GPTuner: An LLM-Based Database Tuning System

Jiale Lao
Sichuan University
solidlao.jiale@gmail.com

Yibo Wang
Sichuan University
wangyibo.cs@gmail.com

Yufei Li
Sichuan University
liyufeievangeline@gmail.com

Jianping Wang
Northwest Normal University
2022222119@nwnu.edu.cn

Yunjia Zhang
University of Wisconsin-Madison
yunjia@cs.wisc.edu

Zhiyuan Cheng
Purdue University
cheng443@purdue.edu

Wanghu Chen
Northwest Normal University
chenwh@nwnu.edu.cn

Mingjie Tang*
Sichuan University
tangrock@gmail.com

Jianguo Wang
Purdue University
csjgwang@purdue.edu

ABSTRACT

Selecting appropriate values for the configurable knobs of Database Management Systems (DBMS) is essential to improve performance. But because the complexity of this task has surpassed the abilities of even the best human experts, the database community turns to machine learning (ML)-based automatic tuning systems. However, these systems still incur significant tuning costs or only yield suboptimal performance, attributable to their overly high reliance on black-box optimization and the lack of integration with domain knowledge, such as DBMS manuals and forum discussions. Hence, we propose GPTuner, a manual-reading database tuning system that extensively leverages domain knowledge to automatically optimize the 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, training-free knob selection strategy, (2) develop a search space optimization technique considering the value range of each knob, (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) and DBMS (PostgreSQL and MySQL). Compared to state-of-the-art methods, GPTuner identifies better configurations in **16x** less time on average. Moreover, GPTuner achieves up to **30%** performance improvement over the **best-performing** alternative.

* The corresponding author.

1. INTRODUCTION

Database Management Systems (DBMS) expose hundreds of configurable parameters (i.e., knobs) that control their runtime behavior [32]. Selecting appropriate values for these knobs is essential to optimizing system performance [42]. Given the large number of knobs with continuous or categorical domains, database administrators (DBAs) face significant challenges in identifying effective configurations, especially in the cloud environment where workloads and underlying physical configurations can vary significantly over time and across different DBMS instances [33].

To reduce manual tuning efforts of DBAs, state-of-the-art approaches automate the knob tuning via Machine Learning (ML) techniques, including Bayesian Optimization [18, 41, 45, 13, 46, 22] and Reinforcement Learning [43, 12, 38]. 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.

Although these methods possess the potential to eventually identify well-performing configurations, they still incur significant tuning costs [21, 22]. Previous studies [12, 45, 44] have revealed that state-of-the-art systems still require hundreds to thousands of iterations to find an ideal configuration, with each iteration taking minutes or more to execute the target workload. Such high tuning costs stem from their inefficiency in handling: (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 [12, 21, 13, 18, 43, 49], sacrificing the flexibility to choose workload-relevant knobs, or execute workloads numerous times to identify important knobs [44, 41, 20], which is resource-intensive. Regarding the second difficulty, most approaches use the default value ranges provided by DBMS vendors [18, 41, 45, 13, 46, 22, 43, 12]. However, the default value ranges are excessively broad for flexibility, which complicates the tuning process and introduces the risk of system crashes [33, 42].

In contrast to ML-based tuning approaches that adjust DBMS solely based on performance statistics, human DBAs often rely on domain knowledge (DBMS manuals and forum discussions). Unlike performance statistics, external domain knowledge directly reveals tuning hints, including important knobs and typical value ranges for each knob.

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

First, there are discussions on which knobs significantly impact DBMS performance. In web forums like *Hacker News*, it is widely discussed 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 [3]. 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 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 $\times 10^{308}$] 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 capable of leveraging domain knowledge. However, this is non-trivial for the following challenges. *C1. It is challenging to unify a structured view of the heterogeneous domain knowledge.* Domain knowledge typically comes in the form of DBMS manuals and web forum discussions, which can be in various formats. To leverage such knowledge, we need to transform 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 [30]. Existing approaches cannot meet the trade-off between cost and quality. They either demand domain-specific training [34], which is more complicated in our scenario since it requires expert knowledge to annotate DBMS knowledge, or they rely on strong assumptions that lack flexibility (e.g., focusing on specific document format) [16]. *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 and Reinforcement Learning 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 [15].

