



E2ETUNE: End-to-End Knob Tuning via Fine-tuned Generative Language Model

Xinmei Huang
Renmin University of
China
huangxinmei@ruc.edu.cn

Haoyang Li
Renmin University of
China
lihaoyang.cs@ruc.edu.cn

Jing Zhang
Renmin University of
China
zhang-jing@ruc.edu.cn

Xinxin Zhao
Renmin University of
China
zhaoxinxin798@ruc.edu.cn

Zhiming Yao
Renmin University of
China
yaojimmy2005@ruc.edu.cn

Yiyan Li
Renmin University of
China
liyiyang@ruc.edu.cn

Tieying Zhang
ByteDance Inc.
tieying.zhang
@bytedance.com

Jianjun Chen
ByteDance Inc.
jianjun.chen
@bytedance.com

Hong Chen
Renmin University of
China
chong@ruc.edu.cn

Cuiping Li
Renmin University of
China
licuiping@ruc.edu.cn

ABSTRACT

Database knob tuning is a significant challenge for database administrators, as it involves tuning a large number of configuration knobs with continuous or discrete values to achieve optimal database performance. Traditional methods, such as manual tuning or learning-based approaches, typically require numerous workload replays and are both time-consuming and resource-intensive. To address this challenge, we introduce E2ETUNE, an end-to-end knob tuner powered by a fine-tuned generative language model. The key idea is to leverage the exceptional sequence-to-sequence modeling capabilities of generative language models to capture the complex mapping between workloads (inputs) and their corresponding promising configurations (outputs). To achieve this goal, we propose a novel data generation framework to efficiently produce a large amount of training data, where each data sample consists of a workload and its promising configuration. Then, these data are used to fine-tune a generative language model, yielding an end-to-end knob tuner. This tuner offers out-of-the-box configuration recommendations for new workloads. We conduct extensive experiments to evaluate E2ETUNE's efficiency and effectiveness using 10 representative and 3 real-world benchmarks. Compared to state-of-the-art methods, E2ETUNE can identify competitive configurations in significantly less time.

PVLDB Reference Format:

Xinmei Huang, Haoyang Li, Jing Zhang, Xinxin Zhao, Zhiming Yao, Yiyan Li, Tieying Zhang, Jianjun Chen, Hong Chen, and Cuiping Li. E2ETUNE: End-to-End Knob Tuning via Fine-tuned Generative Language Model. PVLDB, 18(13): 5540 - 5554, 2025.
doi:10.14778/3773731.3773732

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

Xinmei Huang and Haoyang Li contributed equally to this work. Work done during the internship at ByteDance.
Jing Zhang and Tieying Zhang are the corresponding authors.

PVLDB Artifact Availability:

The source code, data, and/or other artifacts have been made available at <https://github.com/RUCKBReasoning/E2ETune>.

1 INTRODUCTION

Knob tuning in databases involves adjusting the values of various knobs to optimize performance for specific workloads. This process is essential because default database configurations are not always suitable for every workload. Effective knob tuning can significantly improve databases' performance and reliability. However, this task is NP-hard due to the vast number of tunable knobs and their wide configuration ranges. Traditionally, database administrators (DBAs) manually perform knob tuning, relying on their experience—a time-consuming and impractical approach for numerous workloads and database instances. With advances in machine learning, researchers have developed automated knob tuners, primarily including Bayesian Optimization (BO)-based methods (e.g., iTuned [19], SMAC [29]) and Reinforcement Learning (RL)-based methods (e.g., CDBTune [87], UDO [81]).

Limitations of Existing Methods. Although existing BO and RL-based methods are capable of autonomously identifying appropriate configurations, they face challenges with tuning efficiency. In particular, for each new given workload, these methods typically require extensive iterations to achieve the desired database performance. Each iteration involves three steps: exploring a new configuration from a learned model, replaying the workload under the new configuration, and refining the model based on the database feedback. This process often involves numerous workload replays, resulting in suboptimal tuning efficiency. For example, for the TPC-H benchmark [71] at a scale factor of 6, our preliminary studies reveal that HEBO [15], a robust BO method for hyperparameter optimization, takes approximately 23 hours to identify a

licensed to the VLDB Endowment.

Proceedings of the VLDB Endowment, Vol. 18, No. 13 ISSN 2150-8097.
doi:10.14778/3773731.3773732

promising configuration within 100 iterations. To speed up the tuning process, researchers have proposed various transfer learning methods to reduce the number of required iterations. For example, workload mapping [77], model ensemble [89, 90], and model pre-training [39] aim to initialize models using historical tuning data to potentially quicken model convergence speed. Additionally, the knob pruning method [36, 73, 91] focuses on narrowing the search space of configurations, by pinpointing essential knobs and their optimal ranges, thus improving tuning efficiency. However, even with these methods in place, finding a promising configuration still demands dozens to hundreds of iterations. Moreover, when the new workload significantly differs from those previously encountered, transfer approaches become almost ineffective.

Motivation and Our Proposal. As described above, the inefficiency of previous methods mainly derives from the iteration process, which incurs a lot of workload replays. To further speed up the tuning process, we propose to develop a novel, end-to-end knob tuning method named E2ETUNE. Figure 1 illustrates the core concept of E2ETUNE, which reimagines the knob tuning task as an end-to-end modeling process, effectively eliminating the time-consuming iterative nature of previous methods. To achieve this goal, inspired by the observation that skilled database administrators (DBAs) are often able to manually determine a suitable configuration for a new workload, drawing on their extensive experience, *we posit that there exists a complex distribution-mapping relationship between workloads and their optimal configurations*. Hence, by training a model capable of understanding the distribution-mapping relationship, we can establish an end-to-end solution for knob tuning.

Challenges and Our Solutions. However, developing such an end-to-end knob tuner presents significant technical challenges: (1) In our preliminary attempts, the first challenge we encounter is that traditional machine learning algorithms, such as multilayer perception (MLP) and random forest, can not efficiently learn the distribution mapping from the training data due to their limited learning capabilities. To tackle this challenge, motivated by the powerful end-to-end modeling capabilities of generative language models (LMs)¹, such as GPT-4 [51] and LLaMA-3 [48], fine-tuning a language model to learn and capture the complex distribution mapping will be a feasible solution. Therefore, in this paper, we innovatively formalize the knob tuning task as a sequence-to-sequence generation task, where the input is the features of the workload and the output is the promising knob configuration. (2) Then, to effectively train an end-to-end knob tuner, it's crucial to obtain a substantial dataset of <workload, promising configuration> pairs. Yet, the absence of publicly accessible datasets fulfilling this requirement poses a significant hurdle, marking the scarcity of training data as the second major challenge. To overcome this challenge, we introduce a novel training data construction framework designed to efficiently gather the necessary training data. This framework is structured around two principal components: the generation of workloads and the collection of labels. In the first component, our goal is to generate a large number of diverse and high-quality workloads, leveraging existing database instances. Following this, in the second component, we employ HEBO [15] to identify a promising

¹In this paper, we specifically use “language model” and its abbreviation “LM” to denote the sequence-to-sequence generative language model.

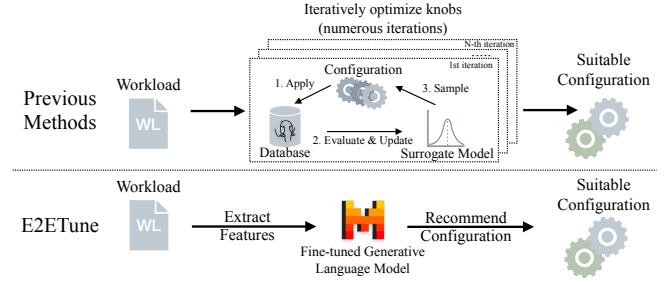


Figure 1: Previous knob tuning methods vs. E2ETUNE.

configuration for each synthesized workload as its label. Furthermore, to expedite the process of label collection, we have developed a cost model that serves as a substitute for actual workload executions within the iterative process of HEBO. In practice, after obtaining enough training data, we fine-tune Mistral-7B [30], a Transformer-based [79] generative language model with 7 billion learnable parameters, to perform end-to-end knob tuning.

Evaluation. To thoroughly evaluate E2ETUNE’s effectiveness, we use 10 representative benchmarks—TPC-H, TPC-DS, JOB, SSB, SSB-flat, TPC-C, Twitter, Wikipedia, Smallbank, and YCSB—and 3 real-world benchmarks: StackOverflow, SSAG, and AMPS. As E2ETUNE is fine-tuned using a wide range of diverse and high-quality workloads, it may exhibit the ability to generalize effectively to new test workloads across various database instances, whether seen or unseen during training, without the need for additional fine-tuning. Therefore, our evaluations encompass both “in-schema” and “cross-schema” scenarios. In the “in-schema” scenario, we evaluate E2ETUNE’s performance on test workloads from database instances included in the training sets (for example, training on generated TPC-H workloads and testing on the TPC-H official workload). Conversely, the “cross-schema” scenario explores E2ETUNE’s capacity to generalize to test workloads involving database instances it has not encountered previously (for instance, training on generated TPC-DS workloads and testing on the TPC-H official workload).

The main contributions of this paper are as follows:

- **End-to-end Database Knob Tuning:** We introduce E2ETUNE, a data-driven, end-to-end database knob tuning method based on the fine-tuned language model. It analyzes workload features and then directly recommends promising configurations, eliminating the need for numerous iterations. To the best of our knowledge, E2ETUNE is the first LM-based end-to-end knob tuning method.
- **Novel Data Generation Pipeline:** To gather training data for E2ETUNE, we develop a novel data generation pipeline. This involves synthesizing new workloads, labeling them with appropriate configurations, and introducing a new cost model as a proxy for actual execution to significantly speed up the data collection process.
- **Comprehensive Evaluation:** Our extensive evaluations, spanning 10 representative and 3 real-world benchmarks, reveal that E2ETUNE not only identifies the most effective configurations but also significantly enhances tuning efficiency, surpassing existing

methods. Moreover, detailed “cross-schema” experiments demonstrate that E2ETUNE can smoothly adapt to new workloads on previously unseen database instances.

