AgentTune: An Agent-Based Large Language Model Framework for Database Knob Tuning

YIYAN LI, Renmin University of China, China and Key Laboratory of Data Engineering and Knowledge Engineering, MOE, China
HAOYANG LI, Renmin University of China, China
JING ZHANG, Renmin University of China, China
RENATA BOROVICA-GAJIC, University of Melbourne, Australia
SHUAI WANG, ByteDance Inc., China
TIEYING ZHANG, ByteDance Inc., United States
JIANJUN CHEN, ByteDance Inc., United States
RUI SHI, ByteDance Inc., China
CUIPING LI, Renmin University of China, China and Engineering Research Center of Database and Business Intelligence, MOE, China
HONG CHEN, Renmin University of China, China and Key Laboratory of Data Engineering and Knowledge Engineering, MOE, China

Database knob tuning is a long-standing challenge in the database community, aimed at enhancing the performance of database management systems (DBMSs) by minimizing latency and maximizing throughput. Manual tuning, which relies heavily on human expertise, is often inefficient and impractical for large-scale or dynamic deployments. Recent work has explored automating this process using machine learning (ML) and large language models (LLMs). However, existing methods typically require hundreds of workload replays or rely on extensive training data, leading to low tuning efficiency or high preparation costs. Moreover, they also risk generating invalid configurations that can degrade performance or even crash the database. To address these limitations, we introduce **AgentTune**, the first agent-based knob tuning framework powered by LLMs, designed for efficiency, adaptability, and reliability. AgentTune decomposes the tuning process into four specialized agents: *Workload Analyzer*, *Knob Selector*, *Range Pruner*, and *Configuration Recommender*, each responsible for a distinct subtask. These agents collaborate through structured prompt chaining. AgentTune first analyzes the input workload to identify impactful knobs and reconstructs their valid ranges to reduce the

Yiyan Li and Haoyang Li contributed equally to this work.
Cuiping Li is the corresponding author.

Authors' Contact Information: Yiyan Li, Renmin University of China, Beijing, China and Key Laboratory of Data Engineering and Knowledge Engineering, MOE, Beijing, China, liyiyan@ruc.edu.cn; Haoyang Li, Renmin University of China, Beijing, China, lihaoyang.cs@ruc.edu.cn; Jing Zhang, Renmin University of China, Beijing, China, zhang-jing@ruc.edu.cn; Renata Borovica-Gajic, University of Melbourne, Melbourne, Australia, renata.borovica@unimelb.edu.au; Shuai Wang, ByteDance Inc., Beijing, China, wangshuai.will@bytedance.com; Tieying Zhang, ByteDance Inc., San Jose, United States, tieying.zhang@bytedance.com; Jianjun Chen, ByteDance Inc., San Jose, United States, jianjun.chen@bytedance.com; Rui Shi, ByteDance Inc., Beijing, China, shirui@bytedance.com; Cuiping Li, Renmin University of China, Beijing, China and Engineering Research Center of Database and Business Intelligence, MOE, Beijing, China, chong@ruc.edu.cn, licuiping@ruc.edu.cn; Hong Chen, Renmin University of China, Beijing, China and Key Laboratory of Data Engineering and Knowledge Engineering, MOE, Beijing, China, chong@ruc.edu.cn.

ACM 2836-6573/2025/12-ART293
https://doi.org/10.1145/3769758

search space. It then employs a tree-based search strategy to efficiently explore the configuration space and identify suitable knob values.

We conduct extensive evaluations across diverse workloads (public benchmarks and real-world workloads), metrics (latency and throughput), DBMSs (PostgreSQL, MySQL, and TiDB), hardware environments, and database scales. Experimental results demonstrate that, compared to existing baselines, AgentTune is able to identify superior configurations using significantly fewer workload replays. Furthermore, AgentTune rarely generates invalid configurations during the tuning process, ensuring reliability and suitability for real-world deployments.

CCS Concepts: • **Information systems → Autonomous database administration**; **Database utilities and tools**.