To address the challenges above, we propose GPTuner, a manual-reading database tuning system that extensively leverages domain knowledge to automatically 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, we utilize the self-evaluation and self-correction ability of LLM to incorporate two error correction mechanisms in the pipeline. Regarding challenge C2, we use structured knowledge to optimize the search space and enhance the space exploration process. Before the tuning process, we optimize the search space in terms of the space dimensionality and the values in each dimension. For dimensionality, we propose a workload-aware and training-free knob selection strategy by leveraging the text analysis ability of LLM to simulate the knob selection process of DBAs, considering the characteristics of DBMS, workloads, specific query and knob dependencies. We further optimize the values in each dimension by using structured knowledge to discard meaningless regions, highlight promising space and take special situations into consideration. Finally, we develop a novel Coarse-to-Fine Bayesian Optimization Framework to explore the optimized search space. Since it is non-trivial to reduce the size of search space while still retaining the potential for optimal results [19], we seek help from domain knowledge to carefully design two search spaces of different granularity. The two spaces are explored sequentially by BO from coarse to fine granularity, with a bootstrap technique to serve as a bridging mechanism. This balances well between the efficiency of coarse-grained search and the thoroughness of fine-grained search.

We compare GPTuner against state-of-the-art approaches. We consider different benchmarks (TPC-C and TPC-H), metrics (throughput and latency), and DBMS (PostgreSQL and MySQL). GPTuner identifies better configurations in **16x** less time on average, achieving up to **30%** performance improvement over the **best-performing** alternative. In addition to effectiveness, we manually prepare and open-source two datasets for LLM evaluation [2], and conduct experiments to evaluate GPTuner's robustness and scalability. Moreover, we analyze the overheads introduced in GPTuner using different language models, and open-source the built domain knowledge to enable the further study within the database community.

2. BACKGROUND AND RELATED WORK

2.1 Database Knob Tuning

The DBMS 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 has proven to be NP-hard [36] and three kinds of approaches are proposed.

- **Heuristic-based.** Rule-based methods rely on manually created rules to explore the space in a predefined way [15].

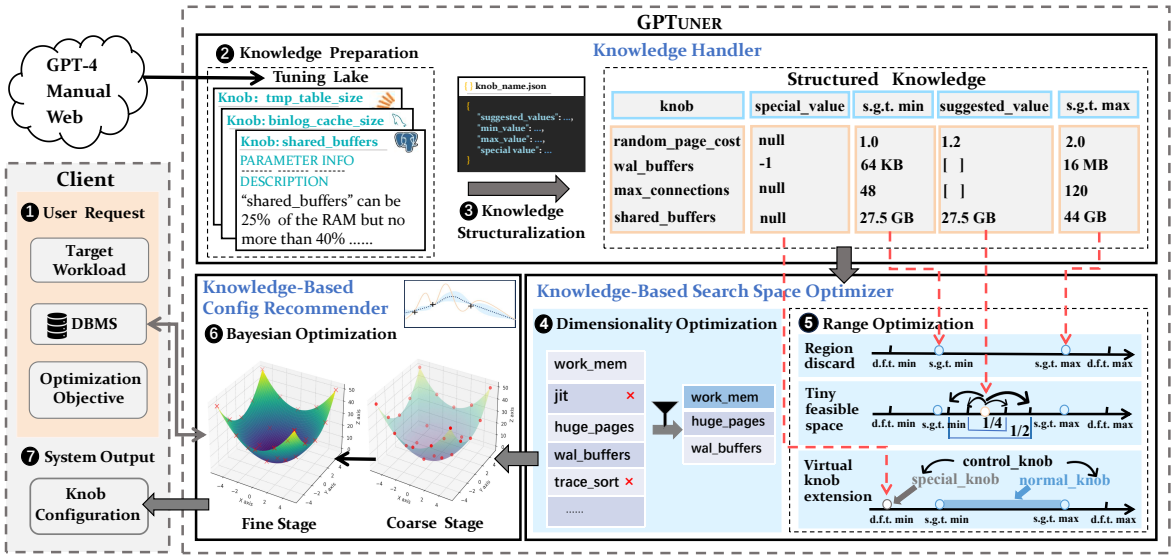

Figure 1: System Overview of GPTuner

Search-based methods explore the space following several heuristics (e.g., avoid revisiting the explored regions and explore nearby regions to improve current optimum) [49, 10].

- **BO-based.** BO-based methods [18, 41, 45, 13, 46, 22] follow the generic BO framework to search the space by (1) fitting a *surrogate model* to map the relation between the knob configurations and the DBMS performance, (2) selecting the next configuration that maximizes an *acquisition function*.
- **RL-based.** RL-based methods explore the space in a trial-and-error manner. The essence is to balance between exploring unexplored space and exploiting known regions, achieved by the interactions between an agent (e.g., neural networks) and the environment (e.g., DBMS) [43, 12, 40].