2 RELATED WORK

2.1 Knob Tuning System

A knob-tuning system typically comprises a knob tuner to identify promising configurations and a knowledge transfer module to leverage historical knowledge. In contrast to prior studies, E2ETUNE integrates knowledge transfer into the core knob-tuning process, representing a novel approach. We will now review existing knob tuners and knowledge transfer methods.

2.1.1 Knob Tuning. Existing knob tuning methods can be classified into 4 categories: heuristic-based methods, Bayesian Optimization (BO)-based methods, Reinforcement Learning (RL)-based methods, and Deep Learning (DL)-based methods.

Heuristic-based methods. Heuristic-based methods involve exploring the search space through manually crafted rules [16] or a set of predefined heuristic rules [1]. Nonetheless, these approaches demand significant human intervention and often exhibit limited search efficiency, resulting in suboptimal performance.

BO-based methods. BO-based methods, such as VBO [80], HEBO [15], and SMAC [29], use surrogate models to estimate database performance metrics from given features. Each method uses a different surrogate model: VBO employs a vanilla Gaussian process, while SMAC uses a random forest. These approaches require many iterations to gather tuning observations. In each iteration, a configuration is sampled from the surrogate model and applied to the database. The workload is executed to obtain performance metrics, which refine the surrogate model. This iterative process enables surrogate models to iteratively improve their accuracy, leading to optimized database configurations.

The pioneering knob tuning system iTuned [19] uses a GP model as its surrogate. Subsequent systems have enhanced surrogate accuracy by incorporating various features. For instance, OnlineTune [90] includes query arrival rates, types and indexes. CGPTuner [9] and ReIM [34] focus on system-level attributes like memory control across workloads, containers, and JVM setups. ResTune [89] adds resource utilization metrics such as CPU, memory, and I/O usage. Despite advancements, BO-based models often require hundreds of iterations to recommend a suitable configuration for a new workload. This process is time-consuming and resource-intensive, highlighting the need for efficiency improvements in tuning.

RL-based methods. RL-based knob tuning methods [22, 39, 87] use reinforcement learning algorithms to adjust database knobs. The Deep Deterministic Policy Gradient (DDPG) [64], known for its actor-critic framework, is widely used in these systems. The actor selects a configuration based on the state, while the critic evaluates its efficacy, optimizing the policy and value function iteratively. DDPG handles continuous action spaces effectively, making it suitable for knob tuning. CDBTune [87] employs DDPG, using runtime metrics as the state to generate database configurations. QTune [39] enhances this approach with Double-State DDPG (DS-DDPG), incorporating query-related features, such as the types of queries

and the tables involved, to improve performance. However, like BO-based methods, RL-based approaches often require many iterations to achieve stable performance, which can be resource-intensive.

DL-based methods. DL-based knob tuning methods [66, 78] train neural networks as learned cost models to predict database performance from given features, facilitating rapid configuration exploration. For instance, [78] uses a deep neural network instead of the Gaussian process model in OtterTune [77]. Similarly, iBTune [66] trains a deep neural network to estimate response time using metrics like logical reads and CPU usage. However, these learned cost models are usually just part of the tuning system, which may still require iterative trials and adjustments.

2.1.2 Knowledge Transfer. Traditional knob tuning methods often require numerous iterations to identify promising configurations, posing efficiency challenges. To address these cold-start issues, researchers have developed various knowledge transfer methods: (1) Workload mapping (OtterTune [77]) uses past tuning data to improve BO-based tuners’ initial states. (2) Model pre-training (QTune [39]) pre-trains RL models with historical data, then fine-tunes them on new workloads for faster convergence. (3) Model ensemble (ResTune [89]) combines models from past tasks to adapt to new workloads. (4) Knob pruning (GPTuner [36], OpAdviser [91]) refines the configuration space by selecting critical knobs and their optimal ranges based on past experiences, reducing the search space.

Despite these advancements, base tuners still need additional iterations for new workloads, creating a bottleneck. Our paper proposes a novel approach to streamline the tuning process by modeling it as an end-to-end task using historical data, aiming to eliminate the need for multiple iterations.

2.2 Language Models for Databases

Recent research in databases has extensively harnessed LMs to optimize various aspects such as database diagnosis (e.g., root cause analysis) [98, 99], text-to-SQL [40, 41, 55], data preparation (e.g., value filling and entity resolution) [67], data integration [5, 65], query processing [72], and table question answering [24, 31, 85].

Notably, DB-BERT [73] and GPTuner [36] leverage LMs for knob tuning by using knowledge from web forums and manuals to constrain the search space, but still require a base tuner for iterative exploration. In contrast, our E2ETUNE trains a language model on extensive historical tuning tasks to directly predict suitable configurations, eliminating the need for online tuning. This model, after offline training, can be applied to various workloads across different database instances, significantly enhancing tuning efficiency.

3 PROBLEM DEFINITION

Definition 1 (Database Knob Tuning). In a database system, there is a set of adjustable knobs, denoted as $K = \{k_1, k_2, \dots, k_n\}$. These knobs control various configurable aspects of the database, such as work memory size and maximum connection limits. Each knob k_i can assume a value s_i that falls within a permissible range S_i , which can be either continuous numerical values or categorical forms. The entirety of feasible configurations forms a multi-dimensional space for the database system, denoted by $S = S_1 \times S_2 \times \dots \times S_n$. A unique configuration within this space is represented by a set of knob

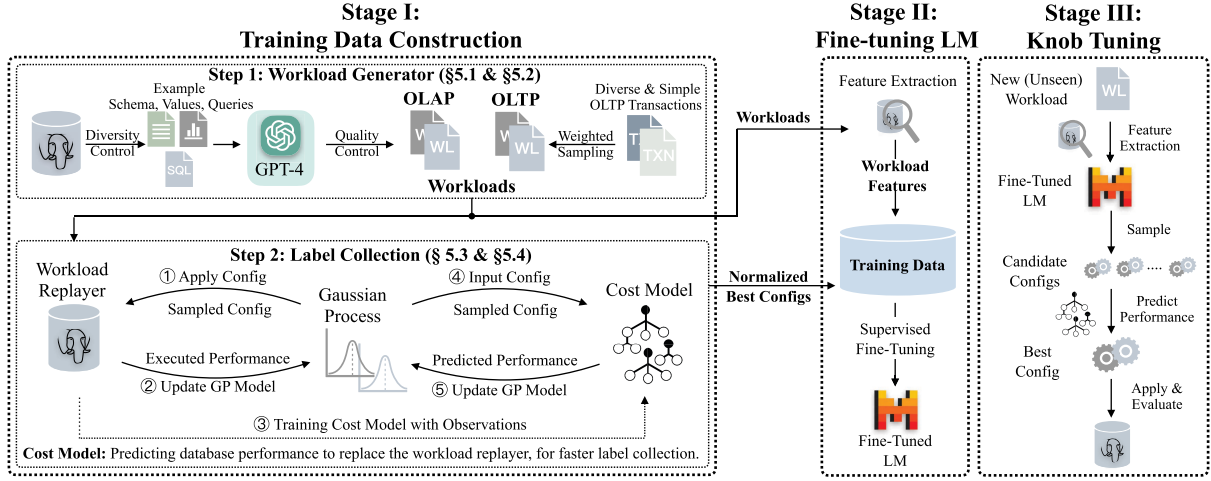


Figure 2: Overview of E2ETUNE (Refer to Section 4 for detailed explanation).

values $\mathbf{s} = (s_1, s_2, \dots, s_n) \in S$. Given a workload $W = \{q_1, \dots, q_m\}$ consisting of m queries and a database instance D , the objective of knob tuning is to find the optimal configuration $\mathbf{s}^* = (s_1^*, s_2^*, \dots, s_n^*)$ within the multi-dimensional space to optimize a performance metric M , such as minimizing latency or maximizing throughput. *Definition 2 (End-to-End Knob Tuning via LMs)*. Given a database instance D and a workload W , E2ETUNE aims to train a language model to learn the distribution mapping function between the workload features and the optimal configuration:

$$LM_{\Theta}(Feature(D, W)) \rightarrow \mathbf{s}^*, \quad (1)$$

where $LM_{\Theta}(\cdot)$ denotes the LM, Θ represents the learnable parameters of the LM, $Feature(\cdot)$ denotes the function for extracting features of workload W on its corresponding database instance D , and $\mathbf{s}^* = (s_1^*, s_2^*, \dots, s_n^*)$ represents an optimal configuration that maximizes or minimizes the performance metric M .

Since this is a data-driven method, it requires a substantial collection of $\langle D, W, \mathbf{s}^* \rangle$ triplets for training. To obtain such a dataset, we introduce a data generation pipeline detailed in Section 5. Specifically, we assume the availability of database instances and proceed to automatically generate a diverse array of OLAP and OLTP workloads (refer to Section 5.1 and Section 5.2). Subsequently, we capture the configurations associated with these workloads that lead to peak performance metrics (refer to Section 5.3). After obtaining enough training data, we use them to train a language model in a sequence-to-sequence manner, as outlined in Section 6. Finally, we deploy the trained LM to suggest suitable knob values for new workloads, as elucidated in Section 7.

4 E2ETUNE OVERVIEW

The overview of E2ETUNE is illustrated in Figure 2, which consists of 3 main stages.

Stage 1: Training Data Construction (Section 5). The first stage involves the offline collection of training data for fine-tuning the LM. This begins by generating diverse OLAP and OLTP workloads across database instances and identifying promising configurations.

For OLAP, GPT-4-Turbo creates complex SQL queries based on database characteristics, with diversity and quality control integrated. For OLTP, new workloads are generated by randomly weighting predefined transactions. A BO-based iterative tuner, HEBO, then determines promising configurations to serve as training labels. A cost model accelerates the label collection process during HEBO’s tuning. Specifically, real workload replaying generates observations to train the cost model, which then serves as a proxy for real executions during tuning.

Stage 2: Fine-tuning LM (Section 6). The second stage involves offline fine-tuning of the LM with the data collected in the first stage, teaching it to generate promising configurations based on the features of the workload. More precisely, we gather workload features from three dimensions: workload statistics, query plans, and internal database metrics. Furthermore, to address the language model’s limitations in understanding numerical values, we discretize the values of each configuration knob and task the LM with generating the discretized configurations.

