare high-quality Tuning Lake and Structured Knowledge, (2) the Knob Selection Mechanism prunes the space dimensionality efficiently, (3) the Range Optimization techniques consider each knob's unique semantics to optimize the value range of each dimensionallity, and (4) the Coarse-to-Fine BO framework explores the optimized search space under the guidance of both domain knowledge and runtime feedback efficiently.

8.6 Cost Analysis

There are two kinds of overheads in the context of DBMS knob tuning: (1) Initial Profiling Overhead, and (2) Runtime Overhead. Initial Profiling Overhead is the time required to collect training samples or to pre-train models before the real tuning process (e.g., OtterTune [52] demands "over 30k trials per DBMS" to collect training samples). Runtime Overhead is the execution time taken by an optimizer to generate the next configuration to evaluate per iteration, and does not include the evaluation time. In practice, runtime overhead (e.g., less than one second for SMAC [60]) is much less than the DBMS benchmark evaluation time (e.g., minutes to hours), and thus recent works focus on *sample efficiency* (i.e., the number of DBMS evaluation times it takes to reach a given level of performance improvement) rather than the optimization time [23, 60]. Since GPTuner introduces no extra runtime overhead compared to the BO-based optimizer, we only discuss the Initial Profiling Overhead next.

Given a specific DBMS product (e.g., PostgreSQL and MySQL), we only need to conduct domain knowledge-related stages for one time. We can use the built Structured Knowledge repeatedly in the future until there are major version updates for that DBMS, introducing a large number of changes of knob features (e.g., DBMS vendors significantly add new knobs, delete old knobs or even change the functions of existing knobs). Moreover, we present the number of tokens consumed, the monetary costs and the time required using three language models under three different number of target knobs in Table 3. Without any parallelism optimization, it takes "gpt-4-turbo" about 2101.6k tokens, 23.7 USD and 4 hours to finish the knowledge preparation and transformation stages for 150 knobs from PostgreSQL in total. Given that the built Tuning Lake and Structured Knowledge are reusable, and the costs of GPTuner are much lower than previous approaches (e.g., it takes more than three months to initialize the knowledge base for OtterTune and requires rebuilding from scratch for changes in hardware components and software versions [7, 52]), we conclude that the initial profiling overhead introduced in GPTuner is feasible and practical.

9 DISCUSSIONS AND FUTURE WORK

Online Tuning. We follow the tuning paradigm of most existing approaches by tuning a copy DBMS offline, and then deploy the satisfactory configuration on the DBMS in production environment. However, it takes considerable resources to make a copy DBMS for offline tuning. The lack of a practical online tuning paradigm comes from the fact that a configuration sampled from the large

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

configuration space could make the system crash, and such insecurity is disastrous to online services. In the future, we will explore the online deployment of GPTuner since domain knowledge plays a decisive role in eliminating dangerous configurations.

Optimizing other Data Processing Systems. In this work, we only utilize GPTuner to optimize DBMS performance. However, the idea of leveraging extensive domain knowledge to enhance the black-box optimization algorithm is generic, and can be utilized to enhance other data processing systems like Spark [57] and Flink [1]. In the future, we will identify the unique challenges when targeting at other systems, and explore the possibility of extending GPTuner to optimize these systems.

10 CONCLUSION

This paper introduces GPTuner, an automatic manual-reading database tuning system that leverages domain knowledge to enhance the knob tuning process. Extensive experiments are conducted to demonstrate the effectiveness of GPTuner and it outperforms existing state-of-the-art approaches.

REFERENCES

[1] [n.d.]. Apache flink: Stream and batch processing in a single engine. https://api.semanticscholar.org/CorpusID:3519738

[2] Toufique Ahmed and Premkumar Devanbu. 2023. Few-shot training LLMs for project-specific code-summarization. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22)*. Association for Computing Machinery, New York, NY, USA, Article 177, 5 pages. https://doi.org/10.1145/3551349.3559555

[3] Sihem Amer-Yahia, Angela Bonifati, Lei Chen, Guoliang Li, Kyuseok Shim, Jianliang Xu, and Xiaochun Yang. 2023. From Large Language Models to Databases and Back: A discussion on research and education. arXiv:2306.01388 [cs.DB]

[4] Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. 2014. OpenTuner: An Extensible Framework for Program Autotuning. In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation* (Edmonton, AB, Canada) *(PACT '14)*. Association for Computing Machinery, New York, NY, USA, 303–316. https://doi.org/10.1145/2628071.2628092

