LLMTune: Accelerate Database Knob Tuning with Large Language Models

Xinmei Huang[1], Haoyang Li[1], Jing Zhang[1], Xinxin Zhao[1], Zhiming Yao[1], Yiyan Li[1], Zhuohao Yu[3], Tieying Zhang[2], Hong Chen[1], Cuiping Li[1]
[1] School of Information, Renmin University of China, [2] ByteDance, [3] Peking University, China
{huangxinmei,lihaoyang.cs,zhang-jing,zhaoxinxin798,yaojimmy2005,liyiyan,chong,licuiping}@ruc.edu.cn
zyu@stu.pku.edu.cn
zhangtiey@gmail.com

arXiv:2404.11581v1 [cs.AI] 17 Apr 2024

ABSTRACT

Database knob tuning is a critical challenge in the database community, aiming to optimize knob values (*i.e.*, configurations) to enhance database performance for specific workloads. Modern Database Management Systems (DBMS) often feature hundreds of tunable knobs, with each knob having continuous or discrete values, posing a significant challenge for database administrators (DBAs) to recommend an optimal configuration. Consequently, a range of machine learning-based (ML-based) tuning methods has been developed to automate this configuration process. Despite the introduction of various optimizers, practical applications have unveiled a new problem: these methods typically require numerous workload runs to achieve satisfactory performance, a process that is both time-consuming and resource-heavy. This inefficiency largely stems from the optimal configuration often being substantially different from the default setting, necessitating multiple iterations during tuning. Recognizing this, we argue that an effective starting point could significantly reduce redundant exploration in less efficient areas, thereby potentially speeding up the tuning process for the optimizers. Based on this assumption, *we introduce LLMTune, a large language model (LLM)-based configuration generator designed to produce an initial, high-quality configuration for new workload.* These generated configurations can then serve as the starting points for various base optimizers, accelerating their tuning processes. To obtain training data for LLMTune's supervised fine-tuning, we have devised a new automatic data generation framework capable of efficiently creating a large number of <workload, configuration> pairs. We have conducted thorough experiments to evaluate LLMTune's effectiveness with different workloads, such as TPC-H and JOB. In comparison to leading methods, LLMTune demonstrates a quicker ability to identify superior configurations. For instance, with the challenging TPC-H workload, our LLMTune achieves a significant **15.6x** speed-up ratio in finding the **best-performing** configurations.

PVLDB Reference Format:
Xinmei Huang[1], Haoyang Li[1], Jing Zhang[1], Xinxin Zhao[1], Zhiming Yao[1], Yiyan Li[1], Zhuohao Yu[3], Tieying Zhang[2], Hong Chen[1], Cuiping Li[1] .
LLMTune: Accelerate Database Knob Tuning with Large Language Models. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

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

1 INTRODUCTION

Performance optimization of database management systems (DBMS) is a complex yet critical task, with knob tuning serving as a central technique. Specifically, knob tuning involves adjusting various configuration parameters (*a.k.a.*, "knobs") within the DBMS to maximize the execution efficiency for a given workload. These knobs control aspects such as memory allocation, query optimization strategies, caching mechanisms, and concurrency settings. However, knob tuning is an NP-hard problem due to the presence of numerous knobs in modern DBMS, sometimes numbering in the hundreds [53]. This abundance of knobs results in an immense search space of possible configurations, presenting a significant challenge in identifying the optimal combination tailored to the specific workload.

In recent years, there has been considerable attention on automated knob tuning techniques, which aim to automatically adjust database configuration parameters through various intelligent algorithms, adapting to varying workloads and operational environments. As a result, these techniques could alleviate the burden on database administrators (DBAs) by reducing manual intervention. Generally, these base optimizers could be classified into two main categories [49]: Bayesian Optimization-based (BO-based) methods, such as iTuned [8] and SMAC [12], and Reinforcement Learning-based (RL-based) methods, such as CDBTune [48] and UDO [44].

Although these methods are good at finding suitable configurations, they usually require a large number of workload runs to achieve a satisfactory level of workload performance, which results in poor tuning efficiency. For instance, BO-based methods typically necessitate hundreds of iterations to model the distributions derived from configurations and their corresponding performances. In each iteration, the workload is executed under a specific configuration. On the other hand, RL-based methods usually need additional online training, which also involves hundreds of interactions with environments (*i.e.*, databases). Therefore, for real-world applications, it is crucial to reduce the workload runs needed to find a satisfactory configuration.

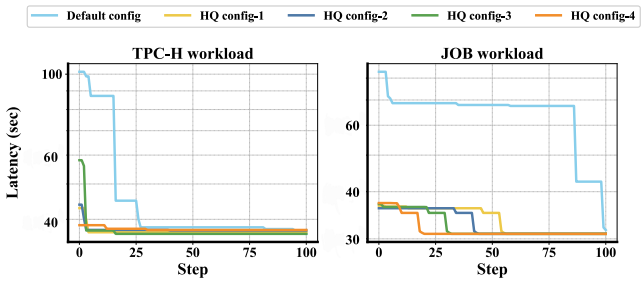

Figure 1: Illustration of the knob tuning processes with default and high-quality starting points of configurations, where "default config" indicates performing the knob tuning method HEBO from a default configuration, and "HQ config" means performing HEBO starting from high-quality configurations. The results demonstrate that a high-quality initial configuration can significantly enhance tuning efficiency and concurrently lead to improved tuning outcomes.

In light of this, numerous transfer learning techniques have been proposed to accelerate the tuning process by leveraging the knowledge from the historical tuning tasks. Related studies include workload mapping [4, 41], model ensemble [50], model pre-training [10, 21], and search space reduction [18, 52]. Typically, the first three techniques involve initializing the tuning model by leveraging and transferring knowledge gleaned from historical tuning tasks stored in a repository. Subsequently, for a new workload, this initialized tuning model is further refined to recommend a good configuration. Leveraging insights gained from past tuning tasks, the initialized model could speed up the convergence process, thereby enhancing tuning efficiency. The last technique differs slightly, as it focuses on reducing the search space for the new workload based on information learned from historical tuning tasks. By narrowing down the search space, this technique could enhance the efficiency of existing tuning methods.

Motivation. While various attempts have been made to expedite the tuning process, they consistently initiate the search (or iteration) from the default values of knobs, overlooking the significance of the starting point's influence. We posit that an appropriate starting point can expedite the tuning algorithm's convergence towards a solution and enhance the likelihood of discovering the global optimum. To substantiate this hypothesis, we conduct a pilot study on two widely-adopted workloads, namely TPC-H and JOB, employing a conventional BO-based approach called High-Efficiency Bayesian Optimization (HEBO) for knob tuning. The objective is to compare the effects of tuning from default values versus a high-quality starting point of configurations.

The pilot study unfolds in three steps: **(1) Obtain the best configuration:** Initially, HEBO optimizes knobs iteratively over 100 rounds, starting from default values, to obtain the best configuration. **(2) Perturb to obtain a high-quality starting point:** Subsequently, random noise is injected into the best configuration derived from the first step. This slight perturbation may yield a new configuration superior to the default yet inferior to the best configuration, labeled as a high-quality configuration, which serves as the starting point for HEBO. **(3) Tuning from the high-quality starting point:** By adjusting the magnitude of the random noise, four different suboptimal configurations are generated as starting points, and HEBO is performed on each individually. We record the intermediate configurations during the tuning processes, evaluating their latency in seconds and depict them in Figure 1.

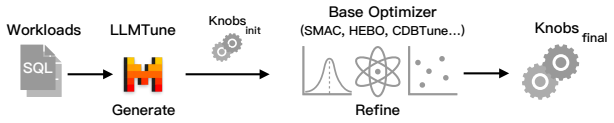

Figure 2: The proposed generate-then-refine framework.

It becomes evident that trials commencing from a high-quality configuration swiftly achieve exceptional performance within as few as 5 iterations. Furthermore, after a total of 100 iterations, an advantageous initial configuration could lead to further refinement, resulting in better performance than starting from the defaults. Through the pilot study, we observe that finding a good initialization configuration for the workload may represent an unexplored and efficient approach to accelerating the knob tuning process.

Building on the concept of transfer learning, we can train a machine learning or a deep learning model to discern the relationship between database workloads and their near-optimal configurations based on historical tuning tasks. This model can then infer the initial configurations for various base optimizers like SMAC [12], HEBO [7], GA [20], and OtterTune [41], thereby improving their tuning efficiency. Nonetheless, the challenge lies in the question: *How can we effectively learn from historical tuning tasks and apply this knowledge to new, unseen tasks?* The complexity stems from the myriad of knobs in databases, each with a wide range of potential values, both continuous and discrete. The broad search space makes it difficult for traditional machine learning or deep learning models to efficiently extract insights from past tuning tasks.