Stage 3: Knob Tuning with Fine-tuned LM (Section 7). Once the LM is trained, it can be deployed to recommend configurations for any given workload. In practice, we sample multiple configurations from the LM by setting the temperature to 1.0 and then we use the cost model trained in stage 1 to select the configuration with the best estimated performance as the recommended result.

5 TRAINING DATA CONSTRUCTION

This training data construction process begins with generating diverse, high-quality workloads tailored to specific database instances. Workloads are categorized into OLAP and OLTP. OLAP workloads focus on complex analytical queries, primarily using SELECT statements for operations like aggregation and slicing. Their performance is measured by latency, the time taken to execute queries. OLTP workloads handle real-time transactional processes, comprising SELECT, INSERT, UPDATE, and DELETE statements. Their efficiency is evaluated by throughput, measured in transactions

per second. Given their distinct characteristics, we employ tailored workload generation strategies for OLAP and OLTP.

Subsequently, to identify appropriate configurations for the newly generated workloads, we leverage HEBO [15], a powerful BO-based knob tuner, to acquire the training labels of the LM (refer to Section 5.3). Nevertheless, the tuning process of HEBO necessitates numerous workload replaying, significantly impeding the pace of the data preparation. To address this challenge, we introduce a cost model to replace real executions in HEBO’s tuning process. This model reduces execution time and allows simultaneous tuning of multiple workloads on a single machine, eliminating resource contention. By integrating HEBO with the cost model, we efficiently generate a large volume of high-quality <workload, suitable configuration> pairs within a feasible offline timeframe.

5.1 OLAP Workload Generation

Given that each OLAP benchmark usually comprises a limited number of query templates with variable slots, generating new OLAP workloads by simply substituting these slots with different values might lead to insufficient diversity, potentially undermining the overall generalizability of the fine-tuned language model. Therefore, to create diverse and high-quality OLAP workloads, we first utilize GPT-4-Turbo [51, 52] to generate realistic and complex SQL queries based on the database information. After obtaining a sufficient number of queries, new OLAP workloads can be obtained by choosing a subset of these queries.

The prompt utilized to guide GPT-4 to generate OLAP queries comprises six components:

- “Task Overview” clarifies the objectives for GPT-4-Turbo.
- “Database Schema” outlines the database instance’s structure using Data Definition Language (DDL) statements, including table names, column names, primary and foreign keys, etc.
- “Guidance for Query Generation” sets constraints for query generation, such as the required SQL dialect.
- “Predicate Generation Aid” offers some column values to assist GPT-4-Turbo in formulating query predicates.
- “Sample OLAP Queries” presents a few benchmarked SQL queries to exemplify the desired complexity level of the generated queries.
- “Output Format” specifies the format of GPT-4-Turbo’s output text, aiding in parsing the generated query.

Diversity Control: To ensure query diversity, we have implemented several strategies: (1) First, we randomly select tables from the database and incorporate their DDL statements into the “Database Schema” part. This approach also reduces API costs associated with invoking GPT-4-Turbo, especially for database schemas with many tables and columns, as the model charges based on the number of tokens it must process. (2) Subsequently, we randomly choose column values from the selected tables to form the “Predicate Generation Aid” part. (3) Finally, we randomly select queries from the benchmarked queries to compose the “Sample OLAP Queries” part. To prevent direct copying, we clarify in the prompt that these queries are for illustrative purposes only.

Quality Control: To prevent syntax errors, we use the EXPLAIN command to evaluate the syntactical accuracy of generated queries. If errors are detected, GPT-4-Turbo is prompted to fix them based

on the EXPLAIN feedback. Queries with persistent syntax errors are discarded. Among the error-free queries, we execute each one and exclude any that exceed a specified execution time threshold (e.g., over one hour) to mitigate long runtimes.

5.2 OLTP Workload Generation

In the realm of OLTP benchmarks, each benchmark typically comprises a set of predefined transactions. For instance, in the case of TPC-C, five distinct transactions are defined: delivery, new order, order status, payment, and stock level. In contrast to OLAP benchmarks, we observe that adjusting the weights of individual transactions in OLTP benchmarks can result in a wide range of diverse OLTP workloads. This is because the attributes of an OLTP workload, such as the read-write ratio, can be altered by varying the transaction weights. Hence, by manipulating these weights, we can effectively generate a multitude of new OLTP workloads.

5.3 Label Collection

We identify suitable configurations for generated workloads to serve as training labels for the LM using HEBO [15]. Specifically, we perform HEBO to explore the desired configurations. However, HEBO, being a BO-based method, necessitates numerous iterations during the tuning process. Practically, tuning the knobs for a workload demands one to several hours, and a single machine can only tune one workload at a time to avoid resource preemption, significantly slowing data collection.

To address this issue, we introduce a novel cost model to substitute real executions in the iterations. The cost model is trained to predict database performance under a given configurations and workloads. By utilizing this model, we guide HEBO’s search process without extensive workload runs. Additionally, it allows concurrent tuning of multiple workloads, further improving the efficiency. Further details regarding the cost model are elaborated in Section 5.4.

Given that the cost model’s predictions may not be entirely precise, therefore, after obtaining the final configuration from the cost model-driven HEBO, we apply it to the database and execute the workload to verify that it surpasses the default configuration in terms of the database performance metric.

5.4 Cost Model

Cost Model Architecture. Research conducted by [88] has shown that Gradient Boosting Regressor (GBR) and Random Forest Regressor (RFR) stand out as powerful regression models, well-suited for the cost estimation task. GBR and RFR are trained independently on the same dataset and their predictions are averaged in an ensemble during inference, reducing overfitting and improving generalization to unseen data.

Cost Model Input. Given that the core aim of the cost model is to estimate performance metrics under specific configurations and workloads, the model’s input comprises two key components: the configuration specifics and workload features.

To enhance the efficiency, stability, and performance of the cost model, normalization of each knob value in a configuration is crucial due to the varying permissible value ranges across different knobs. We use min-max normalization for each numerical knob value,

expressed as:

$$\hat{s}_i = \frac{s_i - \min(S_i)}{\max(S_i) - \min(S_i)}, \quad (2)$$

where s_i represents the value of the i -th knob, and $\max(S_i)$ and $\min(S_i)$ denote the maximum and minimum values of the i -th knob, respectively. The range of knob values is typically dictated by hardware specifications, such as memory size, number of CPU cores, etc. In instances where a knob lacks a specified range S_i , a default range from 0 to $2^{31} - 1$ is assigned. Consequently, $\hat{s} = (\hat{s}_1, \hat{s}_2, \dots, \hat{s}_n)$ denotes a standardized representation of the configuration.

For workload features, we integrate database engine and operating system statistics to characterize the workload comprehensively. Following a careful selection process, we have identified 14 key metrics: `xact_commit`, `xact_rollback`, `blks_read`, `blks_hit`, `tup_returned`, `tup_fetched`, `tup_inserted`, `conflicts`, `tup_updated`, `tup_deleted`, `disk_read_count`, `disk_write_count`, `disk_read_bytes`, and `disk_write_bytes`². The metrics derived from running the workload using the default configuration offer valuable insights for recommending a promising configuration. For example, a high `disk_read_count` indicates that performance bottlenecks may stem from demanding disk read operations, suggesting that adjusting memory-related knobs could potentially improve overall performance.

Similar to configurations, we apply min-max normalization to the metric values based on the allowable range of the metrics to ensure consistency and numerical stability.

Finally, the input to the cost model is the concatenation of the normalized configuration features, denoted as \hat{s} , and the normalized workload features, denoted as \hat{f} . This combined input, $\text{Concat}(\hat{s}, \hat{f})$, ensures the cost model incorporates both configuration details and workload characteristics to provide robust and generalized performance estimations. Additionally, since the input vector length remains constant, the trained cost model can be applied to new workloads and database instances.

Cost Model Output. Given the varying scales of performance metrics across different workloads, directly predicting specific metric values presents significant challenges for the cost model. However, it's crucial to understand that the cost model doesn't need to precisely estimate these values across different workloads since comparing performance metrics between different workloads isn't required. Instead, the primary focus should be on distinguishing between superior and inferior configurations within the same workload, rather than accurately predicting absolute performance metric values. This approach simplifies the learning complexity for the cost model. To achieve this, we normalize the performance metrics across diverse configurations within a specific workload similarly to Eq. 2, obtaining \hat{p}_{ij} , the normalized performance metric for the j -th configuration of the i -th workload. Finally, the cost model is trained to predict normalized performance based on normalized configuration and workload features.

Cost Model Training Data. To train the cost model, we initially collect a significant amount of tuning observations (*i.e.*, <workload, configuration, performance metric> triplets) as training data using

²These metrics are derived from PostgreSQL, which serves as our experimental platform. It's worth noting, however, that other database engines, such as MySQL, offer capabilities to access similar internal metrics.

HEBO under the guidance of actual executions. Specifically, we start by randomly selecting 13 newly generated workloads from each of the 10 considered database instances, resulting in a total of 130 workloads. Subsequently, we conduct knob tuning using HEBO alongside real executions, setting the maximum number of iterations to 100, to find promising configurations for the selected workloads. At each iteration of the HEBO process, we capture a tuning observation as a training data sample for the cost model, accumulating a total of 13,000 training data samples. Following this, we train the cost model within a 10-fold cross-validation framework to assess its estimation accuracy. Further evaluation details of the cost model can be found in Section 8.4.2. Once a dependable cost model is established, we integrate it into HEBO to replace real executions, thereby expediting the label collection process.

6 FINE-TUNING LM

Once a sufficient number of historical tuning tasks have been acquired, we can proceed to train a language model to learn the complex relationship between workload and its suitable configuration. This section elaborates on how we formalize the knob tuning task in an end-to-end manner using the LM.