Performance Comparison. According to [44], a BO-based method, Sequential Model-based Algorithm Configuration (SMAC) [26] yields the best performance since it is efficient in modeling the heterogeneous search space. *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 for Database

Natural Language Processing has seen great progress with the advent of LLM, where GPT-4 [31] has demonstrated great prowess across a wide range of tasks. This impact further extends beyond the scope of NLP to the database community, experiencing a surge in the adoption of LLM to enhance DBMS, including data profiling [47, 37], code generation [39, 14], and table-based question answering [48]. When using LLM, there are three typical choices: (1) training a model from scratch, (2) fine-tuning an existing model, and (3) using a pre-trained model without parameter modifications. The first two options require a relatively large amount of resources, including hardware resources and training data. Given the impressive in-context learning ability of GPT-4, we have chosen the third option and use GPT-4 as our default model unless noted otherwise.

We are aware of only one work DB-BERT [38] 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 suboptimal performance since it utilizes the knowledge narrowly and suffers from the inadequate exploration of search space. GPTuner tackles these limitations and thus achieves faster convergence and better performance. Please refer to our research paper [23] for a comprehensive comparison.

3. OVERVIEW OF GPTUNER

Workflow. GPTuner is a manual-reading database tuning system to suggest satisfactory knob configurations with reduced tuning costs. Figure 1 presents the tuning workflow. 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 which are discussed in the following sections.

4. KNOWLEDGE HANDLER

4.1 Knowledge Preparation

Knowledge Preparation is to collect tuning knowledge from various sources, filter out noisy contents, summarize the legal parts and make sure the summarization is factual consistent with source contents. We provide a running example in our technical report [7] and a demonstration [24] for ease of understanding. The ultimate 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).*

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 [9], 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. We handle the case that GPT gives nonsense suggestions in the next step.

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). Moreover, we give a few examples in the prompt to utilize the in-context learning ability of LLM. LLM evaluates whether the tuning knowledge conflicts with the system view and we discard any knowledge that does conflict.

*Step 3: Summarizing knowledge from various sources.*There can be multiple tuning knowledge for a knob. While such tuning guidance obeys 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.

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 [28], 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 no error is detected.

Robustness Study. It is important to note that hallucination still remains a challenge in the NLP field, and thus we propose two error correction mechanisms to minimize such impact as much as possible. To quantify the reliability, we manually prepare and open-source two datasets, and conduct experiments. 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 7.4.

4.2 Knowledge Transformation

Knowledge transformation is to build a structured view of the *Tuning Lake* such that it can be utilized by ML.

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 a set of attributes $A = \{a_1, a_2, \dots, a_n\}$ (e.g., $a_1 = suggested\_values$) with corresponding values $V = \{v_1, v_2, \dots, v_n\}$ (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*. First, 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. Second, 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 values 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 models since they lead to distinct behaviours of DBMS. Thus we use *special_value* to handle such special situations. How to use these values are detailed in Section 5.2.

• **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 [25]). Since it is useful to acquire a more reliable result by aggregating multiple imperfect but effective results [11], we develop a *Prompt Ensemble Algorithm* to determine the attribute values effectively. 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.

Step 1: we model the transformation 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 to be discussed in Section 5.2. 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 $\{knowledge\}$).

Step 2: we vary the prompts by changing the examples provided for few-shots learning. We manually prepare K examples and store them in an *Example Pool*. Then we randomly sample n examples for each prompt ($n \leq K$). Specifically, we manually prepare 10 examples ($K = 10$) since 10 examples are empirically good enough to meet the diversity requirement of ensemble algorithms [8]. 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, resulting in longer processing time and higher monetary costs. Moreover, since too many examples might lead to long-context errors, we use $n = 3$ as suggested [11].

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 $\mathcal{S}$.

5. SEARCH SPACE OPTIMIZER

5.1 Dimensionality Optimization

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 v14.9 has 346 knobs), not all of them significantly impact DBMS performance, and it is infeasible to consider all knobs due to the curse of dimensionality. Recent studies have shown that tuning a small number of knobs is enough to yield near-optimal performance while significantly reducing tuning costs [42, 20], thus we only select important knobs to tune. Existing methods rely on ML-based algorithms to select important knobs and this requires hundreds to thousands of evaluations on DBMS under different workloads and configurations [20, 44, 41]. Differently, DBAs seek help from manuals to select which knobs are worth tuning and this yields better performance [42]. However, it takes significant burden for DBAs to handle hundreds of knobs. Therefore, we prompt LLM to simulate DBA's empirical judgment to achieve a workload-aware and training-free approach, considering following 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`" from PostgreSQL, "`innodb_buffer_pool_size`" and "`innodb_log_file_size`" from MySQL [42]. More importantly, we find such wisdom is included in the corpus of GPT-4 [31] and we can extract it by prompting it to recommend knobs based on the DBMS product.