Additional Key Words and Phrases: Database Knob Tuning, Large Language Models

ACM Reference Format:
Yiyan Li, Haoyang Li, Jing Zhang, Renata Borovica-Gajic, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, Cuiping Li, and Hong Chen. 2025. AgentTune: An Agent-Based Large Language Model Framework for Database Knob Tuning . *Proc. ACM Manag. Data* 3, 6 (SIGMOD), Article 293 (December 2025), 29 pages. https://doi.org/10.1145/3769758

1 Introduction

Modern database management systems (DBMSs) typically expose hundreds of tunable knobs that govern various aspects of system behavior, including memory allocation, thread scheduling, and caching mechanisms [6]. The goal of database knob tuning is to discover optimal configurations of these knobs to enhance DBMS performance, achieving low latency and high throughput. However, this task is inherently challenging due to the continuous and high-dimensional nature of most knobs, which results in a vast configuration search space. Consequently, it is difficult for database administrators (DBAs) to efficiently determine high-performing configurations for a specific workload [28, 44].

Over the past decade, various machine learning (ML)-based knob tuning systems have emerged, capable of automatically discovering effective configurations without extensive human intervention. These include approaches based on Reinforcement Learning [28, 65] and Bayesian Optimization [2, 26, 67] techniques. Such systems generally follow a "try-collect-adjust" loop: given a workload, the ML model suggests a configuration, which is then applied to the DBMS and evaluated through performance metrics (e.g., latency or throughput). The model then updates itself using this feedback and produces a new configuration. Through multiple iterations, such methods can converge to a configuration that significantly improves DBMS performance.

Recently, the field has seen growing interest in applying large language models (LLMs) to database knob tuning [14, 17, 23, 26, 32]. A recent survey [32] highlights the impressive potential of LLMs in this domain. Existing LLM-based approaches fall into two broad categories: (1) using LLMs to assist traditional ML-based methods [14, 26], such as by reducing the knob space or guiding the sampling process; and (2) using LLMs as end-to-end knob tuners [17, 23], directly recommending configurations for the given workload.

Limitations of Existing Methods. While these methods have demonstrated promise, several critical limitations hinder their effectiveness in large-scale real-world applications:

(L1) Low tuning efficiency. ML-based methods [2, 65] often require hundreds of iterations to converge to a promising configuration. Each iteration involves executing the workload on the DBMS (i.e., workload replay), resulting in tuning processes that may span several hours. Although LLM-assisted approaches achieve faster convergence, they remain limited by the inherent inefficiencies of ML-based knob tuners.

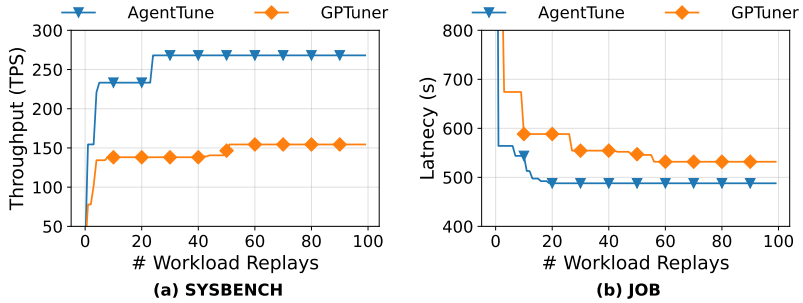

Fig. 1. Performance comparison between AgentTune and GPTuner (a state-of-the-art LLM-assisted knob tuner).

(L2) Low reliability. Although ML-based tuning balances exploration and exploitation to avoid local optima, it frequently produces invalid configurations - those that degrade performance below the default or even crash the DBMS. Similarly, our preliminary studies show that directly using LLMs for knob recommendations can lead to unsafe values (e.g., overly small buffer pool sizes), stemming from issues like hallucination [22] and limited numerical reasoning capabilities [39]. These problems can result in resource exhaustion or system instability during the tuning process.