6.1 LM Input Sequence

The input to the LM primarily consists of workload features essential for enabling the model to understand and interpret the workload, thereby aiding in generating appropriate configurations. When presented with a workload, we comprehensively assess three dimensions of workload features: workload statistics (information at the workload level), query plans (information at the query level), and internal metrics (information at the system level), all of which are then fed into the LM. Workload statistics offer crucial insights into the inherent characteristics of the workload. Query plans furnish more detailed information regarding the individual queries within the workload. Internal metrics provide granular details about the workload at a system level. Note the workload features input to the LM are much richer than those used for the cost model because the LM can accept a more flexible format and longer input than traditional small GBR and RFR models used for building the cost model. The specific details are outlined below:

- **Workload Statistics.** Workload statistics include: the access frequency of each table, the total number of SQL statements, the read-write ratio, the average number of predicates per SQL query, and the proportion of key operators such as `ORDER BY`, `GROUP BY`, and aggregation functions.
- **Query Plans.** Inspired by Qtune [39], we integrate the query plans associated with all SQL statements in the workload. The query plans can be obtained using the `EXPLAIN` command. Unlike prior research that typically utilizes TCNN [47] or QueryFormer [93] for embedding query plans, we treat the query plans as text. Given that the query plan is structured as a sequence, we use nested parentheses to represent the hierarchical relationships between operations. Each pair of parentheses encloses either a single operation or a group of operations executed within a larger step in the plan. Additionally, we augment each operation with the cost estimated by the database engine.

Table 1: Data storage size and the number of training samples in each database instance.

Benchmark	TPC-H [71]	TPC-DS [70]	SSB [53]	SSB_flat [53]	JOB [37]	TPC-C [18]	YCSB [18]	SmallBank [18]	Twitter [18]	WikiPedia [18]
Size	8.6 GB	2.1 GB	13.3 GB	11.7 GB	6.9 GB	5.0 GB	4.0 GB	20.0 GB	0.4 GB	4.4 GB
#Workload	271	285	299	299	299	300	300	300	300	300

- **Internal Metrics.** Internal database metrics offer crucial insights into workload execution, resource utilization, and potential bottlenecks or inefficiencies within the database system. We utilize the identical set of 14 metrics as inputs for the cost model (see section 5.4). These metrics are obtained by running the workload under the default configuration. To aid the LM in comprehending large numerical values, we simplify them by representing them in orders of magnitude. For instance, we convert “83,438,203” to “83.4 million”.

All these features are flattened and concatenated into a sequence before being fed into the LM. The LM’s attention mechanism effectively integrates these multi-dimensional features to facilitate the subsequent configuration generation task.

6.2 LM Output Sequence

Utilizing the extracted workload features, the LM is trained to model the distribution of suitable configurations. However, generating numerical knob values directly poses challenges due to varying value scales across different knobs. To address this, we normalize the knob values in the configuration using the method detailed in Eq. 2. Then, we discretize these normalized values into pre-defined buckets. Specifically, we segment the knob values into 10 buckets, labeled as “0% to 10%”, “10% to 20%”, and so on. This approach enables the LM to generate the buckets corresponding to each knob, which is believed to be easier for the LM to learn compared to outputting exact numerical values. For additional training details, please refer to Section 8.1.4.

6.3 Loss Function

Given an input sequence x and its corresponding output sequence y , we optimize the parameters of the LM with a conditional language modeling loss: $p_{\theta}(y|x) = \prod_{t=1}^l p_{\theta}(y_t|x, y_{<t})$, where θ denotes the learnable parameters of the LM, l denotes the length of the output sequence, and $y_{<t}$ represents all preceding tokens before position t in the output sequence. The primary goal of fine-tuning is to enhance LM’s ability to predict the output sequence with maximum likelihood, given the input sequence.

7 KNOB TUNING WITH FINE-TUNED LM

After obtaining the fine-tuned LM, we can perform knob tuning in a **sampling-then-ranking** inference strategy.

Step1: Sampling. Given that the LM is fine-tuned to model the probability distribution of suitable configurations based on the workload features. Therefore, we can conduct sampling on the LM’s output distribution to obtain multiple candidate configurations during the inference phase. This sampling-based approach significantly boosts the diversity and innovation of the generated configurations, enabling the LM to explore a wider range of potential solutions and identify the optimal configuration. In practice, we typically use a temperature of 1.0 and set the sampling count to

8. Once the LM predicts a discretized configuration, we convert the buckets into numerical values via denormalization. For instance, if the LM predicts that the “max_wal_senders” knob falls within the “30% to 40%” bucket, we calculate the midpoint of this bucket, $(30\% + 40\%) / 2 = 35\%$, and then obtain the specific knob value using the maximum and minimum values of that knob.

Step2: Ranking. Following the generation of 8 candidate configurations, we rank them to identify the best option. For this ranking phase, we leverage the cost model introduced in Section 5.4 to evaluate and order the sampled configurations, ultimately selecting the best one as the final recommended configuration.

8 EXPERIMENT

8.1 Experimental Settings

8.1.1 Datasets. To evaluate E2ETUNE’s performance, we use 10 widely adopted benchmarks, including five OLAP benchmarks (TPC-H, JOB, SSB, SSB-flat, TPC-DS) and five OLTP benchmarks (TPC-C, Smallbank, YCSB, Twitter, Wikipedia). Each benchmark includes a database instance and predefined templates for SQL queries (OLAP) or transactions (OLTP).

We train the model on the synthetic training dataset and test its performance on the original workload of each benchmark. For OLAP’s test set, queries are generated by populating templates with random values. For OLTP’s test set, default transaction weights are used to create testing workloads. For the training set, we use the method introduced in Section 5 to generate 300 new workloads per database instance and gather their corresponding labels (*i.e.*, promising configurations). After filtering out underperforming configurations, we have 2,953 training samples for the LM.

The training data originates from two sources: HEBO with real execution and HEBO with a cost model. Real execution tunes 130 workloads for training the cost model. Labels for the remaining 2,823 samples are generated via the cost model. We ensure that original workloads are not included in the training data. Table 1 shows data size and training sample count per database instance.

To further evaluate the practical application value of E2ETUNE, we additionally consider three benchmarks from real-world scenarios: StackOverflow (4.5GB)³, SSAG (58GB), and AMPS (25GB). StackOverflow is an OLAP benchmark designed to analyze the website data, for example, mining the features of unanswered questions. SSAG and AMPS are two private business workloads in ByteDance. Specifically, SSAG is an OLAP benchmark utilized in slow SQL analysis and governance scenarios, encompassing complex queries such as slow SQL template analysis, logical database analysis, and new slow SQL identification. AMPS, on the other hand, is an OLTP benchmark used in the AI platform services, which includes transactions related to user management, permission control,

³The database is sourced from <https://stackoverflow.blog> and benchmarked queries are sources from <https://data.stackexchange.com/stackoverflow/queries>.

algorithm management, model management, and task scheduling. These benchmarks are employed solely as test sets.

8.1.2 Baselines. Traditional Methods. We evaluate the BO-based methods SMAC [29] and HEBO [15]. SMAC has shown strong performance in database knob tuning [88], while HEBO has demonstrated effectiveness across hyperparameter optimization tasks [15]. In addition, we try an RL-based method CDBTune [87], which uses the DDPG algorithm for database tuning. All methods are executed for a maximum of 100 iterations, with each iteration involving time-consuming workload replay. Furthermore, we utilize our trained cost model to replace real execution during the iterative process, resulting in three new baselines: SMAC + Cost Model, HEBO + Cost Model, and CDBTune + Cost Model. Finally, we randomly sample 1,000,000 configurations using the Latin Hypercube Sampling strategy from the knob space and employ the cost model to identify the best configuration. This baseline is referred to as Random Sampling + Cost Model.

Knowledge Transfer Methods. As categorized in Section 2.1.2, we consider four types of transfer methods: workload mapping, model ensemble, model pre-training, and knob pruning.

Workload Mapping : Introduced in OtterTune [77], this technique leverages historical tuning data from similar workloads to initialize a new tuning model. This technique is often integrated into BO-based methods, and we integrate it with HEBO and SMAC.

Model Ensemble : Proposed by Restune [89], this technique combines multiple historical knob-tuning models and incorporates workload features. For a new workload, similar historical workloads’ models are ensembled. Commonly, this integrates with BO-based methods.

Model Pre-training : QTune [39] introduces this technique, the actor and critic models are pre-trained with past tuning tasks and fine-tuned with RL-based methods. In our experiments, we assess it with CDBTune in two scenarios: with and without online tuning, where online tuning involves fine-tuning based on real-time feedback.

Knob Pruning : This technique reduces the search space. OpAdviser [91] refines the search space based on similar historical workloads’ insights. DB-BERT [73] and GPTuner [36] leverage language models to extract useful information from database manuals, effectively trimming the search space during tuning.

To ensure fairness, the historical tuning tasks utilized in these knowledge transfer techniques are the training dataset for E2ETUNE. The model for selecting tuning methods in OpAdviser is trained using their publicly available dataset.

8.1.3 Metrics. For OLAP workloads, our goal is to minimize the query latency. Therefore, the performance improvement is defined as: $\Delta = \frac{\text{default latency} - \text{optimized latency}}{\text{default latency}}$, where “default latency” is from the default configuration and “optimized latency” is from the recommended configuration. For OLTP workloads, our goal is to maximize throughput (*i.e.*, transactions per second, tps). The performance improvement is defined as $\Delta = \frac{\text{optimized tps} - \text{default tps}}{\text{default tps}}$, where “default tps” is from the default configuration and “optimized tps” is from the recommended configuration.

8.1.4 Implementation details. We use PostgreSQL 12.2 to manage all databases. Following prior studies [87, 88], we manually select 45 crucial knobs for tuning, while retaining default values for the rest. The DBMS is restarted with each new configuration since some knob modifications require it. For training the LM, we use Mistral-7B-Instruct-v0.2⁴ as the base model, fine-tuned with PyTorch 2.1 [2] and Hugging Face Transformers [83]. The learning rate is set to $2e-5$, batch size to 128, and maximum context length to 8,192. The model is trained for 4 epochs using a cosine decay learning rate scheduler. To save GPU memory, we use FlashAttention-2 [17] and DeepSpeed ZERO [58] for optimized data parallelism.