(2) Workload-Level selects knobs based on workload types. 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 I/O-intensive OLTP workload, where write operations compete for the limited disk I/O, we can tune disk-related knobs like "`effective_io_concurrency`". For OLAP queries, we can tune planning and resource-related knobs to handle their complex structure and resource-intensive nature.

(3) Query-Level selects knobs based on the query bottleneck. DBAs always delve into the execution plan of the time-consuming and frequently executed queries, diagnosing performance bottlenecks and focusing on related knobs. Given the analysis ability of LLM, we can include the execution plan in the prompt such that LLM can choose the bottleneck-aware knobs. 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`" to control 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 [42]. However, if the given knob set contains important knobs but excludes the interdependent knobs, such ability is wasted. Since many dependencies are explicitly mentioned in the manuals, we can prompt 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`" [5], indicating that we should consider the two knobs at the same time.

Based on the above four factors, we develop an **LLM-based Knob Selection Algorithm**. It aspires to utilize LLM to select important knobs from the complete knob set of a DBMS tailored to a specific workload. The algorithm starts by preparing a configurable set of knobs and excluding knobs related to debugging, security and path-setting. Next, it selects knobs from the four levels. In System Level, DBMS product is provided and knobs known to hugely influence its performance are identified. For the Workload Level, we provide LLM with the type of the workload (OLTP or OLAP) and the optimization target (throughput or latency) for further selection. Query Level Selection retrieves the execution plan of each query, and utilizes LLM to analyze plans and select knobs by diagnosing performance bottlenecks and identifying the related knobs. Finally, given the already selected knobs, we leverage LLM to replenish interdependent knobs.

5.2 Range Optimization

Given each knob's unique semantics and associated tuning knowledge, we optimize the value range of 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 search algorithm is likely to sample very large values. However this will lead to poor performance since it is suggested to set it to "$1.x$" if the disk has a random access profile similar to that of SSDs [1]. 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 DBMS crash. For resource-related knobs, setting a value that exceeds the available resources could prevent the DBMS from starting (cannot set memory-related knobs to values higher than the available RAM). Finally, the recommended range $[min\_value, max\_value]$ is much more narrow than the range provided by DBMS.

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:

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

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 [6]. Given the maximum factor 4 used in DB-BERT [38], the maximum value to try is $90 \times 4 = 360$ and this is much smaller than 86400, meaning a lot of promising values are ignored. Thus we 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 number of values for knobs is significantly reduced, and *Feasible* indicates the values are promising.

Virtual Knob Extension. This extension is to handle knobs that use special values to do something different from what the knobs normally do [33]. For example, "`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, making "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 and DBMS performance will be modeled to degrade as the knob value converges to zero [33]. Thus, we develop a *Virtual Knob Extension*. Firstly, we utilize *Structured Knowledge* to select which knobs have special values. Such information is explicitly provided in official documents [5] and thus can be extracted by *Knowledge Handler* (Section 4.1). Secondly, we add "virtual knobs" (*control_knob*, *normal_knob* and *special_knob*) for these knobs. *control_knob* is a knob with a value of zero or one to determine which value range will be used. *normal_knob* and *special_knob* represent the normal and the special value range of the original knob, respectively. Specifically, when *control_knob* is set to 0, the normal value range of the knob is considered. If it is set to 1, then the special value range of the knob is used. So in every tuning iteration, based on the value of *control_knob*, only one of *normal_knob* and *special_knob* will be activated. Above technique makes the special values of these knobs to be considered by optimizers.

6. CONFIGURATION RECOMMENDER

Basic Idea of Bayesian Optimization. BO is a sequential model-based algorithm [35] consisting of two components: the *surrogate model* and the *acquisition function*. The model takes a configuration as input and predicts DBMS performance. The acquisition function evaluates the utility of candidate configurations (e.g., Expected Improvement (EI)), and we choose the next configuration with the maximum utility. After the surrogate model is initialized, BO works iteratively: (1) selecting the next configuration by maximizing the acquisition function, (2) evaluating the configuration on DBMS and updating the surrogate model with the new observation. This is repeated until it runs out of resources.

Limitation of Bayesian Optimization. The key to the success of BO 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 [44]. However, the number of samples required to have sufficient confidence in predictions could still be significant [22, 21]. 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.

Algorithm 1: Coarse-to-Fine BO Framework. Blue underlined text highlights differences to original BO.

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 $\mathcal{C}$; Initial Number n.
Output: Knob Configuration $\mathcal{X}$.