Fortunately, the recent advancements in artificial intelligence, especially with large language models (LLMs), offer a promising avenue. These models have shown exceptional skill in modeling intricate distribution mapping relationships, owing to the "attention" architecture and pre-training with large-scale corpora. For example, LLMs have demonstrated their outstanding performance in various complex tasks like solving math word problems [34], text-to-SQL [22], KBQA [25], and competitive programming [26], as indicated by various studies. This suggests that LLMs have the potential to learn the complex distribution mapping between input workloads and the resulting near-optimal configurations in database systems.

Our Approach: The primary objective of this study is to accelerate the previous process of knob-tuning in databases by initiating knob configurations from an advantageous starting point. In pursuit of this goal, we present LLMTune, an innovative data-driven strategy that utilizes LLMs to recommend a high-quality initial configuration for any database workload. This method is built upon the insight that DBAs typically collect a vast amount of historical tuning data, covering a broad spectrum of workloads and database schemas. Then, LLMTune employs LLMs to learn valuable insights

from these historical tuning tasks, allowing for generating a high-quality initial configuration tailored to the specific features of a new tuning task. Finally, starting with this initial setup, previous knob tuning methods can then be applied for further refinement. Therefore, our LLMTune adopts a "generate-then-refine" framework, as illustrated in Figure 2. The innovation of LLMTune lies in its use of LLMs to directly generate an initial configuration. This approach circumvents the time-intensive and often inefficient "trial and error" search process common in previous methods. Additionally, unlike RL-based tuning methods, which typically require further online tuning for new workloads, LLMs are inherently better at generalizing to new situations due to their extensive learnable parameters. Consequently, once the model is fully trained, it can be readily applied to a wide array of new workloads, thereby eliminating the need for additional online tuning and its associated overheads.

The development of LLMTune hinges on accessing historical tuning data from a vast array of workloads, ideally numbering in the hundreds or thousands. Presently, such a comprehensive dataset is not readily available within the open-source community. Moreover, it's crucial to know the tuning results for each workload to serve as training labels. To overcome this limitation, we introduce a fully automated data generation framework. This system is capable of creating a diverse and substantial set of workloads for any specified database, and it autonomously optimizes these workloads to acquire the respective high-quality configurations. Specifically, in the workload generation phase, we utilize GPT-4 to craft workloads, drawing on the database's schema and a selection of sampled values. For the labeling phase, the HEBO algorithm is employed in place of DBAs to optimize the knobs of the GPT-4-generated workloads, thereby generating the necessary training labels. However, a challenge arises with the HEBO algorithm, which requires numerous runs of each workload to effectively train a Gaussian Process model. This necessity significantly escalates the time required for data generation. To address this bottleneck, we introduce a cost model that substitutes actual execution. This model is designed to estimate the running time of a workload under a specific configuration, thereby streamlining the process.

The main contributions of this paper are as follows:

- We introduce a novel framework, LLMTune, designed to enhance the efficiency and effectiveness of current knob tuning techniques. This is achieved by utilizing large language models to recommend well-suited initial configurations. Importantly, once LLMTune is trained, it possesses the inherent ability to adapt to new, unseen workloads and database schemas, eliminating the need for additional online tuning.
- To gather training data for LLMTune while reducing the need for extensive human annotation labor, we have developed an automated data generation framework. This system efficiently synthesizes new workloads using GPT-4 and assigns appropriate high-quality configurations as labels. For this labeling process, we employ the HEBO algorithm, augmented with a cost model, to ensure effective and accurate configuration optimization.
- To facilitate ongoing research within the database community, we open-source the code, model checkpoints, databases, and all generated workloads.
- We have carried out a comprehensive set of experiments to validate the efficacy of LLMTune, focusing on a variety of new workloads from both seen and unseen database schemas. Our findings indicate that, compared to current leading methods, LLMTune is capable of finding the most effective configuration in significantly fewer tuning steps.

2 RELATED WORK

We review current approaches for knob tuning and the application of LLM in database optimization.

2.1 Knob Tuning

We survey works contributing to knob tuning, categorized into four types: Bayesian Optimization (BO) based methods, Reinforcement Learning (RL) based methods, Deep Learning (DL) based methods, and Knowledge Transfer methods [53].

BO-based methods. BO-based methods utilize Gaussian Process (GP) models as surrogate models. During each tuning step, a configuration is sampled from the GP model, applied to the database, and the workload is executed against the database engine to acquire performance metrics. These metrics are then employed to guide the update of the GP model. The pioneering approach, iTuned [8], adopts this methodology by refitting a GP model for each workload without utilizing historical experience.

Subsequent works have further enhanced BO-based methods by integrating additional workload and underlying data characteristics into GP models. For instance, OnlineTune [51] incorporates query arrival rates and types as query features, and data tuples and indexes as data distribution features. CGPTuner [4] and RelM [16] consider features and interactions across different system levels, such as memory control across workloads, containers, and JVM setups. Additionally, ResTune utilizes resource utilization metrics such as CPU, memory, and I/O usage.

However, even with these advancements, when a completely different workload arrives, BO-based models still require multiple steps of iteration and updating to recommend configurations. This iterative process can consume significant time, ranging from approximately 30 minutes to several hours.

RL-based methods. RL-based methods exhibit superior generalizability compared to BO-based methods due to their trained neural networks serving as both the actor and critic. CDBTune [48], the pioneer in employing Deep Deterministic Policy Gradient (DDPG) [32] for database knob tuning, utilizes database runtime metrics for state representation. QTune [21] enhances this representation by incorporating query and execution information, thereby improving adaptability to various workloads, although it encounters challenges when dealing with unfamiliar database schemas. Similarly, WATuning [10] employs an attention-based network for workload categorization primarily based on read-write ratios, offering a more customized approach. However, there's a risk of neglecting other critical workload features. These RL-based methods often face difficulties in convergence.

DL-based methods. Some approaches have employed deep learning-based methods. The DNN Method [42] uses a deep neural network with two hidden layers and Gaussian noise, aiding in exploring

diverse configurations. iBTune [33] specializes in tuning buffer pool sizes, selecting candidates based on cache miss ratios, and using neural networks for performance prediction. These DL-based models typically serve as cost models to predict a workload's performance, substituting the role of executing against the database. While this can enhance tuning efficiency, it may come at the cost of sacrificing tuned performance.

Knowledge Transfer. In database knob tuning, knowledge transfer methods encompass workload mapping, model pre-training, and model ensemble, each with distinct approaches and varying impacts on task generalization. Workload mapping, exemplified by OtterTune [41] and CGPTuner [4], utilizes historical workload similarities to initiate tuning models, offering a better starting point of tuning models but potentially limiting adaptability to unique or evolving workloads. Model pre-training, as seen in methods proposed by QTune [21] and WATuning [10], integrates detailed workload features into tuning models, enhancing specificity but possibly at the cost of overfitting to particular workload types. Lastly, Model ensemble approaches combine multiple well-trained models to address a wider range of workloads, effectively tackling the cold-start problem and ensuring adaptability, yet they may face challenges in balancing the ensemble for optimal performance across highly diverse workload scenarios [50]. In addition, a line of studies [18, 52] uses knowledge learned from historical tuning experience to optimize the search space. They usually dynamically select a sub-set of important knobs and narrow the value range of each knob to accelerate the tuning process.

2.2 Large Language Models for Databases

Recently, there has been a lot of research on using LLMs to enhance database systems. DB-GPT [57] introduces an automated prompt strategy utilizing LLMs for query rewriting and index tuning. DB-bert [37] implements the BERT model for database knob tuning. CodeXDB [36] develops a framework built upon GPT-3 to simplify complex SQL queries into manageable steps. Additionally, Trummer [40] offers a tutorial aimed at DBAs on utilizing LLMs for large-scale data management. Evaporate [1] proposes a comprehensive system for processing semi-structured documents into queryable tables. Furthermore, the capability of GPT to undertake additional database-related tasks, such as converting text to SQL, is demonstrated in recent works [38, 39].

3 PROBLEM DEFINITION

We first introduce the preliminaries of LLM and then formalize the database knob tuning problem.

3.1 Large Language Models

LLMs, which are primarily built on the Transformer architecture introduced by Vaswani et al. [43], offer a robust framework for learning from extensive textual data. This foundational architecture has paved the way for the creation of highly complex models, such as GPT-4 by OpenAI (2023) [26], PaLM [6], and LLaMa [34], that boast billions of parameters. These parameters enable the models to discern and replicate the nuanced patterns of human language. By pre-training on wide-ranging and diverse datasets, models like GPT-4 can assimilate a vast array of knowledge from different fields. This extensive pre-training equips them to excel at a variety of language-related tasks with impressive effectiveness, as demonstrated in the work by Brown et al. [2], and even the complex decision-making scenarios [3, 31].