(L3) Resource-intensive preparations. To enhance efficiency, many ML and LLM-based approaches utilize historical tuning data for model initialization [2, 68], reducing the configuration space [69], or fine-tuning end-to-end knob tuners [23]. However, collecting this data is time and resource-intensive. For instance, OtterTune [2] requires "over 30k trials per DBMS", and E2ETune [23] needs nearly 3k <workload, suitable configuration> samples. Gathering this data involves running workloads on real machines, taking several days to weeks. Moreover, pre-trained models often struggle to generalize to new environments (e.g., different hardware or DBMS engines) due to data distribution shifts [12, 18, 27, 57], necessitating costly re-training for adaptation.

Motivation. In contrast to these automated methods, experienced DBAs typically approach knob tuning in a step-by-step manner, breaking down the complex task into simpler subtasks. Specifically, given a workload, they first analyze workload characteristics (e.g., data volume, read/write ratio), identify relevant knobs, determine valid knob ranges based on hardware constraints, and apply configurations informed by experience. They then monitor internal DBMS metrics and iteratively refine configurations based on observed feedback. This structured, feedback-driven process enables DBAs to reach satisfactory configurations in just a few steps.

Recent advancements in LLMs such as GPT-4 [40] and Claude [4] have revolutionized natural language processing. Pre-trained on vast corpora using the Transformer architecture [59], these models exhibit remarkable capabilities across a range of complex tasks, such as text-to-SQL translation [16, 29, 30, 46], code generation [8, 31], and tool usage [47, 50]. These advancements have sparked growing interest in LLM-based autonomous agents [10, 61, 64]. However, limited attention has been given to agent-based frameworks for knob tuning, despite LLM's potential to emulate DBA workflows in a structured and interpretable manner.

Our Proposal. Motivated by these observations, we propose **AgentTune**, a novel LLM-driven agent-based framework designed to achieve effective and reliable database knob tuning, without extensive training data preparation. AgentTune decomposes the overall tuning task into four key subtasks, each handled by a dedicated agent: a central *Configuration Recommender* for generating knob values, supported by three auxiliary agents: *Workload Analyzer*, *Knob Selector*, and *Range Pruner*. These agents collectively emulate the behavior and decision-making of experienced DBAs.

Specifically, given a workload, the *Workload Analyzer* interprets the workload and relevant DBMS settings to extract key features, guiding the subsequent tuning process. The *Knob Selector* then identifies the most impactful knobs, following a *Range Pruner* defined to discover valid ranges to avoid unsafe configurations (addressing L2). Finally, the *Configuration Recommender* uses a novel tree-based iterative process to generate high-performing configurations. This process starts with the default knob configuration. In each iteration, the LLM refines the configuration based on DBMS feedback and generates multiple new candidate configurations. We then prioritize candidates using an innovative centroid-distance-based ranking mechanism and select the top-k most self-consistent candidates to start a new iteration.

By leveraging DBMS feedback, this iterative process quickly converges to high-quality configurations, akin to how human DBAs iterate, resulting in high efficiency (addressing L1). Additionally, AgentTune requires minimal preparation when adapting to new environments, as it is driven by prompt engineering and rule-based tools. For example, AgentTune can be seamlessly deployed across different machines or DBMS platforms without needing re-training or extensive data collection (addressing L3).

While AgentTune's pipeline shares high-level stages with existing approaches such as OtterTune [2] and GPTuner [26], its core novelty lies in the fully LLM-driven, agent-based design. Specifically, although GPTuner leverages LLMs to select promising knobs, it ultimately relies on traditional Bayesian optimization for the configuration search. In contrast, AgentTune decomposes the human DBA decision-making process into specialized LLM agents, each responsible for a distinct subtask within the pipeline. *This enables a shift from optimization based on fitted data points to a paradigm grounded in semantic-rich context and logical reasoning, as enabled by LLMs.* As demonstrated in Figure 1, this approach allows AgentTune to consistently find better configurations with fewer workload replays on SYSBENCH and JOB benchmarks. By closely emulating DBA workflows, AgentTune delivers substantial improvements in tuning efficiency, reliability, and out-of-the-box usability.