[5] Simran Arora, Avanika Narayan, Mayee F. Chen, Laurel Orr, Neel Guha, Kush Bhatia, Ines Chami, Frederic Sala, and Christopher Ré. 2022. Ask Me Anything: A simple strategy for prompting language models. arXiv:2210.02441 [cs.CL]

[6] Baoqing Cai, Yu Liu, Ce Zhang, Guangyu Zhang, Ke Zhou, Li Liu, Chunhua Li, Bin Cheng, Jie Yang, and Jiashu Xing. 2022. HUNTER: An Online Cloud Database Hybrid Tuning System for Personalized Requirements. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 646–659. https://doi.org/10.1145/3514221.3517882

[7] Stefano Cereda, Stefano Valladares, Paolo Cremonesi, and Stefano Doni. 2021. CGPTuner: A Contextual Gaussian Process Bandit Approach for the Automatic Tuning of IT Configurations under Varying Workload Conditions. *Proc. VLDB Endow.* 14, 8 (apr 2021), 1401–1413. https://doi.org/10.14778/3457390.3457404

[8] Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. 2023. Binding Language Models in Symbolic Languages. arXiv:2210.02875 [cs.CL]

[9] Benoît Dageville and Mohamed Zait. 2002. SQL Memory Management in Oracle9i. In *Proceedings of the 28th International Conference on Very Large Data Bases* (Hong Kong, China) *(VLDB '02)*. VLDB Endowment, 962–973.

[10] Damai Dai, Yutao Sun, Li Dong, Yaru Hao, Shuming Ma, Zhifang Sui, and Furu Wei. 2023. Why Can GPT Learn In-Context? Language Models Secretly Perform Gradient Descent as Meta-Optimizers. In *Findings of the Association for Computational Linguistics: ACL 2023*. Association for Computational Linguistics, Toronto, Canada, 4005–4019. https://doi.org/10.18653/v1/2023.findings-acl.247

[11] Xiang Deng, Prashant Shiralkar, Colin Lockard, Binxuan Huang, and Huan Sun. 2022. Dom-lm: Learning generalizable representations for html documents. *arXiv preprint arXiv:2201.10608* (2022).

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[13] Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudré-Mauroux. 2013. OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases. *PVLDB* 7, 4 (2013), 277–288. http://www.vldb.org/pvldb/vol7/p277-difallah.pdf

[14] Songyun Duan, Vamsidhar Thummala, and Shivnath Babu. 2009. Tuning Database Configuration Parameters with ITuned. *Proc. VLDB Endow.* 2, 1 (aug 2009), 1246–1257. https://doi.org/10.14778/1687627.1687767

[15] EDB. 2023. https://www.enterprisedb.com/blog/tuning-maxwalsize-postgresql

[16] Matthias Feurer and Frank Hutter. 2019. Hyperparameter optimization. *Automated machine learning: Methods, systems, challenges* (2019), 3–33.

[17] Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. 2015. Initializing Bayesian Hyperparameter Optimization via Meta-Learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence* (Austin, Texas) *(AAAI'15)*. AAAI Press, 1128–1135.

[18] HackerNews. 2023. https://news.ycombinator.com/item?id=28869509

[19] Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-Learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence* (Phoenix, Arizona) *(AAAI'16)*. AAAI Press, 2094–2100.

[20] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2018. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[21] Zhengbao Jiang, Yi Mao, Pengcheng He, Graham Neubig, and Weizhu Chen. 2022. OmniTab: Pretraining with Natural and Synthetic Data for Few-shot Table-based Question Answering. arXiv:2207.03637 [cs.CL]

[22] Konstantinos Kanellis, Ramnatthan Alagappan, and Shivaram Venkataraman. 2020. Too Many Knobs to Tune? Towards Faster Database Tuning by Pre-Selecting Important Knobs. In *Proceedings of the 12th USENIX Conference on Hot Topics in Storage and File Systems (HotStorage'20)*. USENIX Association, USA, Article 16, 1 pages.