While LLMs gain broad linguistic knowledge through pre-training, they often require additional expertise for specific tasks. Supervised Fine-Tuning (SFT) addresses this by further training the model with task-specific labeled data, enhancing its pre-trained knowledge base with targeted insights. In contrast, in-context learning allows LLMs to adapt to new tasks through tailored prompts, bypassing the need for SFT. This method's success hinges on the prompt's quality and the model's capabilities. For tasks like knob tuning, which involves determining optimal settings based on specific metrics and workloads, the gap between the task's technical nature and the LLM's language-based training makes in-context learning less feasible. As such, preparing targeted supervised data for SFT emerges as the preferred strategy to equip LLMs for these specialized tasks.

3.2 Problem Definition

Consider a database system endowed with a collection of adjustable system parameters, denoted by $K = \{k_1, k_2, ..., k_n\}$. These parameters, or "knobs", encompass various configurable aspects of the database, such as the size of work memory, and the maximum number of connections, among others. Each knob k_i is associated with a specific value s_i that falls within a predefined range S_i, meaning $s_i \in S_i$, which indicates the spectrum of permissible values for each knob.

The entirety of possible knob settings forms a multidimensional configuration space for the database system, represented by $S = S_1 \times S_2 \times ... \times S_n$. A particular point within this space signifies a unique database configuration, characterized by a set of knob values $\mathbf{s} = (s_1, s_2, ..., s_n) \in S$.

The aim of the knob tuning task for databases is to determine the optimal configuration within this multidimensional space S for a given database D, which includes details of the database engine's status, its schema, and its content, under a specific workload w. "Optimal" here refers to achieving the best outcome according to a performance metric M, such as minimizing query execution time, maximizing system throughput, or optimizing resource utilization. Formally, knob tuning in this paper is defined as:

Problem 1. *Given a databse D and a workload w, develop an LLM with parameters Θ that serves as a mapping function:*

$$\mathtt{LLM}_\Theta : (D, w) \rightarrow \mathbf{s}$$

This function takes the database D and workload w as inputs and outputs a configuration $\mathbf{s} = (s_1^*, s_2^*, ..., s_n^*) \in S$ that is chosen to either maximize or minimize the metric M, depending on the specific requirements of the metric.

The resulting configuration $\mathbf{s}$ can serve either as the final tuning outcome or as a high-quality initial point for further refinement using conventional tuning techniques. The primary aim of this research is to identify an optimal configuration that minimizes the need for additional tuning steps.

Workload Generalization. Our approach, fundamentally a learning-based model, necessitates an evaluation of its ability to generalize

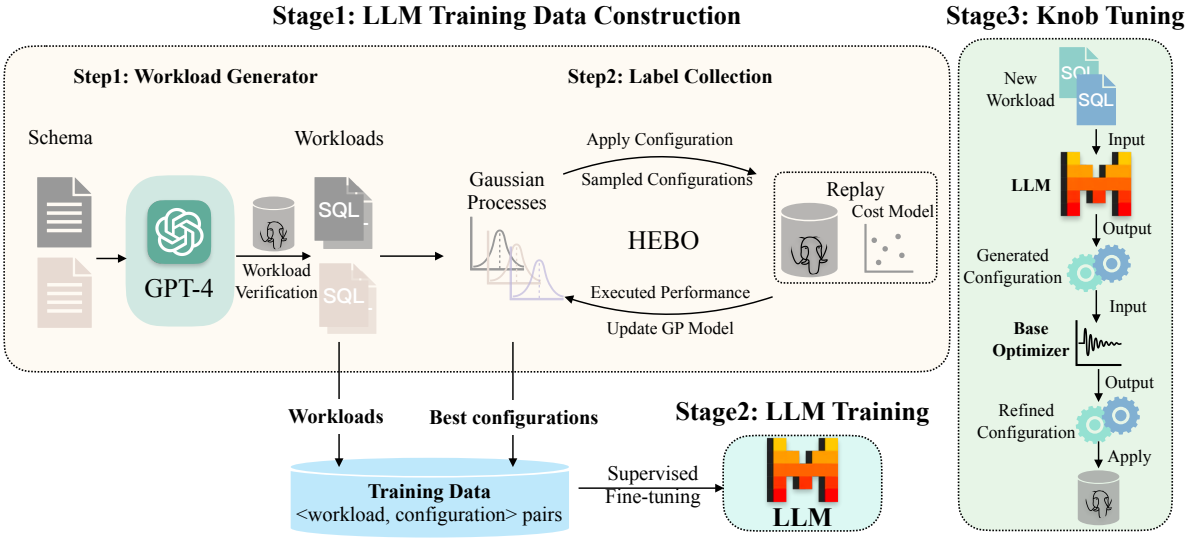

Figure 3: Overview of our workflow.

to workloads not seen during training. To assess this capability, we train our model on a specific dataset and then test its performance on a completely different set of workloads. Moreover, we distinguish between in-schema and cross-schema workloads: in-schema workloads refer to those within the same database schema as the training data, while cross-schema workloads involve different database schemas, presenting a more rigorous test of the model's adaptability. A model that excels in both in-schema and cross-schema tests demonstrates strong generalization capabilities. Unlike traditional tuning methods such as BO and RL, which are applicable to any workload irrespective of the schema but require substantial tuning effort, our model aims to identify an effective starting configuration for new workloads across both in-schema and cross-schema scenarios. This approach seeks to reduce the necessity for extensive subsequent tuning.

4 SYSTEM OVERVIEW

We propose LLMTune for knob tuning that unfolds in three main stages. The overall framework is depicted in Figure 3.

Stage 1: LLM Training Data Construction (Section 5). The first stage focuses on offline collection of training data to train LLMs, addressing the lack of existing training data. This is achieved through a fully automated data generation framework capable of producing a vast array of workloads using GPT-4 across various database schemas and determining their optimal configurations as labels with a SOTA BO-based tuning method, HEBO [7].

For workload generation, we provide GPT-4 with detailed inputs including the database schema, selected column values, and definitions for workload types (OLAP and OLTP), guiding it to generate realistic database workloads. These inputs are crafted to ensure that GPT-4 produces a wide variety of workloads that reflect real-world database operations. By specifying schema details and workload characteristics, we enable GPT-4 to generate workloads that are not only syntactically correct but also contextually relevant to typical database usage scenarios.

HEBO, upon receiving a specific workload, dynamically adjusts database parameters through 100-200 iterative cycles. Each iteration involves stress testing on actual database instances, causing a total tuning time of over tens of hours, which is highly time-intensive and impractical for us to collect enough training data. To mitigate this challenge, we collect and analyze data from each HEBO iteration to train a surrogate cost model, effectively capturing the relationship between configurations and database performance. This surrogate model can be used to substitutes the database testing in the HEBO cycle, significantly reducing the tuning time. Through the surrogate model, we observe a reduction in tuning duration by a factor of 10 to 100, streamlining the optimization process while maintaining accuracy in parameter selection.

Stage 2: LLM Training (Section 6). The second stage involves offline fine-tuning of the LLM with the data collected in the first stage, teaching it to generate optimal configurations based on given workloads and database characteristics. We collect real-time performance metrics and operational characteristics of the database under various workloads. This includes recording a wide array of data points such as query execution times, resource utilization rates (CPU, memory, disk I/O), internal metrics, and transaction latency. These metrics, alongside specific features of the workloads (such as query complexity and read vs. write operations), serve as inputs in our training dataset. As output, we generate the change in normalized configuration.

Stage 3: Knob Tuning (Section 7). Once the model is trained, it can perform inference. By outputting modifications for each knob value, our model supports iterative inference, allowing for multiple adjustments of knob values until there is no further improvement in performance.

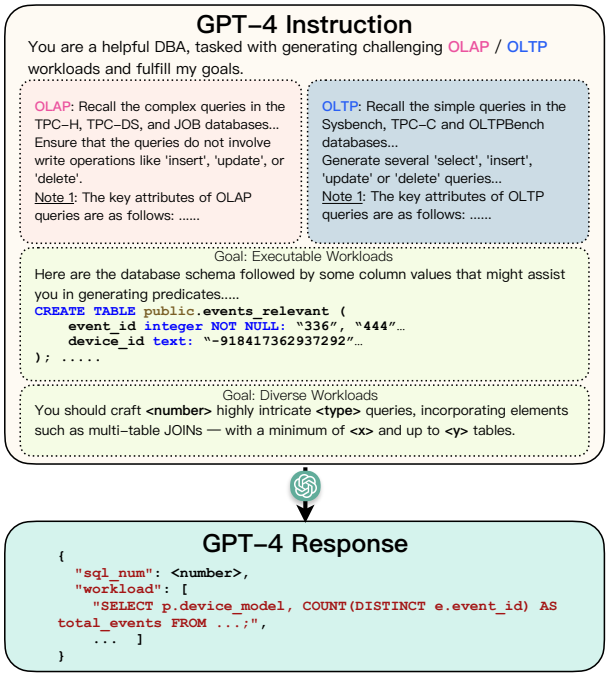

Figure 4: The prompt for workload generation using GPT-4.