Evaluation. We extensively evaluate AgentTune across diverse workloads (public benchmarks and real-world tasks), performance metrics (latency and throughput), DBMSs (PostgreSQL, MySQL, and TiDB), hardware setups, and database scales. Compared to the state-of-the-art baselines on each workload, AgentTune is able to identify superior configurations using significantly fewer workload replays. Furthermore, we show that AgentTune is substantially more reliable, rarely generating invalid configurations during tuning. Finally, we present comprehensive ablation studies and analyze the LLM-related overhead.

Contributions. Our work makes the following contributions:

- We propose **AgentTune**, the first LLM-driven agent-based framework for database knob tuning, offering high efficiency, reliability, and adaptability. To achieve this goal, we decompose knob tuning into four sub-tasks - workload analysis, knob selection, range pruning, and knob recommendation - each managed by a specialized agent.
- We present a novel tree-based search framework, combined with centroid-distance-based ranking, to accelerate convergence while incorporating DBMS feedback during tuning.
- We perform extensive evaluations across various DBMSs, benchmarks, hardware environments, and database scales, showcasing the effectiveness and robustness of AgentTune.

2 Background and Related Work

Database Knob Tuning. Configuration knobs control many aspects of database systems (*e.g.*, user connections, query optimization, and resource management), and different combinations of knob values can significantly impact system robustness, performance, and resource usage [6]. Given a workload, knob tuning aims to adjust these values to improve database performance [70].

For example, memory management in a database is governed by various knobs, such as those controlling space for user connections, shared buffers, and background processes. Proper tuning of these parameters can reduce disk I/O and improve data access efficiency.

Traditionally, knob tuning relies on database administrators (DBAs) to manually experiment with different configurations. However, this process is costly, time-consuming, and difficult to scale, especially in large cloud environments with thousands of database instances [44]. To address this, researchers have developed automated tuning systems using machine learning (ML) techniques, capable of identifying effective configurations without human intervention [2, 24, 28, 34, 51].

The approaches for database knob tuning fall into four primary categories: reinforcement learning (RL) based approaches [5, 28, 55, 65], Bayesian optimization (BO) based techniques [2, 13, 67, 68], deep learning (DL) based methods [3, 34, 51], and heuristic methods [7, 73]. These tuning methods typically require repeated interactions with the DBMS, where each iteration involves executing the workload to evaluate a new configuration. This iterative process is often slow and resource-intensive, limiting its applicability in time-sensitive or cost-sensitive environments. **Large Language Models**. Language models (LMs) have long been a foundational component of natural language processing (NLP). Large language models (LLMs), a powerful subclass of LMs, leverage massive datasets and computational resources to achieve state-of-the-art performance in a wide range of NLP tasks, including machine translation [1, 9] and summarization [36, 49]. Early neural language models struggled to capture the full richness of natural language. A major breakthrough came with the introduction of the Transformer architecture [59], which employs self-attention to model relationships across entire sequences, enabling much deeper contextual understanding. The Transformer architecture serves as the foundation for many prominent LLMs, including BERT [11], GPT-2 [48], and XLNet [63]. More recent models such as ChatGPT [42] and GPT-4 [40] demonstrate broad generalization capabilities and excel at tasks involving reasoning, synthesis, and interaction.

LLMs have recently shown promising results in database-related tasks, including performance diagnosis [72], text-to-SQL translation [15, 30], and query rewriting [33, 35]. Their extensive domain knowledge, reasoning capabilities, and interpretability make LLMs attractive for tackling challenges in database optimization. Several recent studies have started to integrate LLMs into knob tuning. For instance, DB-Bert [26] and GPTuner [26] combine LLM-driven insights with Bayesian optimization. However, these approaches primarily treat LLMs as auxiliary tools to enhance traditional machine learning workflows, rather than positioning LLMs as the central tuning mechanism. In contrast, E2ETune [23] fine-tunes an LLM for direct knob recommendation, but relies on thousands of historical tuning records, which are often difficult to obtain, limiting its adaptability to new environments. These limitations highlight a promising opportunity to develop LLM-centric tuning frameworks that fully harness the capabilities of modern language models while retaining their zero-shot adaptability for dynamic and diverse scenarios.