Generate Tiny Feasible Space $\mathcal{T}$ from $\mathcal{S}$;
Generate n samples $p_i \in \mathcal{T}$ with space-filling design (LHS);
Evaluate samples on $\mathcal{D}$ to get performance y_i;
Update $\mathcal{M}$ with observations $\{(p_i, y_i)\}$;
Coarse Stage:
for $i = 1$ *to* $\mathcal{C}$ **do**
  $\vec{x_i} \leftarrow \arg\max_{\vec{x} \in \mathcal{T}} \mathcal{A}(\vec{x})$;
  Evaluate $\vec{x_i}$ on $\mathcal{D}$ to get performance y_i;
  Update $\mathcal{M}$ with $(\vec{x_i},\ y_i)$;
end
Fine Stage:
Reuse the surrogate model $\mathcal{M}$ from Coarse Stage;
Apply *Region Discard* on $\mathcal{P}$ to get $\mathcal{P}'$;
Apply *Virtual Knob Extension* on $\mathcal{P}'$ to get $\mathcal{P}''$;
while *not stopping condition* **do**
  $\vec{x_i} \leftarrow \arg\max_{\vec{x} \in \mathcal{P}''} \mathcal{A}(\vec{x})$;
  Evaluate $\vec{x_i}$ on $\mathcal{D}$ to get performance y_i;
  Update $\mathcal{M}$ with $(\vec{x_i},\ y_i)$;
end
$\mathcal{X} \leftarrow \arg\max_{\vec{x_i}} y_i$;
return $\mathcal{X}$;

Coarse-to-Fine Bayesian Optimization Framework. The intuition is to take advantages of the efficiency of coarse-grained search (aim to acquire non-optimal but effective solutions with low expenses) and the thoroughness of fine-grained search (aim to find the optimal solutions by exploring the space thoroughly). Algorithm 1 describes the workflow.

Coarse-grained Stage. In the first stage, we only explore part of the whole space. A widely adopted approach is to discretize the value ranges of parameters evenly for coarse-grained search. However, this will lose too many promising solutions because such a space reduction technique is inherently imprecise and non-adaptive. We instead seek help from knowledge to reduce the space while still retaining the potential for optimal results. We explore the *Tiny Feasible Space* (Line 1) defined in Section 5.2, which is small but reliable because it comes from manuals. Following previous works [18, 21, 44], we initialize the surrogate model with ten samples ($n = 10$) generated by Latin Hypercube Sampling (LHS) [29], 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 $\mathcal{C}$ iterations (Line 5-8). After the coarse-grained stage, we try out $n + \mathcal{C}$ 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 (Line 13-17). However, this process could be exhausting, 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). Please refer to our technical report [7] for a detailed discussion on our bootstrap.

7. EXPERIMENTAL EVALUATION

We conduct experiments to evaluate the effectiveness, scalability, and robustness of GPTuner. We also analyze and present the costs, and use an ablation study to reveal the effect of each component. We present only parts of the results here, please refer to [23] for a complete evaluation.

7.1 Experimental Setup

Workloads. We use TPC-H (OLAP type) with scale factor 1 and TPC-C (OLTP type) with factor 200 as our benchmarks. TPC-C uses ten terminals with unlimited arrival rate and the implementations are from BenchBase [17].

Baselines. GPTuner is implemented with SMAC3 library [27] and uses OpenAI completion API [31]. We compare it with state-of-the-art methods: DDPG++ [42] is a RL-based approach, GP is a Gaussian Process-based optimizer used in iTuned [18] and OtterTune [41], SMAC [44] is the best-performing BO-based method with random forest as its surrogate, and DB-BERT is a database tuning tool [38] that uses BERT for text analysis to guide a RL algorithm.

Tuning Settings. We run experiments with PostgreSQL v14.9 and MySQL v8.0. Following previous works [43, 21, 44], all algorithms tune the same 60 and 40 knobs of PostgreSQL and MySQL respectively. 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 [44], we optimize throughput for OLTP workload and 95-th percentile latency for OLAP workload, reporting the median and quartiles of the best performances. Please refer to our technical report for more details on the tuning settings [7].

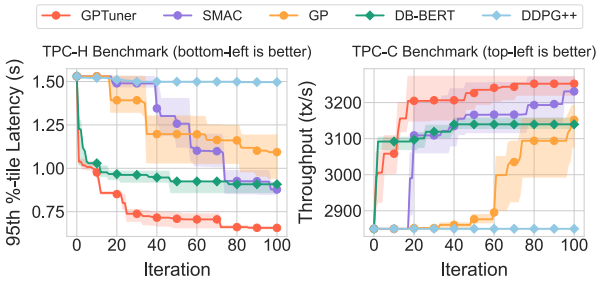

Figure 2: Best performance over iterations on PostgreSQL

7.2 Performance Comparison

Optimizing for PostgreSQL. As shown in Figure 2, 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. On average, GPTuner takes 92.6% less optimization time to achieve the best baseline performance.

Optimizing for MySQL. The results on MySQL are similar to that on PostgreSQL, please refer to [23] for more details.