5 LLM TRAINING DATA CONSTRUCTION

We begin by providing a comprehensive overview of the entire workflow, followed by detailed explanations of each component.

5.1 Data Construction Workflow

Our approach to generating training data hinges on pairing each workload with its corresponding optimal configuration. This begins with **generating workloads (Section 5.2)** using GPT-4, each tailored to a specific database schema. To verify the accuracy of these workloads, we execute them in the database, discarding any that result in SQL errors. Subsequently, we apply HEBO [7], a SOTA BO-based tuning method, to adjust the database knobs for these workloads, aiming to find the most efficient configuration for **label collection (Section 5.3)** of LLM training data.

A notable challenge in this process is that the HEBO algorithm necessitates multiple runs of the database engine per workload to assess the performance of sampled configurations, significantly extending the data generation timeline. To tackle this issue and streamline the data generation stage, we **construct a cost model (Section 5.4)**. Instead of executing a workload in real time to evaluate its performance, we utilize the trained cost model to predict the performance of a workload, thereby improving the overall efficiency of our training data preparation.

5.2 Workload Generation

We instruct GPT-4 by prompts to generate workloads to satisfy the following requirements:

- **Executable.** GPT-4's robust capabilities in code generation ensure strict adherence to SQL syntax. However, beyond mere syntax compliance, it is crucial that the generated SQL queries accurately reference the relevant tables and columns within the schema. To achieve this, GPT-4 is supplied with the schema creation statements of the database. Moreover, we pre-select values for each column in tables to enrich its semantics, providing GPT-4 with references for generating SQL queries within workloads.
- **Categorically Distinctive.** Workloads typically fall into two primary categories: OLAP (Online Analytical Processing) and OLTP (Online Transaction Processing). It is crucial that the workloads generated possess clear categorical characteristics to enhance their practical utility. The prompts explicitly define these categories, instructing GPT-4 to craft workloads with these specific attributes. Notably, OLAP-type workloads are instructed to exclude write operations.
- **Diverse.** Diversity in the generated workloads is assured by clearly specifying in the prompt the expected number of SQL queries and other characteristics, such as the average number of join tables. This ensures a broad spectrum of workload scenarios.

To meet these requirements, we finalize the prompt design for instructing GPT-4 as illustrated in Figure 4.

5.3 Label Collection

Given a database D with simulated content and a workload W generated by GPT-4, we tune the HEBO model to obtain the optimal configuration $\mathbf{s}$. After each tuning step, we apply the obtained configuration and execute the workload on the database engine to assess its performance. By comparing the current performance with the previous performance, we can determine whether the current configuration has improved and decide the next configuration to be sampled. The final tuned configuration $\mathbf{s}$ serves as the label for the current workload, resulting in the training data $\{(D, w, \mathbf{s})\}$.

To further support the multi-step inference of LLMTune as introduced in Section 7, we also record the intermediate tuned configurations at step t that significantly enhance performance compared to the configuration τ steps prior, resulting in the training data $\{(D, w, \{(\mathbf{s}^{t-\tau}, \mathbf{s}^{t})\}\}$[1].

5.4 Cost Model Construction

We train a cost model to efficiently predict a workload's performance instead of directly executing it on the database engine per tuning step to obtain accurate performance metrics. Previous studies [49] have shown that Random Forest and Gradient Boosting models consistently deliver superior accuracy when employed as the cost model. Consequently, we adopt a regression ensemble comprising these two models for our cost model. Building upon previous research efforts [48, 49], we concentrate on optimizing 45 critical knobs identified by DBAs specifically for PostgreSQL.

Cost Model Input. The feasible range for each parameter is determined based on the hardware's capabilities. For example, the shared buffer's range is set from 0 to 40% of the available memory size, as specified in the official documentation. In cases where a specific range S_i is not explicitly defined, we default it to a range from zero to Python's largest integer value. Utilizing the predefined

[1] τ is set as 5 empirically.

value range, we perform min-max normalization for the value of each knob, as follows:

$$\hat{s}_i = \frac{\max(S_i) - s_i}{\max(S_i) - \min(S_i)}, \tag{1}$$

where s_i denotes the value of the i-th knob and S_i denotes the value range of the i-th knob as defined in Section 3.2. Then $\hat{\mathbf{s}} = (\hat{s}_1, \hat{s}_2, \cdots, \hat{s}_n)$ represents a normalized configuration for the knobs.

In addition to the normalized configuration, we also integrate features extracted from the workload, encompassing six metrics: the frequency of table access, the number of SQL queries, the read-write ratio, the average number of predicates per SQL query, and the ratio of significant keywords including order by, group by, and aggregation functions. Overall, the input of our cost model can be summarized as:

$$\text{Input} : \text{concat}(\hat{\mathbf{s}}, \mathbf{t}, \mathbf{f}), \tag{2}$$

where $\hat{\mathbf{s}}$ denotes the normalized configuration, $\mathbf{t}$ denotes the vector consisting of the access frequency of each table in the given workload, and $\mathbf{f}$ denotes the vector of the aforementioned six workload metrics. We omit the inclusion of database features due to the significant differences in schema and content, which require variable-length feature vectors to represent tables in a schema and additional features to depict the content distribution in the database, thus making feature representation complex. To address the absence of database features, we train a separate cost model for each database. This strategy proves efficient as the cost model is lightweight, resulting in low training costs.

Cost Model Output. Since the scale of output performance for different workloads varies, and the objective is to distinguish between good and bad performance among different configurations for the same workload, we normalize the output performance within each workload to the range [0,1]. This normalization enables us to interpret configurations approaching 0 as good and those approaching 1 as bad. This interpretation aligns with our paper's performance metric, where lower average latency is considered better across all SQL queries in a workload.

To achieve this, we aggregate configurations belonging to the same workload and normalize them using the formula:

$$\text{Output} : \hat{p}_{ij} = \frac{p_{ij} - \min(P_i)}{\max(P_i) - \min(P_i)} \tag{3}$$

where p_{ij} denotes the performance of the j-th configuration for the i-th workload, and P_i denotes the set of performances for all configurations belonging to the i-th workload. This normalization approach is based on the understanding that our primary interest lies not in the absolute performance metrics of database execution, but rather in discerning the relative preference of configurations for the same workload.

Cost Model Training Data. For each database, we select approximately 30 workloads to tune against the databases and obtain the configurations at each step of the tuning process. Using these (workload, configuration) pairs, we train the cost model until the predicted performance closely matches the actual performance obtained through execution.

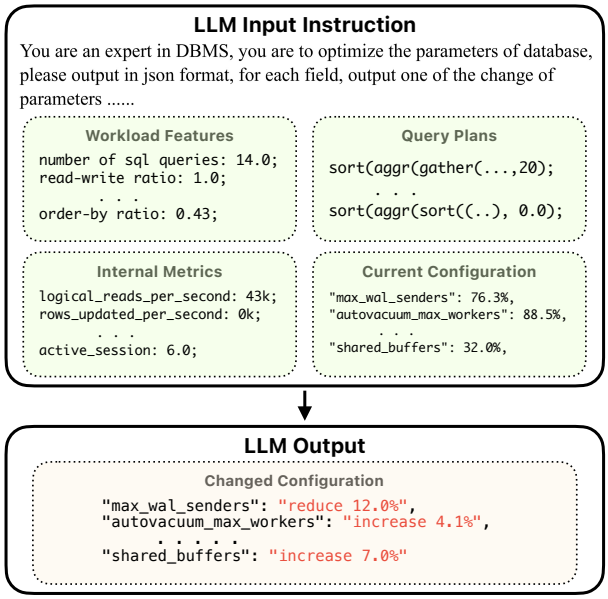

Figure 5: Illustration of LLMTune's input and output.

Once we have the cost model, we can use it to provide the performance of the sampled configurations at each tuning step instead of relying on actual database execution. This significantly reduces tuning time, resulting in more efficient collection of the entire LLM training data.

It's important to note that for collecting the labels of configurations, we still need to execute the workload under the configurations against the database to evaluate its real performance. Only configurations with superior performance compared to the default configuration and the configuration τ steps prior are recorded in the training data.

6 LLM TRAINING

This section provides a comprehensive explanation of the input and output for LLM training, followed by an introduction to the details of the LLM fine-tuning method.

6.1 LLM Input

In terms of the LLM's input, it is imperative to provide extensive information pertinent to workload and hardware specifications. This is crucial for enabling the model to adeptly discern the intricate relationships between workloads and their corresponding knob values. We refer to the PostgreSQL official documentation and relevant research on workload characterization, such as Qtune [21], OnlineTune [51], and zero-shot [11] and select the following features as LLM's input.