8.1.5 Environments. We use a CPU server with an Intel Xeon E5-2650 v4 (12 cores, 24 threads) and 64GB RAM to host the PostgreSQL database for experiments. Fine-tuning and deploying the LM are done on a machine with an Intel Xeon Gold 5218 CPU, 256GB RAM, and 4 NVIDIA GeForce RTX 3090 GPUs. The process of fine-tuning the LM takes approximately 10 hours. These machines are connected via high-speed local networks.

8.1.6 Evaluation Settings. The evaluation is conducted in three different settings: **In-Schema**: The LM is trained on our generated workloads and tested on the original workloads of 10 representative benchmarks. **Cross-Schema**: We employ a 5-fold cross-validation approach to simulate cross-schema scenarios. In each fold, we select one OLAP and one OLTP benchmark as the hold-out test set, training the LM on our generated samples from the remaining 8 benchmarks. This ensures there is no schema overlap between the training and testing sets, allowing us to evaluate E2ETUNE’s performance on new database instances. **Real-World**: The LM is trained using our generated data samples and directly applied to recommend configurations for three new real-world benchmarks.

8.2 Main Results

8.2.1 In-Schema Evaluation. Regarding the in-schema evaluation results shown in Figure 3 (a), we have the following findings:

E2ETUNE emerges as the fastest knob tuning method while delivering competitive database performance improvements. It achieves the shortest tuning time across all ten benchmarks, showcasing the advantages of end-to-end modeling. For instance, in the TPC-H benchmark, HEBO takes 1,381.7 minutes; even with workload mapping, it still requires 328.7 minutes. In contrast, E2ETUNE completes the task in just 19.8 minutes, achieving up to a 98.6% reduction in time ($\frac{1,381.7 - 19.8}{1,381.7} = 98.6\%$). Regarding performance, E2ETUNE identifies the best configurations in four benchmarks (JOB, SSB-flat, Twitter, Smallbank) and the second-best in five benchmarks (TPC-H, SSB, TPC-DS, TPC-C, Wikipedia).

E2ETUNE breaks its upper bound, HEBO. Despite the fact that the training labels (*i.e.*, promising configurations) for E2ETUNE are derived from HEBO, E2ETUNE surpasses HEBO’s performance across all benchmarks. This unexpected outcome can be attributed to E2ETUNE’s “sampling-then-ranking” inference strategy. The sampling introduces randomness, allowing the trained LM to explore a range of configurations around HEBO’s upper bound. This insight suggests that in the future, E2ETUNE could potentially be leveraged

⁴<https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2>

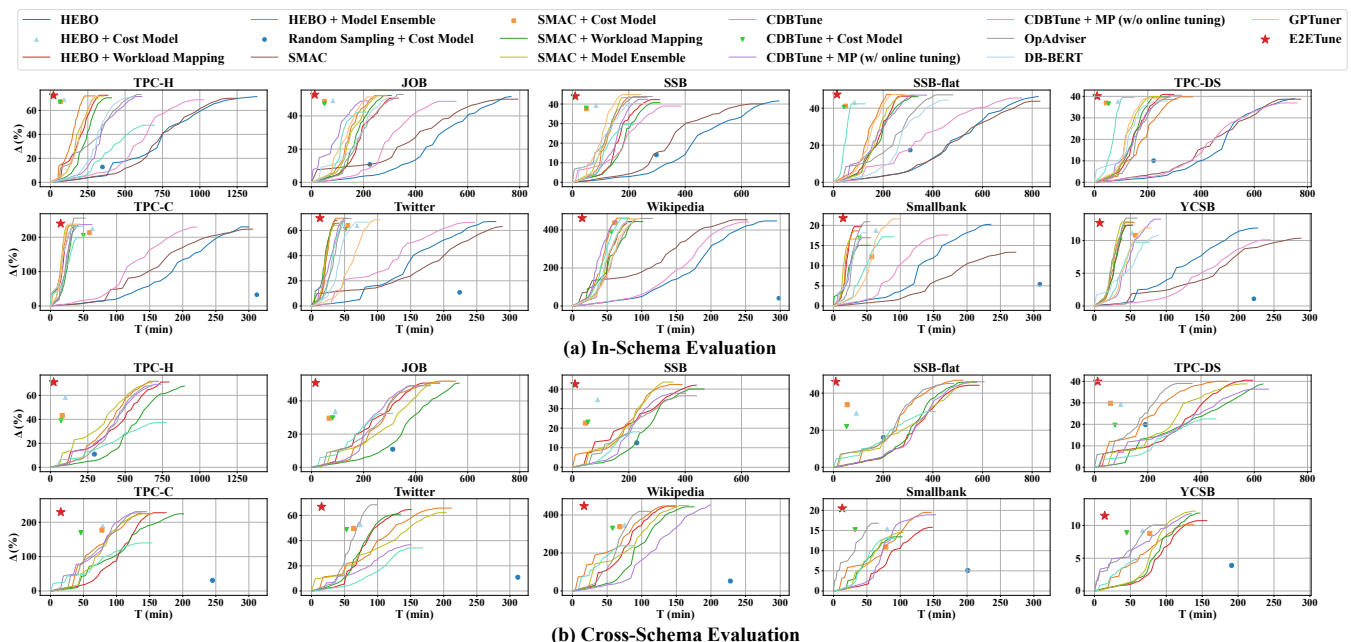


Figure 3: Best performance improvement over tuning time across 10 representative benchmarks. “MP” is the abbreviation of Model Pre-training. (top-left is better)

to replace HEBO in generating training labels, a concept known as “self-training” or “self-improvement” [28].

E2ETUNE surpasses other knowledge transfer methods that utilize historical tuning knowledge (refer to +Workload Mapping, +Model Ensemble, and +MP). While traditional knowledge transfer techniques can speed up the tuning process, they still require much longer time for tuning compared to E2ETUNE. These methods still rely on the iterative processes of traditional knob tuners. In contrast, E2ETUNE uses historical tuning data to train an end-to-end knob tuner, removing redundant iterative steps.

E2ETUNE outperforms approaches that also use LMs in knob tuning, such as DB-BERT [73] and GPTuner [36]. While these methods and our E2ETUNE both utilize LMs, they differ in their primary objectives. DB-BERT and GPTuner use LMs to constrain the search space for traditional knob tuners, thus they still necessitate numerous workload replays. In contrast, E2ETUNE trains a language model to directly predict suitable configurations. Consequently, E2ETUNE not only significantly accelerates the tuning process but also achieves comparable or superior performance enhancements compared to these methods.

E2ETUNE is faster and better than traditional methods assisted by the cost model (refer to +Cost Model). While our trained cost model can replace time-consuming workload execution, traditional methods still require a lot of additional time to update their models online and recommend new configurations with each iteration, resulting in low efficiency. Furthermore, the cost model can not always accurately reflect database performance, leading traditional methods to identify sub-optimal configurations. Moreover, even with random sampling of 1 million configurations,

the best option identified by the cost model often performs poorly, highlighting the vast search space inherent in the knob tuning task.

8.2.2 Cross-Schema Evaluation. The cross-schema scenario assesses how well a knowledge transfer method handles workloads from new database instances. Traditional knob tuners and two knowledge transfer methods (GPTuner and DB-BERT) exhibit the same behavior in both cross-schema and in-schema scenarios since they don’t use historical tuning tasks to initialize their models. As a result, these baselines are excluded from the cross-schema evaluation.

As illustrated in Figure 3 (b), in the cross-schema setting, E2ETUNE not only achieves the highest tuning efficiency but also identifies configurations that are comparable to, or even surpass, those of state-of-the-art approaches. Notably, E2ETUNE discovers the best configuration for the Smallbank benchmark and the second-best configurations for TPC-H, SSB, TPC-C, and Twitter benchmarks.

An interesting discovery is that existing knowledge transfer methods significantly expedite the tuning process and marginally improve database performance in the in-schema setting, but their effectiveness diminishes in the cross-schema environment. This decline can be attributed to several factors. Firstly, techniques such as workload mapping, model ensemble, and OpAdviser heavily rely on workload matching, which becomes challenging when finding similar workloads across different database instances. Additionally, the limited generalization capability of the model pre-training technique, due to the small size of the actor and critic neural networks, contributes to the reduced speedup and performance improvement in the cross-schema scenario.

In contrast, E2ETUNE consistently outperforms baseline methods. Its tuning speed is unaffected by the familiarity of database instances. Thanks to the generalization capabilities of the LM and

high-quality training data, E2ETUNE can effectively recommend good configurations for out-of-distribution workloads.

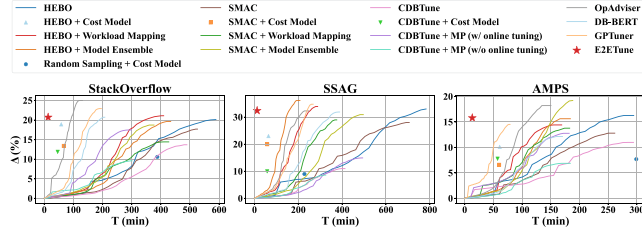


Figure 4: Maximum performance improvement over tuning time on three real-world benchmarks. (top-left is better)

8.2.3 Real-World Evaluation. We train E2ETUNE using training samples collected from 10 representative benchmarks and then directly apply it to 3 real-world benchmarks. Notably, for SSAG and AMPS, we implement all baseline methods and our E2ETUNE in ByteDance’s internal development environments to assess their effectiveness. These real-world benchmarks introduce several new challenges for E2ETUNE, including more complex queries, larger database scales, private database instances, and different deployment environments.

The experimental results are presented in Figure 4. In this highly challenging evaluation setting, E2ETUNE exhibits remarkable out-of-the-box capabilities, consistently identifying configurations comparable to those of state-of-the-art methods while significantly reducing the time required. Although E2ETUNE does not achieve the best configurations in this setting, it substantially saves time and resources on tuning tasks, making it more suitable for real-world large-scale deployments.

8.3 Ablation Study

8.3.1 Ablations on Input Features. As detailed in Section 6.1, the input of E2ETUNE includes workload statistics, query plans, and internal metrics. Ablation studies, shown in Table 2, reveal performance drops when any component is removed, highlighting their significance. Notably, omitting query plans greatly reduces performance in the cross-schema setting. Query plans are vital as they detail SQL operations and costs, reflecting the database’s data distribution, thus aiding the LM in generalizing across diverse database instances.