7.3 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 3, compared to other methods, GPTuner finds better configurations in much fewer iterations in all 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 attribute to the fact that the complexity of modeling the relation between configurations and DBMS performance is higher in larger 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 [38].

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 a fixed set of knobs mentioned in the input documents, making it infeasible to manually control the number of target knobs [38]. As shown in Figure 5, GPTuner consistently achieves the best

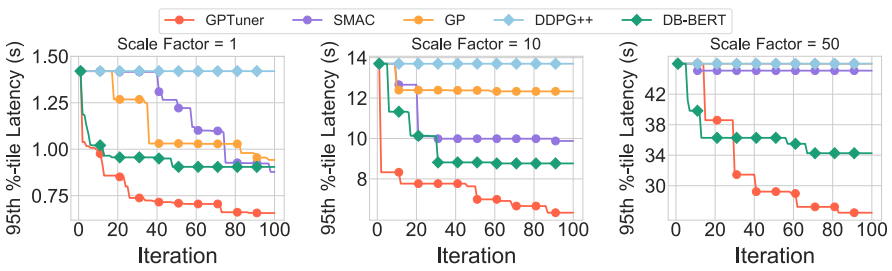

Figure 3: Effect of Database Size on Tuning Performance (bottom-left is better)

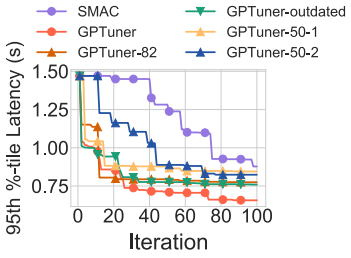

Figure 4: Effect of Knowledge Quality (bottom-left is better)

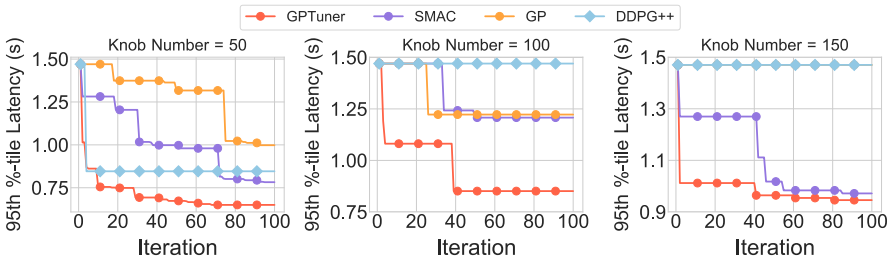

Figure 5: Effect of Space Dimensionality on Tuning Performance (bottom-left is better)

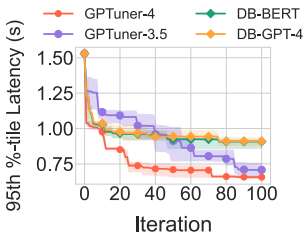

Figure 6: Effect of Language Models (bottom-left is better)

performance in all space sizes. While other methods perform well in low-dimensional case, their performance deteriorates in high-dimensional cases. This is because although the dimensionality is fixed for all methods, GPTuner still benefits from its Range Optimization discussed in Section 5.2.

7.4 Robustness Study

Effect of Domain Knowledge Quality. *(1) Outdated Knowledge.* We use PostgreSQL manual of version 9.1 as the knowledge input for GPTuner to tune PostgreSQL v14.9, which is denoted as "GPTuner-outdated". *(2) Inaccurate Knowledge.* Firstly, we use domain knowledge with an accuracy of 82% without using step 2 as a filter of step 1 ("GPTuner-82") and compare it to the original GPTuner. Next, among all the target knobs, we randomly select half of them and assign the factual inconsistent knowledge to them. This procedure is repeated five times and we present two representative results, which are denoted as "GPTuner-50-1" and "GPTuner-50-2", respectively.

In Figure 4, the higher the knowledge quality, the better the tuning performance. Note that outdated knowledge only affects tuning performance slightly, while inaccurate knowledge impacts tuning performance with different degrees. This relates to our motivation in Section 5.1, that only some knobs are crucial to optimize DBMS performance. GPTuner benefits a lot if the knowledge of these important knobs is correct. In production environment, we encourage users to double check such knowledge. Given that GPTuner under noisy knowledge is still better than the best optimizer without knowledge input (SMAC), we believe GPTuner is robust enough to learn from mistakes in its BO stage and achieve satisfactory performance optimization ultimately.

Effect of Different Language Models. We analyze the effect of LLM 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" ("GPTuner-3.5" and "GPTuner-4"). As shown in Figure 6, 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, and GPTuner significantly benefits from GPT-4, demonstrating its ability to leverage more sophisticated LLM.