- **Workload Features.** The workload features are selected the same as those utilized for the cost model discussed in Section 5.4.
- **Internal Metrics.** Internal database metrics include metrics such as "pg_stat_database" and "pg_stat_bgwriter" within the PostgreSQL database engine. We utilize a serialization method to convert workload features and internal metrics into natural language

inputs. To enhance comprehension for the language model, we simplify large numbers, such as converting 83,438,203 into 83.44 million, for simplicity and clarity.

- **Query Plans.** Drawing inspiration from Qtune [21], we incorporate the query plan as input information. Instead of embedding them beforehand, we directly input the query plans associated with all SQL queries in the workload into LLMs. In cases of tree-like query plan structures, we represent their configuration using nested parentheses, and additionally, we append each node with the cost estimate provided by the PostgreSQL database engine.

6.2 LLM Output

Concerning the model's output, the representation of database knob values as large numbers significantly increases the complexity of the model's learning process. Therefore, we normalize these values using the same method outlined in Eq. 1, which is applied to the cost model's input. Formally, based on the collected training data $\{(D, W, \{(\mathbf{s}^{t-\tau}, \mathbf{s}^t)\})\}$ introduced in Section 5.3, we normalize $\mathbf{s}^t$ from step t and $\mathbf{s}^{t-\tau}$ from step $t-\tau$ into $\hat{\mathbf{s}}^t$ and $\hat{\mathbf{s}}^{t-\tau}$, respectively. We then calculate their difference $\delta = \hat{\mathbf{s}}^t - \hat{\mathbf{s}}^{t-\tau}$ as the configuration change, which serves as the LLM output. To complement the change output, we also include the old configuration $\hat{\mathbf{s}}^{t-\tau}$ as the LLM input.

In this manner, LLM is designed to take the previous configuration as input and generate the configuration change, which is conjectured to be easier for the model to learn compared to directly outputting the new configuration. Additionally, with the incorporation of value changes, LLMTune can iteratively infer multiple times, with each iteration based on previously predicted values, thereby gradually approaching the optimal knob settings. Figure 5 illustrates LLM's input and output.

6.3 LLM Fine-tuning

For LLM fine-tuning, we utilize the LLaMA-Factory framework[2] to train Mistral-7B [14] with PyTorch 2.1.2. Our model is trained on a server running Ubuntu 22.04, equipped with 8 NVIDIA H100 80GB HBM3 GPUs and 2048GB of RAM. We accelerate training using Huggingface Transformers [45] 4.36.2 and the DeepSpeed[29] ZERO-3 [28] Optimizer. With a learning rate of 2e-5, a batch size of 4 per GPU, and gradients accumulated over 4 steps, we achieve a total batch size of 128 across 8 GPUs. Training employs a cosine learning rate scheduler and lasts for 4 epochs. After training, we deploy models using the HuggingFace text-generation-inference package on a separate server with the PostgreSQL engine for knob tuning. Since we use a 7B-parameter backbone model, all experiments, except training, can efficiently run on machines with a consumer-grade GPU and at least 24GB VRAM.

7 KNOB TUNING

Generation. Once the model is trained, it can iteratively generate configurations through multi-step inference, facilitated by the combination of the old configuration input and the configuration change output. In each iteration, we apply the previously generated configuration on the database, run the workload against the database engine, collect query plans and internal metrics, and then feed them, along with the old configuration, into our tuned LLM to obtain the configuration change. We then add this change to the old configuration to derive the new configuration. By repeating these iterations, we gradually converge towards a final configuration as the predicted one. We halt the multi-step inference process when there is no performance improvement observed for three continuous steps.

Refinement. The generated configuration can serve as a starting point for further tuning using base optimizers to refine it.

[2]Available at https://github.com/hiyouga/LLaMA-Factory/

Table 1: Data statistics

Name	#Workloads	#Tables	Size
TPC-H	402	8	24.0GB
IMDB (JOB)	118	21	6.9GB
codebase_comments	420	4	2.4GB
language_corpus	420	6	17.0GB
movie_platform	425	5	3.2GB
bike_share_1	422	4	4.2GB
aminer_simplified	427	9	1.5GB
talkingdata	383	12	5.5GB
world_development_indicators	427	6	1.3GB
donor	427	4	2.7GB

8 EXPERIMENT

In this section, we carry out experiments to assess the performance of the proposed LLMTune by comparing it with other leading methods. These comparisons are made within the context of knob tuning tasks across four different datasets, under both in-schema and cross-schema workload scenarios. Our primary focus is on analyzing the tuning performance as well as the online runtime efficiency of the various tuning approaches. Additionally, considering that LLMTune features sophisticated component designs, including both input features and output formats, which can potentially affect its overall performance. Moreover, the choice of inference strategy and LLM backbone are also likely to influence its effectiveness. To evaluate the impact of each of these factors, we perform extensive ablation studies, ensuring a thorough verification of each component's contribution to LLMTune's performance.

8.1 Experimental Settings.

8.1.1 Datasets. Our evaluation employs 10 databases, encompassing a total of 3,871 workloads. Among these, TPC-H and JOB are two well-recognized databases, each accompanied by a single workload. The remaining eight databases are sourced from BIRD [23], a benchmark frequently used for assessing text-to-SQL tasks. Utilizing these databases, we emulate the database content and generate workloads based on the methodology outlined in Section 5.2. Besides the predefined workloads from TPC-H and JOB, we further create additional workloads for each. The workloads are then divided into training and testing sets for both in-schema and cross-schema evaluations. Specifically, we randomly select 40 workloads from TPC-H, JOB, and the first seven BIRD databases as the test set, allocating the rest as training data. This setup constitutes our in-schema evaluation framework, which is referred to TPC-H, JOB, and BIRD (in-schema).

Table 2: Comparison of performance and efficiency. We present average performance improvements (percent, denoted as Δ) and the total execution time to find the optimal configuration (minutes, denoted as T) of LLMTune and other baseline methods, including the basic tuning models HEBO [7], SMAC [12], and CDBTune [48], as well as transfer learning methods applied to these basic tuning models. Results are reported on in-schema benchmarks including TPCH [35], JOB [19], and BIRD (in-schema), as well as the cross-schema benchmark, BIRD (cross-schema). The best results are bolded.

Methods	**TPC-H** [35]		**JOB** [19]		**BIRD (in-schema)**		**BIRD (cross-schema)**	
	Δ (%) ↑	T (min) ↓	Δ (%) ↑	T (min) ↓	Δ (%) ↑	T (min) ↓	Δ (%) ↑	T (min) ↓
HEBO	61.7	1381.7	59.8	766.4	21.7	377.3	**18.3**	489.8
+ Workload Mapping	61.7	424.3	60.7	308.2	20.6	80.6	16.6	368.6
+ Model Ensemble	62.1	254.7	59.2	241.4	21.1	109.2	18.0	321.2
+ LLMTune	**62.7**	**88.8**	**61.3**	**60.3**	**23.8**	**40.6**	**18.3**	**115.0**
SMAC	58.9	1182.5	62.5	724.7	21.5	345.6	17.7	474.6
+ Workload Mapping	59.3	378.8	61.3	283.2	20.0	88.4	16.9	393.2
+ Model Ensemble	61.5	272.8	**62.9**	249.9	22.5	79.6	**18.3**	320.0
+ LLMTune	**63.2**	**76.2**	**62.9**	**58.3**	**22.8**	**35.5**	18.1	**135.6**
CDBTune (online tuning)	62.4	965.4	39.6	489.5	18.0	243.4	17.4	257.8
+ Model Pre-training (MP)	64.4	569.4	42.3	198.4	19.9	72.3	17.5	89.4
+ LLMTune	**69.4**	**209.7**	**71.4**	**27.9**	**23.3**	**57.2**	**18.4**	**59.3**
CDBTune w/ MP (no online tuning)	60.6	634.7	38.1	178.9	16.7	61.2	**16.7**	72.1
Pure LLMTune	**62.9**	**45.3**	**59.4**	**34.5**	**21.4**	**26.0**	**16.7**	**56.2**

For the cross-schema evaluation, all workloads from the last two BIRD databases, "language_corpus" and "bike_share_1", are allocated to the test set, distinguishing them from the training data and emphasizing their role in cross-schema assessment, which is named as BIRD (cross-schema). We concentrate on generating complex workloads to simulate the OLAP scenario in this paper. While knob tuning in the OLTP scenario could potentially be addressed using similar methods, this remains to be empirically validated in future research.

8.1.2 Metrics. We respectively evaluate the effectiveness and efficiency of each comparison method. Given the use of OLAP workloads, effectiveness is assessed through the **average latency (seconds) of each query in the workload**. Additionally, to provide a more intuitive understanding of the knob tuning performance, we calculate the improvement in performance as:

$$\Delta = \frac{\text{default latency} - \text{optimized latency}}{\text{default latency}}, \tag{4}$$

where the default latency is obtained by executing the workload before knob tuning, and the optimized latency is obtained after configuring the tuned knobs. We then report the average latency across all workloads.