8.3.2 Ablations on LM Inference Strategy. As outlined in Section 7, we adopt a “sampling-then-ranking” strategy to recommend configurations. To evaluate the effectiveness of this inference strategy, we conduct an additional experiment in which we perform inference without utilizing the cost model. Specifically, using the fine-tuned language model, we only recommend a configuration through the greedy decoding method. The evaluation results of this experiment are presented in Table 2 (see “w/o sampling-then-ranking”). We can see that the “sampling-then-ranking” strategy yields slight improvements over the simple greedy decoding method.

8.3.3 Ablations on Output Knob Format. Section 6.2 discusses discretizing knob values into pre-defined buckets. To validate the

Table 2: Ablation studies of E2ETUNE. We present the average performance improvements Δ (%) \uparrow on OLAP and OLTP benchmarks. “IS” and “CS” represent “in-schema” and “cross-schema” settings, respectively.

	OLAP(IS)	OLTP(IS)	OLAP(CS)	OLTP(CS)
E2ETUNE	51.4	156.2	50.1	150.6
Input Features				
- w/o internal metrics	49.8	152.5	48.6	142.8
- w/o workload features	47.0	148.9	48.5	148.7
- w/o query plans	44.5	145.6	34.9	53.2
LM Inference Strategy				
- w/o sampling-then-ranking	50.1	152.8	47.0	144.3
Knob Output Format				
- specific values	15.1	45.3	9.4	36.8
LM Backbone				
- CodeLLaMA-7B [59]	51.3	152.7	50.5	152.8
- LLaMA2-7B [68]	50.4	152.4	50.1	153.7
- DeepSeekCoder-7B [23]	48.8	159.6	48.8	135.3
- DeepSeekCoder-1B [23]	42.0	142.4	41.0	116.1
- Mistral-7B w/o Pre-training	NA	NA	NA	NA
LM Learning Strategy				
- Few-shot Mistral-7B	33.8	114.2	28.6	109.5
- Few-shot GPT-4	35.3	115.4	30.9	116.4

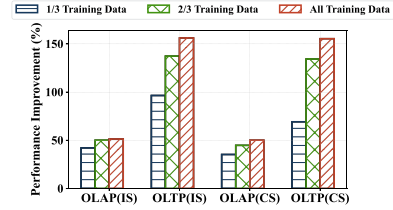


Figure 5: Ablation study of the scale of training data.

efficacy of this bucketing approach, we conduct an experiment where the LM directly output knob values (refer to “specific values” in Table 2). The results clearly indicate a significant performance decline when directly outputting values. The poor performance can be attributed to the extensive range and substantial variation in permissible values across different knobs, which poses significant challenges for LMs when tasked with directly outputting knob values. This highlights the effectiveness of the bucketing strategy.

8.3.4 Ablations on LM Backbone. We utilize Mistral-7B as the base model of E2ETUNE. To evaluate the impact of different base models on knob tuning performance, we replace Mistral-7B with other robust models such as CodeLLaMA-7B, LLaMA2-7B, DeepSeekCoder-7B, and DeepSeekCoder-1B to explore model scale effects. Results in Table 2 show that model performance varies by scale and setting: Mistral-7B and DeepSeekCoder-7B excel in OLAP and OLTP benchmarks in the in-schema setting, while CodeLLaMA-7B and LLaMA2-7B perform best in the cross-schema setting. This performance disparity is likely due to differences in pre-training corpora, indicating that a more powerful base model can enhance E2ETUNE’s performance. The 7B version of DeepSeekCoder significantly outperforms the 1B version, highlighting the benefits of larger models.

To explore the impact of the pre-training phase on knob tuning, we test a new Mistral-7B model with randomly initialized weights followed by fine-tuning on our data. This model produce unparseable outputs, underscoring the crucial role of pre-training.

8.3.5 Ablations on Scale of Training Data. We delve deeper into the effects of the training data scale on the quality of configurations recommended by the LM. Following the evaluation settings outlined in Section 8.1.6, we modify the scale of the training dataset for our analysis. Our experiments are conducted across three different scales: 1/3, 2/3, and the complete training dataset, with the results visualized in Figure 5. The findings illustrate a notable positive relationship between the size of the training data and the enhancement in configuration quality suggested by the LM. It is evident that this trend persists even when utilizing the entire training set, suggesting that the performance of E2ETUNE can be further optimized with an expanded training dataset.

8.3.6 Ablations on LM Learning Strategy. Besides fine-tuning the LM, another approach is few-shot in-context learning. This method uses a small number of task-specific demonstrations within the LM’s context, avoiding the need for extensive fine-tuning. We randomly select 3 samples ($\langle \text{workload features, suitable configuration} \rangle$ pairs) from the training set to use as demonstrations. The original Mistral-7B and GPT-4 models are then instructed to recommend configurations for new workloads using these demonstration prompts. Results in Table 2 (Few-shot Mistral-7B and GPT-4) show that few-shot learning is less effective than full fine-tuning. Even with GPT-4, the tuning performance is below the proposed E2ETUNE, due to lower demonstration utilization efficiency.

8.3.7 Ablations on Modeling Algorithms. In E2ETUNE, we utilize a language model to capture the distribution mapping from workload features to their corresponding promising configurations. In this ablation study, we aim to investigate whether traditional machine learning models can also effectively capture this intricate distribution mapping. We evaluate two commonly used machine learning approaches: random forests and multi-layer perceptrons (MLP). To ensure a fair comparison, we maintain consistency in the input and output information with E2ETUNE. However, some adjustments should be made to accommodate these machine learning techniques. Workload features are represented as a vector that combines workload statistics, query plan features, and internal metrics. Query plan features are extracted using the methodology proposed in [39]. Then, as we discretize the knob values into buckets for E2ETUNE, we can reconstruct the output into classification labels for traditional machine learning methods, where each label corresponds to a specific bucket identifier. In the case of the random forest, we specify 1000 trees and a maximum depth of 50. For the multi-layer perceptron, a neural network with three hidden layers, each comprising 128 dimensions, is employed. To mitigate overfitting, batch normalization and dropout layers are incorporated. Given the varying feature lengths across database instances, a separate model needs to be trained for each new instance. Our evaluations are conducted on the TPC-H and TPC-C benchmarks.

Results from the TPC-H benchmark show that the random forest and MLP models yield performance improvements of 36.8% and 51.3% respectively, significantly lower than the 72.5% enhancement achieved by E2ETUNE. Similarly, on the TPC-C benchmark, the random forest and MLP models exhibit performance gains of 128.8% and 177.5% respectively, while E2ETUNE outperforms them with a remarkable 239.9% improvement. These findings reveal that despite leveraging high-quality input features, traditional machine learning

methods struggle to capture the intricate distribution mapping due to their limited modeling and generalization capabilities.

8.3.8 Ablations on Training Workload Generation Strategy. In the database field, a common approach to generating workload queries is to use predefined templates [47, 50, 84, 100]. However, we believe this strategy limits the diversity of the generated queries, thereby constraining the generalizability of the trained models. In this section, we conduct an ablation study on the JOB benchmark to support this hypothesis. Specifically, following the Cardinality Estimation Benchmark (CEB) [50], we employ 13 templates designed for the JOB’s database instance, resulting in approximately 1,100 SQL queries combined into 299 distinct workloads. We then collect labels (*i.e.*, promising configurations) for these new template-based workloads using the framework introduced in Section 5.3. Next, we train two language models using training samples from GPT-4-Turbo-generated workloads and template-generated workloads, denoted as M_{ours} and M_{CEB} , respectively. Finally, we utilize both M_{ours} and M_{CEB} to recommend configurations for the original JOB benchmark workload. The results reveal that the configurations recommended by M_{ours} yield a 50.26% performance improvement, while those from M_{CEB} result in only a 38.56% improvement. This discrepancy highlights the importance of query diversity in training a highly generalizable language model for knob tuning.

8.4 Evaluation of Training Data

The quality of E2ETUNE’s training dataset is crucial. Each training sample consists of a $\langle \text{workload, promising configuration} \rangle$ pair, crafted using the framework outlined in Section 5. We evaluate the quality of these components individually.

8.4.1 Quality of Workloads. OLAP Workloads. The OLAP workloads used for training are generated by GPT-4-Turbo. In Table 3, we provide a detailed statistical analysis of both GPT-generated and benchmarked SQL queries across five aspects: the number of referenced tables per SQL query, the number of joined tables per SQL query, the number of referenced keywords per SQL query, the number of predicates per SQL query, and the number of distinct query templates. Our analysis reveals that GPT-generated queries not only effectively cover the benchmarked queries but also significantly enhance the diversity of query templates, underscoring the quality of the queries produced by GPT-4-Turbo. **OLTP Workloads.** The OLTP workloads are generated by randomly assigning weights to predefined transactions, ensuring high quality and diversity.

8.4.2 Quality of Labels. In this work, two key factors affect the quality of labels: the knob tuning method and the cost model.

Tuning Method. Our primary objective in choosing a tuning method is to select a powerful and easily deployable knob tuner. In this regard, we assess three potential methods: HEBO, SMAC, and CDBTune. Our experimental findings, detailed in Figure 3, demonstrate that HEBO surpasses the other two methods in performance. Consequently, we adopt HEBO to gather labels.

Cost Model. The quality of the cost model significantly impacts label quality, as a substantial portion of the training set is derived from it. To provide a comprehensive evaluation of the cost model, we examine two critical aspects: (1) the accuracy of the cost model and (2) the performance of HEBO assisted by the cost model. First,

Table 3: A statistical analysis of SQL queries from two sources: those generated by GPT-4-Turbo (denoted as GPT) and those from the benchmark (denoted as Bench). For each statistic, except for the number of distinct templates, the minimum, maximum, and average values are presented as “MIN / MAX / AVG”.