[23] Konstantinos Kanellis, Cong Ding, Brian Kroth, Andreas Müller, Carlo Curino, and Shivaram Venkataraman. 2022. LlamaTune: Sample-Efficient DBMS Configuration Tuning. *Proc. VLDB Endow.* 15, 11 (jul 2022), 2953–2965. https://doi.org/10.14778/3551793.3551844

[24] Mayuresh Kunjir and Shivnath Babu. 2020. Black or White? How to Develop an AutoTuner for Memory-Based Analytics. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 1667–1683. https://doi.org/10.1145/3318464.3380591

[25] Eva Kwan. 2002. Automatic Configuration for IBM ® DB 2 Universal Database TM Compressing years of performance tuning experience into seconds of execution. https://api.semanticscholar.org/CorpusID:15267980

[26] Guoliang Li, Xuanhe Zhou, Shifu Li, and Bo Gao. 2019. QTune: A Query-Aware Database Tuning System with Deep Reinforcement Learning. *Proc. VLDB Endow.* 12, 12 (aug 2019), 2118–2130. https://doi.org/10.14778/3352063.3352129

[27] Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. 2022. On the advance of making language models better reasoners. *arXiv preprint arXiv:2206.02336* (2022).

[28] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. 2022. SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. *Journal of Machine Learning Research* 23, 54 (2022), 1–9. http://jmlr.org/papers/v23/21-0888.html

[29] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. 2022. SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. *Journal of Machine Learning Research* 23, 54 (2022), 1–9. http://jmlr.org/papers/v23/21-0888.html

[30] Colin Lockard, Prashant Shiralkar, and Xin Luna Dong. 2019. Openceres: When open information extraction meets the semi-structured web. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 3047–3056.

[31] Colin Lockard, Prashant Shiralkar, Xin Luna Dong, and Hannaneh Hajishirzi. 2020. Zeroshotceres: Zero-shot relation extraction from semi-structured webpages. *arXiv preprint arXiv:2005.07105* (2020).

[32] Zheheng Luo, Qianqian Xie, and Sophia Ananiadou. 2023. ChatGPT as a Factual Inconsistency Evaluator for Text Summarization. arXiv:2303.15621 [cs.CL]

[33] Michael D. McKay. 1992. Latin Hypercube Sampling as a Tool in Uncertainty Analysis of Computer Models. In *Proceedings of the 24th Conference on Winter Simulation* (Arlington, Virginia, USA) *(WSC '92)*. Association for Computing Machinery, New York, NY, USA, 557–564. https://doi.org/10.1145/167293.167637

[34] metis. 2023. https://www.metisdata.io/blog/postgresql-on-steroids-how-to-ace-your-database-configuration

[35] Avanika Narayan, Ines Chami, Laurel Orr, Simran Arora, and Christopher Ré. 2022. Can Foundation Models Wrangle Your Data? arXiv:2205.09911 [cs.LG]

[36] Fatemeh Nargesian, Erkang Zhu, Renée J Miller, Ken Q Pu, and Patricia C Arocena. 2019. Data lake management: challenges and opportunities. *Proceedings of the VLDB Endowment* 12, 12 (2019), 1986–1989.

[37] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[38] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd C Mowry, Matthew Perron, Ian Quah, et al. 2017. Self-Driving Database Management Systems.. In *CIDR*, Vol. 4. 1.

[39] Andrew Pavlo, Matthew Butrovich, Lin Ma, Prashanth Menon, Wan Shen Lim, Dana Van Aken, and William Zhang. 2021. Make Your Database System Dream of Electric Sheep: Towards Self-Driving Operation. *Proc. VLDB Endow.* 14, 12 (jul 2021), 3211–3221. https://doi.org/10.14778/3476311.3476411

[40] PostgreSQL. 2023. https://www.postgresql.org/docs/current/runtime-config-resource.html

[41] PostgreSQL. 2023. https://www.postgresql.org/docs/current/index.html

[42] PostgresqlCO.NF. 2023. https://postgresqlco.nf/

[43] Omer Sagi and Lior Rokach. 2018. Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8, 4 (2018), e1249.

[44] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. arXiv:2109.05093 [cs.CL]

[45] Jaeho Shin, Sen Wu, Feiran Wang, Christopher De Sa, Ce Zhang, and Christopher Ré. 2015. Incremental knowledge base construction using deepdive. In *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, Vol. 8. NIH Public Access, 1310.

[46] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems* 25 (2012).

[47] David G. Sullivan, Margo I. Seltzer, and Avi Pfeffer. 2004. Using Probabilistic Reasoning to Automate Software Tuning. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems* (New York, NY, USA) *(SIGMETRICS '04/Performance '04)*. Association for Computing Machinery, New York, NY, USA, 404–405. https://doi.org/10.1145/1005686.1005739

[48] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Chicago, Illinois, USA) *(KDD '13)*. Association for Computing Machinery, New York, NY, USA, 847–855. https://doi.org/10.1145/2487575.2487629

[49] Immanuel Trummer. 2021. Can deep neural networks predict data correlations from column names? *arXiv preprint arXiv:2107.04553* (2021).