Additionally, to assess the efficiency of different tuning methods, we record the total time required online to obtain the best knobs for each method. This encompasses the time from the initiation of tuning until the optimal configuration is reached. The time required for each iteration is determined by various factors, including recommending knobs, applying knobs, conducting stress tests, and, for BO and RL-based methods, updating models. For BO-based methods, the recommending knob time refers to the time taken for sampling knobs according to the Gaussian Process. In RL-based methods, the recommending knob time refers to the time required for predicting a knob given the database's internal metrics by the actor. In workload mapping and ensemble techniques, the time also includes workload matching time. For the proposed LLMTune, the time encompasses the LLM inference time.

8.1.3 Baselines. We evaluate the performance of BO and RL, two foundational tuning methodologies, integrating established transfer learning approaches to harness historical tuned knowledge for enhancing the efficiency of these primary tuning strategies. Here are brief descriptions of the fundamental methods utilized:

- **SMAC [12]**: This method, grounded in BO, is employed for database knob tuning and has been demonstrated to achieve SOTA results in knob tuning endeavors. Our implementation of SMAC relies on the Python SMAC3 library [24].
- **HEBO [7]**: Another BO-based strategy, HEBO is designed for optimizing hyperparameters and has shown notable success across various optimization tasks. Due to the absence of publicly available code, we develop our own implementation of HEBO.
- **CDBTune [48]**: As the pioneering RL-based approach for knob tuning, CDBTune's implementation is accessible through the open-source project OtterTune [41]. We leverage this available implementation in our work.

The transfer learning techniques include:

- **Workload Mapping**: Introduced in OtterTune [41], workload mapping aggregates historical tuning knowledge from similar workloads to initialize an effective tuning model. This process utilizes internal database metrics, such as "pg_stat_database" and "pg_stat_bgwriter" in the PostgreSQL database engine, as features to represent a workload. Similar workloads are identified through dot product similarities, and historical tuned knowledge, i.e., (configuration, latency) pairs, from these similar workloads are used to construct the initial tuning model. We apply this technique to enhance the performance of BO-based methods, HEBO and SMAC. For these methods, leveraging historical (configuration, latency) pairs enables the estimation of mean and variance for the proxy model instantiated by the Gaussian Process, thereby improving their efficiency and effectiveness.
- **Model Ensemble**: Proposed by Restune [50], this technique leverages historically tuned models on similar workloads to provide a strong initialization for the new tuning model. Initially,

it utilizes internal metrics from the database, along with workload tf-idf metrics, to represent workloads as features. It then identifies similar workloads using dot product similarities and selects models previously tuned on these similar workloads. Instead of relying solely on historically produced (configuration, latency) pairs for initialization, it ensembles these selected models to initialize the new tuning model. For ensemble construction, workload similarities are used to initialize the weights, which are then dynamically updated based on the prediction accuracy of matched tuned models during each tuning iteration. This technique is also applied to accelerate BO-based methods. Consequently, the model to be tuned remains a Gaussian process, accepting knobs as input and predicting latency as output.

- **Model Pre-training**: This transfer learning technique is commonly employed in RL-based methods to leverage the historical knowledge to initialize a good RL model. Specifically, the actor in the RL model is pre-trained to take database internal metrics as input and produce configuration based on the historical (internal metrics, configuration) pairs obtained during historical tuning processes. Meanwhile, the critic in the RL model is also pre-trained to take a configuration and internal metrics as input and predict latency based on historical (configuration, internal metrics, latency) triplets. This technique is implemented using Deep Deterministic Policy Gradient (DDPG), following the methodology outlined in [49].

The three transfer learning techniques are implemented using PyTorch by our team. To ensure fairness, the workloads utilized for these techniques are identical to the training data employed for the proposed LLMTune.

Workload mapping and model ensemble are applied to both HEBO and SMAC to initially match historical workloads for finding a well-initialized HEBO or SMAC model, followed by further tuning. Model Pre-training is exclusively applied to CBDTune to pre-train the actor and critic components before continuing to tune them within an RL framework. Model pre-training is also solely employed without further continuous tuning.

It takes about 0.5 hour for LLMTune fine-tuning on H100. The proposed LLMTune is applied to all three basic tuning models, with the inferred configuration serving as a starting point for continued tuning of HEBO, SMAC, or CBDTune.

8.1.4 Setup. We use PostgreSQL 12.2 to host and manage all databases and select 45 important knobs by database administrators. All experiments except supervised fine-tuning of LLMs, are performed on a server equipped with an Intel(R) Xeon(R) CPU E5-2650 v4 CPU (12 cores and 24 threads), 64GB RAM, and one NVIDIA RTX 3090 24GB GPU.

Currently, all tuning methods are trained and optimized for the specific database engine and hardware environment to facilitate empirical validation. However, the proposed LLMTune could potentially be generalized to other database engines and hardware environments by collecting data across various settings. We leave this experiment for future exploration.

8.2 Evaluation on In-schema Workloads

Table 2 presents all experimental results, including performance improvements and the runtime of the methods in the in-schema setting, which includes TPC-H, JOB, and BIRD (in-schema). The best-performing methods are highlighted in bold. Table 3 further presents the number of tuning steps and the duration per step for each method in TPC-H. Figure 6 illustrates the tuning processes of two workloads in the in-schema test sets. Our general observation is that, compared with all the baselines, **the proposed LLMTune not only offers a superior starting point for both BO- and RL-based methods to expedite their search but also enhances their performance to a certain extent.** The specifics of our findings via observing Table 2 are as follows:

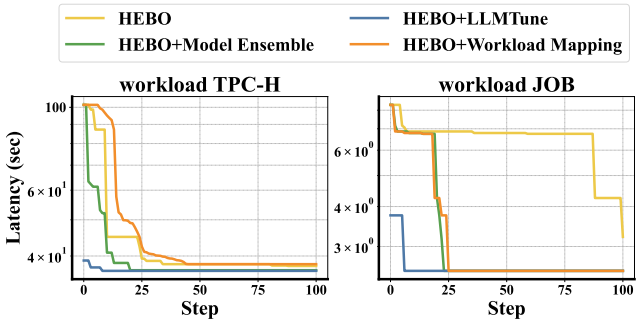

Figure 6: Case study of In-schema Evaluation.

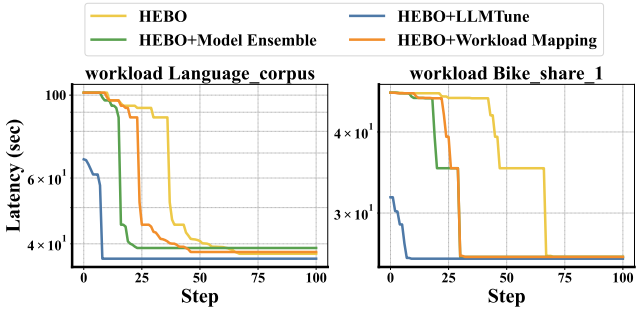

Figure 7: Case study of Cross-schema Evaluation.

- **LLMTune without refining results in comparable performance to basic BO methods.** Even without subsequent tuning by BO or RL-based methods, the pure inference results of LLMTune demonstrate comparable performance on JOB and BIRD (in-schema), and even show larger improvements on TPC-H, compared with the basic HEBO, SMAC, and CDBTune methods. This is attributed to LLMTune fine-tuning a powerful Mistral-7B LLM using 2,200+ supervised fine-tuning data pairs (workload, configuration), enhancing Mistral-7B with a robust ability to recommend knobs by generating optimized configurations.
- **LLMTune with further tuning could achieve better performance.** When applying LLMTune to HEBO, SMAC, and CDBTune on the three in-schema test sets, i.e., continuing to tune the recommended knobs by LLMTune using these basic tuning methods, we observe that almost all performances have improved compared to the pure recommended knobs by LLMTune . This indicates the effectiveness of combining LLM inference results with traditional lightweight tuning methods. However, while the improvements are

generally not significant, and in some scenarios, the improvement drops (i.e., on TPC-H, HEBO+LLMTune underperforms LLMTune by 0.2%), suggesting that the directly recommended knobs are already quite effective. Moreover, this drop in improvement could be avoided by discarding the continually tuned knobs while retaining the initially recommended knobs.