8. CONCLUSION AND OUTLOOK

We present GPTuner, a manual-reading database tuning system that leverages domain knowledge to enhance the knob tuning process. Extensive experiments have demonstrated its effectiveness, scalability, and robustness.

LLM for DBMS Tuning. We are exploring opportunities to further improve the tuning. First, GPTuner applies domain knowledge to individual knobs independently, overlooking interactions between multiple knobs. Second, incorporating domain knowledge to exclude unsafe configurations can enhance reliability and enable support for online tuning. This represents an exciting opportunity to bridge offline knob tuning with online query optimization. Third, it is worth investigating whether the tuning of Vector Databases can benefit from lessons learned in tuning Relational Databases. Finally, using LLM to directly recommend configurations offers promising solutions to cold-start issues in DBMS tuning.

LLM for Databases. LLM and future database systems have the potential to manage and query all the world's data (text, image, video). Integrating LLM into the design of future systems introduces both opportunities and challenges. Possible areas include developing natural language query interface, designing end-to-end pipelines to provide text-to-results convenience, creating query optimizers that leverage LLM (e.g., SQL rewrite) and support multi-modal data, utilizing LLM for data discovery and processing, and rethinking fundamental data management principles for these advancements.

Acknowledgements

Mingjie Tang acknowledges the support of the National Science Foundation of China under Grant Number 92470204.

9. REFERENCES

[1] PostgresqlCO.NF. https://postgresqlco.nf/.

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

[3] HackerNews, 2021. https://news.ycombinator.com/item?id=28869509.

[4] 2023. https://www.postgresql.org/docs/current/runtime-config-resource.html.

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

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

[7] GPTuner: full version, 2024. https://github.com/SolidLao/GPTuner/blob/main/gptuner-technical-report.pdf.

[8] Toufique Ahmed and Premkumar Devanbu. Few-shot training llms for project-specific code-summarization. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ASE '22, New York, NY, USA, 2023. Association for Computing Machinery.

[9] Sihem Amer-Yahia, Angela Bonifati, Lei Chen, Guoliang Li, Kyuseok Shim, Jianliang Xu, and Xiaochun Yang. From large language models to databases and back: A discussion on research and education, 2023.

[10] Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. Opentuner: An extensible framework for program autotuning. In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*, PACT '14, page 303–316, New York, NY, USA, 2014. Association for Computing Machinery.

[11] Simran Arora, Avanika Narayan, Mayee F. Chen, Laurel Orr, Neel Guha, Kush Bhatia, Ines Chami, Frederic Sala, and Christopher Ré. Ask me anything: A simple strategy for prompting language models, 2022.

[12] Baoqing Cai, Yu Liu, Ce Zhang, Guangyu Zhang, Ke Zhou, Li Liu, Chunhua Li, Bin Cheng, Jie Yang, and Jiashu Xing. Hunter: An online cloud database hybrid tuning system for personalized requirements. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 646–659, New York, NY, USA, 2022. Association for Computing Machinery.

[13] Stefano Cereda, Stefano Valladares, Paolo Cremonesi, and Stefano Doni. Cgptuner: A contextual gaussian process bandit approach for the automatic tuning of it configurations under varying workload conditions. *Proc. VLDB Endow.*, 14(8):1401–1413, apr 2021.

[14] 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. Binding language models in symbolic languages, 2023.

[15] Benoît Dageville and Mohamed Zait. Sql memory management in oracle9i. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, page 962–973. VLDB Endowment, 2002.

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

[17] Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudré-Mauroux. Oltp-bench: An extensible testbed for benchmarking relational databases. *PVLDB*, 7(4):277–288, 2013.

[18] Songyun Duan, Vamsidhar Thummala, and Shivnath Babu. Tuning database configuration parameters with ituned. *Proc. VLDB Endow.*, 2(1):1246–1257, aug 2009.

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

[20] Konstantinos Kanellis, Ramnatthan Alagappan, and Shivaram Venkataraman. 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, USA, 2020. USENIX Association.

[21] Konstantinos Kanellis, Cong Ding, Brian Kroth, Andreas Müller, Carlo Curino, and Shivaram Venkataraman. Llamatune: Sample-efficient dbms configuration tuning. *Proc. VLDB Endow.*, 15(11):2953–2965, jul 2022.

[22] Mayuresh Kunjir and Shivnath Babu. 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*, SIGMOD '20, page 1667–1683, New York, NY, USA, 2020. Association for Computing Machinery.

[23] Jiale Lao, Yibo Wang, Yufei Li, Jianping Wang, Yunjia Zhang, Zhiyuan Cheng, Wanghu Chen, Mingjie Tang, and Jianguo Wang. Gptuner: A manual-reading database tuning system via gpt-guided bayesian optimization. *Proc. VLDB Endow.*, 17(8):1939–1952, May 2024.