Statistics	TPC-H		JOB		TPC-DS		SSB		SSB-flat	
	GPT	Bench	GPT	Bench	GPT	Bench	GPT	Bench	GPT	Bench
# Referenced Tables per SQL	1 / 8 / 5.43	1 / 8 / 3.41	1 / 13 / 5.88	4 / 15 / 8.51	1 / 11 / 5.46	1 / 13 / 5.63	2 / 5 / 3.93	2 / 5 / 3.76	1 / 1 / 1.0	1 / 1 / 1.0
# Joined Tables per SQL	1 / 8 / 5.21	1 / 8 / 2.46	1 / 12 / 5.26	4 / 15 / 8.50	1 / 11 / 5.08	1 / 10 / 2.68	1 / 5 / 3.91	2 / 5 / 3.76	1 / 1 / 1.0	1 / 1 / 1.0
# Referenced Keywords per SQL	14 / 34 / 21.96	14 / 29 / 20.27	12 / 40 / 26.62	11 / 27 / 18.24	13 / 42 / 25.41	12 / 38 / 23.44	14 / 33 / 21.79	13 / 19 / 15.77	15 / 33 / 22.74	12 / 16 / 14.15
# Predicates per SQL	3 / 40 / 15.57	1 / 18 / 6.00	1 / 43 / 16.57	14 / 91 / 39.30	3 / 127 / 17.04	0 / 96 / 13.05	4 / 46 / 14.33	8 / 20 / 13.31	1 / 26 / 9.75	7 / 16 / 11.31
# Distinct Templates	1099	21	1099	99	1088	49	1094	13	1100	11

we perform 10-fold cross-validation to train the cost model. For each fold, we use the coefficient of determination (R^2) to evaluate the cost model’s accuracy on the held-out test set: $R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$, where y_i is the observed performance metric for the i -th sample, \hat{y}_i is the predicted performance metric, \bar{y} is the mean of all observed metrics, and n is the total test set observations. Our cost model achieves an average R^2 score of 0.862 across the 10-fold validation, indicating high reliability. In addition, during label collection, the cost model is trained using observations from all database instances. Therefore, the quality of the configurations gathered by HEBO with the cost model can be observed in the in-schema part of “HEBO + Cost Model” in Figure 3. Compared with the original HEBO, using the cost model only leads to a marginal decrease in database performance, affirming the reliability of the cost model.

8.5 Data Collection Cost

To provide clarity on the expenses associated with collecting the complete training dataset, we compute the time for each stage detailed in Section 5, encompassing both workload generation and label collection times. Importantly, it should be emphasized that the training data collection process can be carried out offline.

8.5.1 Workload Generation. For OLAP workloads, we employ GPT-4-Turbo to craft approximately 1,100 OLAP queries per database instance. With a response time of approximately 20 seconds per query and five database instances, the cumulative query generation time amounts to roughly 31 hours, costing approximately \$150. In contrast, generating OLTP workloads incurs no additional expenses.

8.5.2 Label Collection. The training labels of E2ETUNE are sourced from HEBO with real execution and HEBO with the cost model.

Collecting Data via Real Execution. For OLAP workloads, executing the full workload takes between 1 to 20 minutes. Since the original HEBO method requires replaying the workload in each iteration, tuning an OLAP workload across 100 iterations may take 2 to 30 hours. For OLTP workloads, we use a 1-minute stress test for each tuning iteration. As a result, the complete tuning process for an OLTP workload takes approximately 2 hours. To ensure accurate test results, workloads are tuned sequentially on the server. With each database instance tuning 13 workloads, the full process takes around 30 days on a single machine. In practice, we use eight identical CPU servers to expedite this process.

Collecting Data via Cost Model. The results presented in Figure 3 demonstrate that the cost model can significantly enhance tuning efficiency when compared to actual executions. This efficiency boost facilitates the gathering of a substantial volume of

training data for E2ETUNE. Moreover, the elimination of the need for database testing allows for the simultaneous tuning of multiple workloads on a single machine. Leveraging the cost model, despite each database instance needing to tune more than 250 workloads, we can complete this task within 40 days using just one machine. We also utilize eight identical CPU servers to accelerate the process.

9 DISCUSSION

In this section, we first discuss the applicable scenarios for E2ETUNE. By leveraging the advantages of an end-to-end design, E2ETUNE significantly enhances tuning efficiency while recommending promising configurations. When encountering a new database engine, it is necessary to re-collect training data and train a new model to tune knobs due to different knobs. Consequently, E2ETUNE is particularly well-suited for stable or enumerable environments, such as cloud services that maintain uniform hardware configurations and standardized DBMS products. After the initial training, E2ETUNE proves highly effective in recommending configurations for a diverse range of users utilizing cloud databases. We are deploying E2ETUNE in real-world applications at ByteDance and will open-source our fine-tuned model to support further research. Additionally, to accelerate data collection and enhance label quality, we can explore and leverage better hyperparameter optimization algorithms, such as HPFSO [3], SPBOpt [61], and DRE [33], as replacements for HEBO in our data collection framework.

10 CONCLUSION

This paper explores an innovative method for end-to-end database knob tuning. Our emphasis is on utilizing the advanced language model (LM) to accurately capture the complex mapping between the workload and its promising configuration. To achieve this goal, we introduce a data generation framework to automatically produce workloads and their promising configurations, serving as training data samples for the LM. Through extensive experiments on OLAP and OLTP benchmarks, we demonstrate that E2ETUNE not only significantly improves the tuning efficiency over existing approaches, but also showcases notable database performance improvements in both in-schema and cross-schema settings.

ACKNOWLEDGMENTS

This work is supported by the National Key Research & Development Plan of China (2023YFF0725100) and the National Natural Science Foundation of China (62322214, U23A20299, U24B20144, 62172424, 62276270). We also acknowledge the support of the Public Computing Cloud, Renmin University of China.

REFERENCES

- [1] Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman P. Amarasinghe. 2014. OpenTuner: an extensible framework for program autotuning. In *International Conference on Parallel Architectures and Compilation, PACT '14, Edmonton, AB, Canada, August 24-27, 2014*. ACM, 303–316.
- [2] Jason Ansel, Edward Z. Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, C. K. Luk, Bert Maher, Yunjie Pan, Christian Puhres, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Shunting Zhang, Michael Suo, Phil Tillet, Xu Zhao, Eikan Wang, Keren Zhou, Richard Zou, Xiaodong Wang, Ajit Mathews, William Wen, Gregory Chanan, Peng Wu, and Soumith Chintala. 2024. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS 2024, La Jolla, CA, USA, 27 April 2024 - 1 May 2024*. ACM, 929–947.
- [3] Carlos Ansótegui, Meinolf Sellmann, Tapan Shah, and Kevin Tierney. 2021. Learning to Optimize Black-Box Functions with Extreme Limits on the Number of Function Evaluations. In *Learning and Intelligent Optimization: 15th International Conference, LION 15, Athens, Greece, June 20–25, 2021, Revised Selected Papers 15*. Springer, 7–24.
- [4] Anthropic. 2024. Introducing the next generation of Claude. (2024). Available at: <https://www.anthropic.com/news/claude-3-family>.
- [5] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojell, 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).
- [6] 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.
- [7] 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).
- [8] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott M. Lundberg, Harsha Nori, Hamid Palangi, Marco Túlio Ribeiro, and Yi Zhang. 2023. Sparks of Artificial General Intelligence: Early experiments with GPT-4. *CoRR abs/2303.12712* (2023). <https://doi.org/10.48550/ARXIV.2303.12712>
- [9] 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.
- [10] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, et al. 2021. Evaluating Large Language Models Trained on Code. *CoRR abs/2107.03374* (2021). [arXiv:2107.03374](https://arxiv.org/abs/2107.03374)
- [11] Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan R. Routledge, and William Yang Wang. 2021. FinQA: A Dataset of Numerical Reasoning over Financial Data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.)*. Association for Computational Linguistics, 3697–3711. <https://doi.org/10.18653/V1/2021.EMNLP-MAIN.300>
- [12] Lixue Cheng, Ziyi Yang, Changyu Hsieh, Benben Liao, and Shengyu Zhang. 2022. ODBO: Bayesian optimization with search space prescreening for directed protein evolution. *arXiv preprint arXiv:2205.09548* (2022).
- [13] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. PaLM: Scaling Language Modeling with Pathways. *CoRR abs/2204.02311* (2022). [arXiv:2204.02311](https://arxiv.org/abs/2204.02311)
- [14] Alibaba Cloud. 2024. Qwen2 Github. (2024). <https://github.com/QwenLM/Qwen2>.
- [15] 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.
- [16] Benoît Dageville and Mohamed Zait. 2002. SQL Memory Management in Oracle9i. In *Proceedings of 28th International Conference on Very Large Data Bases, VLDB 2002, Hong Kong, August 20-23, 2002*. Morgan Kaufmann, 962–973.
- [17] Tri Dao. 2023. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. *CoRR abs/2307.08691* (2023). <https://doi.org/10.48550/ARXIV.2307.08691>
- [18] Djellal Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudre-Mauroux. 2013. Oltp-bench: An extensible testbed for benchmarking relational databases. *Proceedings of the VLDB Endowment* 7, 4 (2013), 277–288.
- [19] Songyun Duan, Vamsidhar Thummala, and Shvinnath Babu. 2009. Tuning database configuration parameters with ituned. *Proceedings of the VLDB Endowment* 2, 1 (2009), 1246–1257.
- [20] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).
- [21] 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).
- [22] 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.
- [23] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. 2024. DeepSeek-Coder: When the Large Language Model Meets Programming—The Rise of Code Intelligence. *arXiv preprint arXiv:2401.14196* (2024).
- [24] Jonathan Herzig, Paweł Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. TaPas: Weakly Supervised Table Parsing via Pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (Eds.)*. Association for Computational Linguistics, 4320–4333. <https://doi.org/10.18653/V1/2020.ACL-MAIN.398>
- [25] Djoerd Hiemstra. 2000. A probabilistic justification for using tf x idf term weighting in information retrieval. *Int. J. Digit. Libr.* 3, 2 (2000), 131–139. <https://doi.org/10.1007/S007999900025>
- [26] Benjamin Hilprecht and Carsten Binnig. 2021. One model to rule them all: towards zero-shot learning for databases. *arXiv preprint arXiv:2105.00642* (2021).
- [27] Dong Huang, Qingwen Bu, Jie M. Zhang, Michael Luck, and Heming Cui. 2023. AgentCoder: Multi-Agent-based Code Generation with Iterative Testing and Optimisation. *CoRR abs/2312.13010* (2023). [arXiv:2312.13010](https://arxiv.org/abs/2312.13010)
- [28] Jiaxin Huang, Shixiang Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. 2023. Large Language Models Can Self-Improve. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*. Association for Computational Linguistics, 1051–1068.
- [29] 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.
- [30] 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).
- [31] Zhengbao Jiang, Yi Mao, Pengcheng He, Graham Neubig, and Weizhu Chen. 2022. OmniTab: Pretraining with Natural and Synthetic Data for Few-shot Table-based Question Answering. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2022, Seattle, WA, United States, July 10-15, 2022*. Association for Computational Linguistics, 932–942.
- [32] 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).
- [33] Abdus Salam Khazi, Sebastian Pineda Arango, and Josif Grabocka. 2023. Deep ranking ensembles for hyperparameter optimization. *arXiv preprint arXiv:2303.15212* (2023).
- [34] Mayuresh Kunjir and Shvinnath 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.
- [35] 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.
- [36] Jiale Lao, Yibo Wang, Yufei Li, Jianping Wang, Yunjia Zhang, Zhiyuan Cheng, Wanguo 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).
- [37] 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.
- [38] Stefan Lessmann, Robert Stahlbock, and Sven F Crone. 2005. Optimizing hyperparameters of support vector machines by genetic algorithms. In *IC-AI, Vol. 74*. 82.