- **LLMTune outperforms other methods that also leverage historical knowledge.** In comparison with workload mapping, model ensemble, and more pre-training techniques, the proposed LLMTune offers a superior starting point for enabling HEBO, SMAC, and CDBTune to achieve larger performance improvements. Both workload mapping and model ensemble methods require matching the most similar workloads and then leveraging either their corresponding historical tuned data or tuned models to initialize the new tuning model. However, such similarity matching relies on the exact matching of workload tokens or database internal metrics values, which may fail to retrieve desired historical workloads when the tested workloads are dissimilar from historical ones. Although the model pre-training method pre-trains the actor and critic components in the RL models on the same training data as ours, the instantiated deep learning models are relatively small compared to the LLMs we employ, potentially limiting their generalization ability when directly applied to new workloads (CDBTune w/ MP (no online tuning) vs. LLMTune). Even with subsequent online tuning, the model pre-training method still underperforms ours (CDBTune+Model Pre-training vs. CDBTune+LLMTune).
- **LLMTune requires the least amount of time to achieve comparable or even superior performance improvements.** Compared pure LLMTune with basic tuning models, the reductions are most pronounced, with reductions of 89% to 97% in terms of T (minutes), while maintaining similar performance improvements. Even when continuing tuning after the initial knob recommendation, significant time savings are still realized, with reductions of 76% to 95% in terms of T (minutes). Compared with transfer learning methods that leverage historical knowledge, although they also accelerate basic tuning models, their reduction in T (minutes) is only 41% to 82%, which notably lags behind the proposed LLMTune.

When further examining the results on TPC-H as shown in Table 3, we observe that the step duration for different tuning methods is almost identical, with differences primarily lying in the number of steps. This highlights that LLMTune requires the fewest number of steps to obtain the optimal knobs.

The tuning process cases depicted in Figure 6 further illustrate the convergence trends of different workloads. These plots demonstrate that the proposed LLMTune exhibits the most rapid convergence property with the lowest starting latency.

8.3 Evaluation on Cross-schema Workloads

We further evaluate the proposed LLMTune on cross-schema workloads, specifically BIRD (cross-schema), comparing it with all baselines, and presenting the results in Table 2. Our primary findings are as follows:

- **Challenges of Cross-schema Workloads:** Cross-schema workloads pose greater difficulty for tuning. Comparing TPC-H, JOB, BIRD (in-schema) with BIRD (cross-schema), we observe the largest performance gap between HEBO and pure LLMTune on BIRD (cross-schema) (18.3% - 16.7% = 1.6% on BIRD (cross-schema) vs. 1.2% on TPC-H, -0.4% on JOB, and -0.3% on BIRD (in-schema)). This suggests that LLMTune exhibits greater ease in generalizing to in-schema workloads due to their similar workload distributions. Similarly, when continuing to tune the knobs based on recommended configurations by the basic tuning methods, the reduction in tuning time in BIRD (cross-schema) is notably smaller compared to other in-schema test sets. For example, LLMTune reduces HEBO's tuning time by (489.8-115.0)/489.8 = 76.5% on BIRD (cross-schema) but significantly reduces it by (1381.7-88.8)/1381.7 = 93.6% on TPC-H. These observations highlight the challenges inherent in tuning cross-schema workloads.
- **LLMTune 's Superior Tuning Performance:** Equipped with subsequent basic tuning models, LLMTune achieves the best performance improvement compared to other transfer learning methods on cross-schema test sets, even surpassing basic tuning models. This indicates LLMTune's ability to identify a strong starting point for tuning even with out-of-distribution schemas.
- **LLMTune 's Optimal Speedup:** Despite a weaker speedup on BIRD (cross-schema) compared to in-schema test sets, LLMTune still outperforms other transfer learning methods in achieving the best speedup over basic tuning models. Workload mapping and model ensemble techniques that rely on similar workload matching are constrained by the breadth of historical workloads, making them less effective for significantly distinct workloads. Similarly, model pre-training methods suffer from limited generalization ability due to their reliance on small pre-trained models.

Table 3: Comparison of time to find the best knobs. #Steps denotes the number of iteration steps to find the optimal configuration. Step duration is the time required for each iteration, and total time denotes the total time different methods take to find the optimal knobs.

Methods	#Steps ↓	Step duration (min.) ↓	Total time (min.) ↓
HEBO	86	16.06	1381.7
+ Workload Mapping	28	15.15	424.3
+ Model Ensemble	17	**14.90**	254.7
+ LLMTune	**6**	**14.80**	**88.8**
SMAC	78	**15.16**	1182.5
+ Workload Mapping	25	**15.15**	378.8
+ Model Ensemble	18	**15.16**	272.8
+ LLMTune	**5**	**15.15**	**76.2**
CBDTune (online tuning)	64	15.09	965.4
+ Model Pre-training (MP)	38	**14.98**	569.4
+ LLMTune	**14**	**14.98**	**209.7**
CDBTune w/ MP (no online tuning)	42	**15.11**	634.7
Pure LLMTune	**3**	**15.10**	**45.3**

Figure 7 presents two cases of the tuning process on BIRD (cross-schema) test sets. Comparing with Figure 6, we observe that the convergence speed of LLMTune is slower and the starting latency is higher on BIRD (cross-schema) than on in-schema test sets, which also indicates the difficulty of tuning the cross-schema workloads.

8.4 Ablation Study

We conduct ablation experiments and discussions on different components of our method, focusing on the input workload and environment features, inference strategy, output knob format, and LLM backbone. The results are shown in Table 4.

Table 4: Ablation studies of LLMTune. We present the performance improvements Δ (%) ↑ compared to default PostgreSQL settings for each model variation. IS and CS represent "in-schema" and "cross-schema" respectively.

	TPC-H	JOB	BIRD (IS)	BIRD (CS)
Default	0.0	0.0	0.0	0.0
Pure LLMTune	**62.9**	**59.4**	**21.4**	**16.7**
Input Features				
- Internal Metrics	60.4	52.5	19.4	14.6
- Workload Features	56.9	57.9	18.5	13.3
- Query Plans	51.1	39.1	14.0	2.0
LLM Inference Strategy				
- Singe-step Inference	58.8	38.7	12.8	5.9
Knob Output Format				
- Direct Value	15.0	11.2	8.4	0.6
- Normized Direct Value	59.7	55.6	13.7	13.2
Backbones				
- CodeLLaMA-7B [30]	61.3	52.6	20.4	15.9
- LLaMA2-7B [34]	60.4	52.3	20.1	15.9

8.4.1 Input Features. As detailed in Section 6.1, our method's input features encompass query plans and estimated costs for SQL queries within the workload, alongside database runtime internal metrics and workload characteristics. In this section, we systematically ablate these features. The three variations are:

- **Internal metrics**: Removing database internal metrics.
- **Workload features**: Removing workload features.
- **Query plan**: Replacing query plans with their corresponding SQL statements.

The reduction in each feature input results in varying degrees of performance degradation, indicating the correlation between each feature and the optimal configurations of various knobs. This correlation aids the model in making recommendation decisions. Remarkably, removing the query plan feature notably impacts the performance of cross-schema tasks. This is attributed to the model's need to implicitly learn the data distribution of tables when presented with SQL inputs, thus facing challenges in handling new tables appearing in the SQL.

8.4.2 Multi-step Inference. As discussed in Section 5.3, we collect tuning data from multiple intermediate steps and utilize all of them to train LLMTune, enabling the model to predict the intermediate tuned knobs. During inference, LLMTune infers the value changes for knobs multiple times to gradually approach the optimal knob values. To evaluate the necessity of this training approach, we compare it with a single-step inference strategy.

- **Single-step Inference**: In this strategy, we do not collect any tuning data from intermediate steps but directly use the finally tuned optimal configuration. Accordingly, we perform a single inference to obtain the desired configuration.

The results indicate that our multi-step inference strategy outperforms the single-step approach. Firstly, the intermediate tuning data provides the LLM with more comprehensive training data. Secondly, training on such intermediate tuning data equips LLMTune with the ability of iterative reasoning, enabling it to improve its inference performance through multiple rounds of inferences.

8.4.3 Output Knob Format. As outlined in Section 6.2, LLMTune is trained to output the value change relative to each knob's initial value. The initial value, represented by its original value, is accepted by LLMTune as additional input, while the value change, normalized as a percentage using Eq. 1, is accepted by LLMTune as the output. By incorporating the value change into the initial value, we can derive the tuned value. This input-output setting allows the model to iteratively infer value changes, progressively approaching the final optimal knob values. To verify the effectiveness of this proposed value change format, we explore two alternative output formats:

- **Direct Value**: Directly output the values of the knobs.
- **Normalized Direct Value**: For each knob, we divide its value range into k[3] equal-distance buckets and assign values falling within each bucket to the corresponding bucket, effectively normalizing the float values into discrete intervals.

The results indicate that outputting direct values performs the worst. This is because the scope of values for each knob is extensive, and the ranges of different knobs vary significantly. Consequently, this presents substantial challenges for LLMTune to learn the underlying patterns of knob values. In contrast, normalized direct values perform much better. By limiting the values to discrete intervals, clearer patterns emerge for LLMTune to learn. However, even when normalized, outputting direct values still lags behind outputting value changes. This is because learning value changes is inherently easier than directly learning the optimal values. Additionally, with value changes, LLMTune can iteratively infer multiple times, with each iteration based on previously predicted values, thereby gradually approaching the optimal knob settings.