[24] Jiale Lao, Yibo Wang, Yufei Li, Jianping Wang, Yunjia Zhang, Zhiyuan Cheng, Wanghu Chen, Yuanchun Zhou, Mingjie Tang, and Jianguo Wang. A demonstration of gptuner: A gpt-based manual-reading database tuning system. In *Companion of the 2024 International Conference on Management of Data*, SIGMOD/PODS '24, page 504–507, New York, NY, USA, 2024. Association for Computing Machinery.

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

[26] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022.

[27] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of*

Machine Learning Research, 23(54):1–9, 2022.

[28] Zheheng Luo, Qianqian Xie, and Sophia Ananiadou. Chatgpt as a factual inconsistency evaluator for text summarization, 2023.

[29] Michael D. McKay. Latin hypercube sampling as a tool in uncertainty analysis of computer models. In *Proceedings of the 24th Conference on Winter Simulation*, WSC '92, page 557–564, New York, NY, USA, 1992. Association for Computing Machinery.

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

[31] OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[32] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd C Mowry, Matthew Perron, Ian Quah, et al. Self-driving database management systems. In *CIDR*, volume 4, page 1, 2017.

[33] Andrew Pavlo, Matthew Butrovich, Lin Ma, Prashanth Menon, Wan Shen Lim, Dana Van Aken, and William Zhang. Make your database system dream of electric sheep: Towards self-driving operation. *Proc. VLDB Endow.*, 14(12):3211–3221, jul 2021.

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

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

[36] David G. Sullivan, Margo I. Seltzer, and Avi Pfeffer. Using probabilistic reasoning to automate software tuning. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '04/Performance '04, page 404–405, New York, NY, USA, 2004. Association for Computing Machinery.

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

[38] Immanuel Trummer. Db-bert: A database tuning tool that "reads the manual". In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 190–203, New York, NY, USA, 2022. Association for Computing Machinery.

[39] Immanuel Trummer. Demonstrating gpt-db: Generating query-specific and customizable code for sql processing with gpt-4. *Proceedings of the VLDB Endowment*, 16(12):4098–4101, 2023.

[40] Immanuel Trummer, Junxiong Wang, Ziyun Wei, Deepak Maram, Samuel Moseley, Saehan Jo, Joseph Antonakakis, and Ankush Rayabhari. Skinnerdb: Regret-bounded query evaluation via reinforcement learning. *ACM Trans. Database Syst.*, 46(3), sep 2021.

[41] Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, page 1009–1024, New York, NY, USA, 2017. Association for Computing Machinery.

[42] Dana Van Aken, Dongsheng Yang, Sebastien Brillard, Ari Fiorino, Bohan Zhang, Christian Bilien, and Andrew Pavlo. An inquiry into machine learning-based automatic configuration tuning services on real-world database management systems. *Proc. VLDB Endow.*, 14(7):1241–1253, mar 2021.

[43] 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. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, page 415–432, New York, NY, USA, 2019. Association for Computing Machinery.

[44] Xinyi Zhang, Zhuo Chang, Yang Li, Hong Wu, Jian Tan, Feifei Li, and Bin Cui. Facilitating database tuning with hyper-parameter optimization: A comprehensive experimental evaluation. *Proc. VLDB Endow.*, 15(9):1808–1821, may 2022.

[45] Xinyi Zhang, Hong Wu, Zhuo Chang, Shuowei Jin, Jian Tan, Feifei Li, Tieying Zhang, and Bin Cui. Restune: Resource oriented tuning boosted by meta-learning for cloud databases. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, page 2102–2114, New York, NY, USA, 2021. Association for Computing Machinery.

[46] Xinyi Zhang, Hong Wu, Yang Li, Jian Tan, Feifei Li, and Bin Cui. Towards dynamic and safe configuration tuning for cloud databases. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 631–645, New York, NY, USA, 2022. Association for Computing Machinery.

[47] Yunjia Zhang, Avrilia Floratou, Joyce Cahoon, Subru Krishnan, Andreas C Müller, Dalitso Banda, Fotis Psallidas, and Jignesh M Patel. Schema matching using pre-trained language models. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pages 1558–1571. IEEE, 2023.

[48] Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M. Patel. Reactable: Enhancing react for table question answering, 2023.

[49] Yuqing Zhu, Jianxun Liu, Mengying Guo, Yungang Bao, Wenlong Ma, Zhuoyue Liu, Kunpeng Song, and Yingchun Yang. Bestconfig: Tapping the performance potential of systems via automatic configuration tuning. In *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, page 338–350, New York, NY, USA, 2017. Association for Computing Machinery.