- [39] 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.
- [40] 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.
- [41] Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024. CodeS: Towards Building Open-source Language Models for Text-to-SQL. *CoRR abs/2402.16347* (2024). arXiv:2402.16347
- [42] Jinyang Li, Binyuan Hui, Ge Qu, Jiayi 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).
- [43] Yujia Li, David H. Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustín Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d’Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. 2022. Competition-Level Code Generation with AlphaCode. *CoRR abs/2203.07814* (2022). <https://doi.org/10.48550/ARXIV.2203.07814> arXiv:2203.07814
- [44] Marius Lindauer, Katharina Eggenberger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Rühkopf, 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.
- [45] Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. 2023. WizardMath: Empowering Mathematical Reasoning for Large Language Models via Reinforced Evol-Instruct. *CoRR abs/2308.09583* (2023). <https://doi.org/10.48550/ARXIV.2308.09583> arXiv:2308.09583
- [46] 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).
- [47] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2021. Bao: Making Learned Query Optimization Practical. In *SIGMOD ’21: International Conference on Management of Data, Virtual Event, China, June 20–25, 2021*. ACM, 1275–1288.
- [48] Meta. 2024. Introducing Meta Llama 3: The most capable openly available LLM to date. (2024). <https://ai.meta.com/blog/meta-llama-3/>.
- [49] Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. 2022. Cross-Task Generalization via Natural Language Crowdsourcing Instructions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, ACL 2022, Dublin, Ireland, May 22–27, 2022, Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, 3470–3487. <https://doi.org/10.18653/V1/2022.ACL-LONG.244>
- [50] Parimarjan Negi, Ryan Marcus, Andreas Kipf, Hongzi Mao, Nesime Tatbul, Tim Kraska, and Mohammad Alizadeh. 2021. Flow-loss: learning cardinality estimates that matter. *Proc. VLDB Endow.* 14, 11 (July 2021), 2019–2032. <https://doi.org/10.14778/3476249.3476259>
- [51] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]
- [52] OpenAI. 2024. GPT-4 Turbo and GPT-4. (2024). <https://platform.openai.com/docs/models/gpt-4-turbo-and-gpt-4>.
- [53] Patrick E O’Neil, Elizabeth J O’Neil, and Xuedong Chen. 2007. The star schema benchmark (SSB). *Pat* 200, 0 (2007), 50.
- [54] 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.
- [55] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.). http://papers.nips.cc/paper_files/paper/2023/hash/72223cc6f63ca1aa59edaec1b3670e6-Abstract-Conference.html
- [56] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihao Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. *CoRR abs/2307.16789* (2023). arXiv:2307.16789
- [57] Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2024. Tool Learning with Large Language Models: A Survey. *CoRR abs/2405.17935* (2024). arXiv:2405.17935
- [58] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. ZeRO: memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9–19, 2020*. IEEE/ACM, 20.
- [59] 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).
- [60] Victor Sanh, Albert Webson, Colin Raffel, Stephen H. Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, et al. 2022. Multitask Prompted Training Enables Zero-Shot Task Generalization. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25–29, 2022*. OpenReview.net.
- [61] Mikita Sazanovich, Anastasiya Nikolskaya, Yuri Belousov, and Aleksei Shipilman. 2021. Solving black-box optimization challenge via learning search space partition for local Bayesian optimization. In *NeurIPS 2020 Competition and Demonstration Track*. PMLR, 77–85.
- [62] Murray Shanahan. 2024. Talking about large language models. *Commun. ACM* 67, 2 (2024), 68–79.
- [63] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *CoRR abs/2402.03300* (2024). <https://doi.org/10.48550/ARXIV.2402.03300> arXiv:2402.03300
- [64] 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.
- [65] Sahaana Suri, Ihab F. Ilyas, Christopher Ré, and Theodoros Rekatsinas. 2021. Ember: No-Code Context Enrichment via Similarity-Based Keyless Joins. *Proc. VLDB Endow.* 15, 3 (2021), 699–712.
- [66] 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.
- [67] Nan Tang, Ju Fan, Fangyi Li, Jianhong Tu, Xiaoyong Du, Guoliang Li, Samuel Madden, and Mourad Ouzzani. 2021. RPT: Relational Pre-trained Transformer Is Almost All You Need towards Democratizing Data Preparation. *Proc. VLDB Endow.* 14, 8 (2021), 1254–1261.
- [68] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shriti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models, 2023. URL <https://arxiv.org/abs/2307.09288> (2023).
- [69] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shriti Bhosale, Dan Bikel, Lukas Blecher, et al. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. *CoRR abs/2307.09288* (2023). arXiv:2307.09288
- [70] Transaction Processing Performance Council (TPC). [n.d.]. TPC-DS: Decision Support Benchmark. Online. Available: <http://www.tpc.org/tpcds/>.
- [71] Transaction Processing Performance Council (TPC). Year of the specification version. TPC-H Benchmark Specification. Online. Available: <http://www.tpc.org/tpch/>.
- [72] 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.
- [73] 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.
- [74] Immanuel Trummer. 2023. Can Large Language Models Predict Data Correlations from Column Names? *Proceedings of the VLDB Endowment* 16, 13 (2023), 4310–4323.
- [75] 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.
- [76] 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).
- [77] 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.
- [78] Dana Van Aken, Dongsheng Yang, Sebastian 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.
- [79] 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).
- [80] Xingchen Wan, Vu Nguyen, Huong Ha, Binxin Ru, Cong Lu, and Michael A Osborne. 2021. Think global and act local: Bayesian optimisation over high-dimensional categorical and mixed search spaces. *arXiv preprint arXiv:2102.07188* (2021).

- [81] Junxiong Wang, Immanuel Trummer, and Debabrota Basu. 2021. UDO: universal database optimization using reinforcement learning. *arXiv preprint arXiv:2104.01744* (2021).
- [82] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.
- [83] 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).
- [84] Zongheng Yang, Wei-Lin Chiang, Sifei Luan, Gautam Mittal, Michael Luo, and Ion Stoica. 2022. Balsa: Learning a Query Optimizer Without Expert Demonstrations. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, Zachary G. Ives, Angela Bonifati, and Amr El Abbadi (Eds.). ACM, 931–944. <https://doi.org/10.1145/3514221.3517885>
- [85] Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023. Large Language Models are Versatile Decomposers: Decomposing Evidence and Questions for Table-based Reasoning. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2023, Taipei, Taiwan, July 23-27, 2023*. ACM, 174–184.
- [86] 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).
- [87] 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.
- [88] 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.
- [89] 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.
- [90] 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.
- [91] 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.
- [92] Xinyang Zhao, Xuanhe Zhou, and Guoliang Li. 2023. Automatic Database Knob Tuning: A Survey. *IEEE Transactions on Knowledge and Data Engineering* (2023).
- [93] Yue Zhao, Gao Cong, Jiachen Shi, and Chunyan Miao. 2022. QueryFormer: A Tree Transformer Model for Query Plan Representation. *Proc. VLDB Endow.* 15, 8 (2022), 1658–1670.
- [94] 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).
- [95] Aojun Zhou, Ke Wang, Zimu Lu, Weikang Shi, Sichun Luo, Zipeng Qin, Shaoqing Lu, Anya Jia, Linqi Song, Mingjie Zhan, and Hongsheng Li. 2023. Solving Challenging Math Word Problems Using GPT-4 Code Interpreter with Code-based Self-Verification. *CoRR* abs/2308.07921 (2023). [arXiv:2308.07921](https://arxiv.org/abs/2308.07921)
- [96] 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.
- [97] Xuanhe Zhou, Guoliang Li, and Zhiyuan Liu. 2023. Llm as dba. *arXiv preprint arXiv:2308.05481* (2023).
- [98] Xuanhe Zhou, Guoliang Li, Zhaoyan Sun, Zhiyuan Liu, Weize Chen, Jianming Wu, Jiesi Liu, Ruohang Feng, and Guoyang Zeng. 2023. D-Bot: Database Diagnosis System using Large Language Models. *CoRR* abs/2312.01454 (2023). <https://doi.org/10.48550/ARXIV.2312.01454> [arXiv:2312.01454](https://arxiv.org/abs/2312.01454)
- [99] Xuanhe Zhou, Zhaoyan Sun, and Guoliang Li. 2024. DB-GPT: Large Language Model Meets Database. *Data Science and Engineering* (2024), 1–10.
- [100] Rong Zhu, Wei Chen, Bolin Ding, Xingguang Chen, Andreas Pfadler, Ziniu Wu, and Jingren Zhou. 2023. Lero: A Learning-to-Rank Query Optimizer. *Proc. VLDB Endow.* 16, 6 (2023), 1466–1479.