8.4.4 LLM Backbone. To explore the capabilities of different LLM backbones for the knob tuning task, we adopt LLaMA2 [34] and CodeLLaMA [30] respectively to replace the Mistral model used in LLMTune. LLaMA2 is a text-focused LLM and CodeLLaMA is a code-focused LLM. We utilize the officially released instruction-tuned versions with 7B parameters for a fair comparison. In Table 4, they are denoted as "LLaMA2-7B" and "CodeLLaMA-7B". Most models demonstrate commendable knob tuning capabilities, with the Mistral model exhibiting the best performance among them.

9 CONCLUSION

This paper delves into methods for improving the efficiency of database knob tuning, adopting an approach distinct from previous transfer learning techniques. Our focus is on leveraging advanced LLMs to generate effective configurations by effectively assimilating knowledge from historical tuning tasks. The configuration recommended by the LLM serves as the initial starting point for various base optimizers like HEBO and SMAC. Through extensive experimentation, we demonstrate that LLMTune significantly enhances tuning efficiency and effectiveness over current state-of-the-art methods, proving particularly effective in the complex context of cross-schema scenarios.

[3]We set k as 5 in our experiments.

REFERENCES

[1] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. 2023. Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes. *arXiv preprint arXiv:2304.09433* (2023).

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[3] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712* (2023).

[4] 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. *Proceedings of the VLDB Endowment* 14, 8 (2021), 1401–1413.

[5] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. 2023. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology* (2023).

[6] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, and et al. 2022. PaLM: Scaling Language Modeling with Pathways. *CoRR* abs/2204.02311 (2022). arXiv:2204.02311

[7] Alexander I Cowen-Rivers, Wenlong Lyu, Zhi Wang, Rasul Tutunov, Hao Jianye, Jun Wang, and Haitham Bou Ammar. 2020. Hebo: Heteroscedastic evolutionary bayesian optimisation. *arXiv preprint arXiv:2012.03826* (2020), 7.

[8] Songyun Duan, Vamsidhar Thummala, and Shivnath Babu. 2009. Tuning database configuration parameters with ituned. *Proceedings of the VLDB Endowment* 2, 1 (2009), 1246–1257.

[9] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and robust automated machine learning. *Advances in neural information processing systems* 28 (2015).

[10] Jia-Ke Ge, Yan-Feng Chai, and Yun-Peng Chai. 2021. WATuning: a workload-aware tuning system with attention-based deep reinforcement learning. *Journal of Computer Science and Technology* 36, 4 (2021), 741–761.

[11] Benjamin Hilprecht and Carsten Binnig. 2021. One model to rule them all: towards zero-shot learning for databases. *arXiv preprint arXiv:2105.00642* (2021).

[12] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*. Springer, 507–523.

[13] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825* (2023).

[14] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B. arXiv:2310.06825 [cs.CL]

[15] Konstantinos Kanellis, Cong Ding, Brian Kroth, Andreas Müller, Carlo Curino, and Shivaram Venkataraman. 2022. LlamaTune: sample-efficient DBMS configuration tuning. *arXiv preprint arXiv:2203.05128* (2022).

[16] 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*. 1667–1683.

[17] Meghdad Kurmanji and Peter Triantafillou. 2023. Detect, Distill and Update: Learned DB Systems Facing Out of Distribution Data. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–27.

[18] Jiale Lao, Yibo Wang, Yufei Li, Jianping Wang, Yunjia Zhang, Zhiyuan Cheng, Wanghu Chen, Mingjie Tang, and Jianguo Wang. 2023. GPTuner: A Manual-Reading Database Tuning System via GPT-Guided Bayesian Optimization. *arXiv preprint arXiv:2311.03157* (2023).

[19] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How good are query optimizers, really? *Proceedings of the VLDB Endowment* 9, 3 (2015), 204–215.

[20] Stefan Lessmann, Robert Stahlbock, and Sven F Crone. 2005. Optimizing hyperparameters of support vector machines by genetic algorithms.. In *IC-AI*, Vol. 74. 82.

[21] Guoliang Li, Xuanhe Zhou, Shifu Li, and Bo Gao. 2019. Qtune: A query-aware database tuning system with deep reinforcement learning. *Proceedings of the VLDB Endowment* 12, 12 (2019), 2118–2130.

[22] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 13067–13075.

[23] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems* 36 (2024).

[24] 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. *The Journal of Machine Learning Research* 23, 1 (2022), 2475–2483.

[25] Haoran Luo, Zichen Tang, Shiyao Peng, Yikai Guo, Wentai Zhang, Chenghao Ma, Guanting Dong, Meina Song, Wei Lin, et al. 2023. Chatkbqa: A generate-then-retrieve framework for knowledge base question answering with fine-tuned large language models. *arXiv preprint arXiv:2310.08975* (2023).

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

[27] Neha Patki, Roy Wedge, and Kalyan Veeramachaneni. 2016. The synthetic data vault. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 399–410.

[28] Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, and Yuxiong He. 2021. Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–14.

[29] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3505–3506.

[30] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950* (2023).

[31] Murray Shanahan. 2024. Talking about large language models. *Commun. ACM* 67, 2 (2024), 68–79.

[32] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic policy gradient algorithms. In *International conference on machine learning*. Pmlr, 387–395.

[33] Jian Tan, Tieying Zhang, Feifei Li, Jie Chen, Qixing Zheng, Ping Zhang, Honglin Qiao, Yue Shi, Wei Cao, and Rui Zhang. 2019. ibtune: Individualized buffer tuning for large-scale cloud databases. *Proceedings of the VLDB Endowment* 12, 10 (2019), 1221–1234.

[34] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models, 2023. *URL https://arxiv. org/abs/2307.09288* (2023).

[35] Transaction Processing Performance Council (TPC). Year of the specification version. TPC-H Benchmark Specification. Online. Available: http://www.tpc.org/tpch/.

[36] Immanuel Trummer. 2022. CodexDB: Synthesizing code for query processing from natural language instructions using GPT-3 Codex. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2921–2928.

[37] Immanuel Trummer. 2022. DB-BERT: a Database Tuning Tool that" Reads the Manual". In *Proceedings of the 2022 International Conference on Management of Data*. 190–203.

[38] Immanuel Trummer. 2023. Can Large Language Models Predict Data Correlations from Column Names? *Proceedings of the VLDB Endowment* 16, 13 (2023), 4310–4323.

[39] 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.

[40] Immanuel Trummer. 2023. From bert to gpt-3 codex: harnessing the potential of very large language models for data management. *arXiv preprint arXiv:2306.09339* (2023).

[41] 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*. 1009–1024.

[42] 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. *Proceedings of the VLDB Endowment* 14, 7 (2021), 1241–1253.

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[44] Junxiong Wang, Immanuel Trummer, and Debabrota Basu. 2021. UDO: universal database optimization using reinforcement learning. *arXiv preprint arXiv:2104.01744* (2021).

[45] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771* (2019).

[46] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale

human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887* (2018).

[47] Zhuohao Yu, Chang Gao, Wenjin Yao, Yidong Wang, Wei Ye, Jindong Wang, Xing Xie, Yue Zhang, and Shikun Zhang. 2024. KIEval: A Knowledge-grounded Interactive Evaluation Framework for Large Language Models. *arXiv preprint arXiv:2402.15043* (2024).

[48] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, et al. 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*. 415–432.

[49] 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. *Proceedings of the VLDB Endowment* 15, 9 (2022), 1808–1821.

[50] 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*. 2102–2114.

[51] 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*. 631–645.

[52] Xinyi Zhang, Hong Wu, Yang Li, Zhengju Tang, Jian Tan, Feifei Li, and Bin Cui. 2023. An Efficient Transfer Learning Based Configuration Adviser for Database Tuning. *Proceedings of the VLDB Endowment* 17, 3 (2023), 539–552.

[53] Xinyang Zhao, Xuanhe Zhou, and Guoliang Li. 2023. Automatic Database Knob Tuning: A Survey. *IEEE Transactions on Knowledge and Data Engineering* (2023).

[54] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2024. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems* 36 (2024).

[55] Xuanhe Zhou, Chengliang Chai, Guoliang Li, and Ji Sun. 2020. Database meets artificial intelligence: A survey. *IEEE Transactions on Knowledge and Data Engineering* 34, 3 (2020), 1096–1116.

[56] Xuanhe Zhou, Guoliang Li, and Zhiyuan Liu. 2023. Llm as dba. *arXiv preprint arXiv:2308.05481* (2023).

[57] Xuanhe Zhou, Zhaoyan Sun, and Guoliang Li. 2024. DB-GPT: Large Language Model Meets Database. *Data Science and Engineering* (2024), 1–10.