3 Overview of AgentTune

Database knob tuning presents an NP-hard optimization problem due to the high dimensionality of configuration knobs and the sensitivity of database performance to workload characteristics. When utilizing LLMs for knob tuning, key challenges include: (i) enabling LLMs to effectively and efficiently solve the inherently complex task, and (ii) providing LLMs with sufficient relevant information and context to generate high-quality solutions. AgentTune addresses these challenges by introducing a novel agent-based framework driven by LLMs. By decomposing the complex tuning task into several simple subtasks and assigning them to specialized agents, AgentTune improves both tuning efficiency and reliability (addressing challenge i). Then, each subtask is guided by carefully designed prompts that specify clear task instructions, structured input, and well-defined output formats (addressing challenge ii). We believe that our agent-based framework

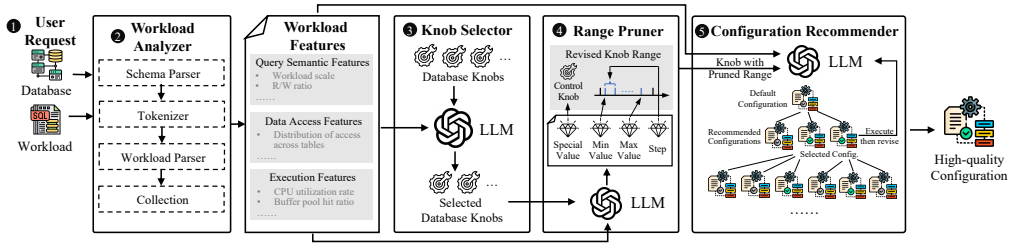

Fig. 2. Workflow of AgentTune.

and prompt designs offer best practices for applying LLMs to knob tuning and can inspire broader applications in hyperparameter optimization.

3.1 Workflow

AgentTune is an innovative LLM-driven agent-based framework designed to recommend high-quality knob configurations tailored to diverse database workloads. Figure 2 presents the tuning workflow, which involves five key steps:

❶ The user provides the DBMS to be tuned (*e.g.*, PostgreSQL or MySQL), the target workload, and the optimization objective (*e.g.*, latency or throughput). ❷ The *Workload Analyzer* examines the given workload and extracts its key characteristics, including both static and dynamic features. ❸ The *Knob Selector* reduces the search space dimensionality by selecting important knobs to tune, based on the workload features identified in the previous step. ❹ The *Range Pruner* further refines the knob search space by narrowing the value range of each selected knob to better align with the current tuning task. ❺ The *Configuration Recommender* leverages the workload features, selected knobs, and their refined ranges to generate a high-quality configuration that optimizes database performance.

3.2 Components

AgentTune consists of four main components—*Workload Analyzer*, *Knob Selector*, *Range Pruner*, and *Knob Recommender* - which collaboratively perform the tuning task as follows:

Workload Analyzer (Rule-based Agent). The Workload Analyzer summarizes the input workload to make it suitable for LLM consumption, ensuring that sufficient information is retained for tuning while reducing LLM-related processing costs (Section 4). The extracted features fall into two categories: static features (e.g., query semantics and data access characteristics) and dynamic features (e.g., internal metrics and query execution statistics). This mirrors the approach taken by human DBAs, who typically focus on high-level workload features.

Knob Selector (LLM-based Agent). The Knob Selector identifies the most influential knobs for the current workload (Section 5). Modern DBMSs include hundreds of knobs, but the most impactful ones can vary significantly depending on the workload. For example, the knob `innodb_buffer_pool_size` is critical for OLTP workloads but less relevant for OLAP scenarios. The Knob Selector uses the LLM to select workload-related important knobs, which enhances optimization efficiency while preserving tuning effectiveness.

Range Pruner (LLM-based Agent). The Range Pruner optimizes the tuning space by refining knob value ranges, iteration granularity, and special-case values (see Section 6). It processes workload features and identified knobs using an LLM-based agent to determine the key value range for each selected knob. Importantly, LLM-generated suggestions are combined with rule-based constraints to enhance both efficiency and reliability.