[50] Immanuel Trummer. 2022. DB-BERT: A Database Tuning Tool That "Reads the Manual". In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 190–203. https://doi.org/10.1145/3514221.3517843

[51] Immanuel Trummer. 2023. Demonstrating GPT-DB: Generating Query-Specific and Customizable Code for SQL Processing with GPT-4. *Proceedings of the VLDB Endowment* 16, 12 (2023), 4098–4101.

[52] Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. 2017. Automatic Database Management System Tuning Through Large-Scale Machine Learning. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 1009–1024. https://doi.org/10.1145/3035918.3064029

[53] Dana Van Aken, Dongsheng Yang, Sebastien Brillard, Ari Fiorino, Bohan Zhang, Christian Bilien, and Andrew Pavlo. 2021. An Inquiry into Machine Learning-Based Automatic Configuration Tuning Services on Real-World Database Management Systems. *Proc. VLDB Endow.* 14, 7 (mar 2021), 1241–1253. https://doi.org/10.14778/3450980.3450992

[54] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, Online, 38–45. https://doi.org/10.18653/v1/2020.emnlp-demos.6

[55] Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023. Large Language Models are Versatile Decomposers: Decompose Evidence and Questions for Table-based Reasoning. arXiv:2301.13808 [cs.CL]

[56] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2019. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. arXiv:1809.08887 [cs.CL]

[57] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. 2016. Apache Spark: a unified engine for big data processing. *Commun. ACM* 59, 11 (oct 2016), 56–65. https://doi.org/10.1145/2934664

[58] Bohan Zhang, Dana Van Aken, Justin Wang, Tao Dai, Shuli Jiang, Jacky Lao, Siyuan Sheng, Andrew Pavlo, and Geoffrey J. Gordon. 2018. A Demonstration of the Ottertune Automatic Database Management System Tuning Service. *Proc. VLDB Endow.* 11, 12 (aug 2018), 1910–1913. https://doi.org/10.14778/3229863.3236222

[59] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, Minwei Ran, and Zekang Li. 2019. An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 415–432. https://doi.org/10.1145/3299869.3300085

[60] Xinyi Zhang, Zhuo Chang, Yang Li, Hong Wu, Jian Tan, Feifei Li, and Bin Cui. 2022. Facilitating Database Tuning with Hyper-Parameter Optimization: A Comprehensive Experimental Evaluation. *Proc. VLDB Endow.* 15, 9 (may 2022), 1808–1821. https://doi.org/10.14778/3538598.3538604

[61] Xinyi Zhang, Hong Wu, Zhuo Chang, Shuowei Jin, Jian Tan, Feifei Li, Tieying Zhang, and Bin Cui. 2021. ResTune: Resource Oriented Tuning Boosted by Meta-Learning for Cloud Databases. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 2102–2114. https://doi.org/10.1145/3448016.3457291

[62] Xinyi Zhang, Hong Wu, Yang Li, Jian Tan, Feifei Li, and Bin Cui. 2022. Towards Dynamic and Safe Configuration Tuning for Cloud Databases. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 631–645. https://doi.org/10.1145/3514221.3526176

[63] Yunjia Zhang, Avrilia Floratou, Joyce Cahoon, Subru Krishnan, Andreas C Müller, Dalitso Banda, Fotis Psallidas, and Jignesh M Patel. 2023. Schema Matching using Pre-Trained Language Models. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 1558–1571.

[64] Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M. Patel. 2023. ReAcTable: Enhancing ReAct for Table Question Answering. arXiv:2310.00815 [cs.DB]

[65] Xinyang Zhao, Xuanhe Zhou, and Guoliang Li. 2023. Automatic Database Knob Tuning: A Survey. *IEEE Transactions on Knowledge and Data Engineering* (2023), 1–20. https://doi.org/10.1109/TKDE.2023.3266893

[66] Xuanhe Zhou, Guoliang Li, and Zhiyuan Liu. 2023. LLM As DBA. arXiv:2308.05481 [cs.DB]

[67] Yuqing Zhu, Jianxun Liu, Mengying Guo, Yungang Bao, Wenlong Ma, Zhuoyue Liu, Kunpeng Song, and Yingchun Yang. 2017. BestConfig: Tapping the Performance Potential of Systems via Automatic Configuration Tuning. In *Proceedings of the 2017 Symposium on Cloud Computing* (Santa Clara, California) *(SoCC '17)*. Association for Computing Machinery, New York, NY, USA, 338–350. https://doi.org/10.1145/3127479.3128605