Query Semantic Features	Data Access Features	Execution Features
• Workload scale • Read/Write ratio • Average tables per query • Average logical predicates per constraint clause • Average aggregation functions per query • Average GROUP BY operations per query • Average projected attributes per result set	• Distribution of access across tables • Distribution of access across columns • Distribution of value ranges for queried columns • Proportion of comparison constraint types in WHERE clauses • Ratio of ascending to descending order operations in queries involve sorting.	***Inner Metrics*** • CPU utilization rate • Buffer pool hit ratio • Disk I/O throughput ***Query Execution Statistics*** • Latency distribution • Transaction throughput • Result set cardinality • Lock contention duration

Fig. 3. Workload features generated by the Workload Analyzer, including static features (query semantics and data access) and dynamic features (execution metrics).

Knob Recommender (LLM-based Agent). We introduce a novel tree-based search framework to identify suitable configurations for the given workload (see Section 7.2). Unlike traditional sequential tuning methods, our approach generates multiple candidate configurations in each iteration, selecting the top-k for further refinement, forming a tree-like structure. To improve efficiency, a ranking mechanism is employed during the iteration process to avoid evaluating every intermediate candidate configuration.

4 Workload Analyzer

Workload plays a pivotal role in database tuning tasks, directly influencing the discovery of suitable configurations. Most ML-based methods utilize workload encoding techniques [2, 28] to uncover latent correlations across tuning scenarios. Similarly, workload-related information should be included in the prompts to help LLMs understand the tuning context and provide more targeted guidance. Yet, the current workload analysis has notable drawbacks:
(1) Over-abstraction: Some techniques use neural network-based representations or binary encoding schemes, resulting in high-dimensional features that lack semantic interpretability for LLMs [71].
(2) Underspecification: Other methods simplify workloads into primitive descriptors (e.g., workload type), offering insufficient contextual signals for LLM-driven decision-making [14].
(3) Verbosity: Some approaches include raw SQL queries in prompts, leading to overly long inputs that increase LLM processing costs and risk exceeding token limits [17].

These methods often overlook the structured, analytical strategies used by database administrators (DBAs), who typically observe system status, analyze workloads, and extract critical features from both data distributions and query patterns. To address this gap, we propose a feature-based workload analysis framework that captures three categories of features: (1) query semantic features, (2) data access features, and (3) execution features. As shown in Figure 3, the first two categories—query semantic and data access features—are static, derived directly from workload analysis, while execution features are dynamic, collected from the database during runtime.

Query semantic features describe the characteristics of SQL statements in the workload, including metrics such as total query count, read/write ratio, and the average number of tables accessed per query.

Data access features capture how the workload interacts with the database schema and access patterns, including frequently queried tables, columns, and values. These features reflect schema structure, data distribution ranges used in queries, and expected access frequencies.

Execution features capture the DBMS's current state under a specific configuration. These features include internal database metrics (e.g., CPU usage, cache hit rate, I/O volume) and query-level metrics (e.g., latency, throughput, result size). They reflect the system's actual behavior under

the workload and encode contextual details such as hardware configuration, knob settings, storage layout, and query execution patterns. For instance, MySQL's InnoDB engine provides metrics like pages read/written, query cache usage, and lock contention, which implicitly reveal workload dynamics and system state. These insights are crucial for refining and optimizing the current configuration. Unlike static features, execution features are dynamic and capture the outcomes of workload-system interactions. Performance metrics such as latency and throughput are influenced by numerous interdependent factors, making static features alone insufficient for comprehensive tuning.

To extract these features, our framework starts with a *schema parser* that processes the database schema from configuration files. Then, a *tokenizer module* uses regular expressions to tokenize SQL statements. The tokenized output is passed to a *workload parser*, which builds syntax trees to identify key components such as table and column names. Following this, a *feature collection module* computes and aggregates semantic and data-related features. Finally, the extracted information is cross-validated against the parsed schema to ensure consistency and completeness. To minimize the impact of internal knowledge memorized by LLMs, we anonymize workload features by omitting specific workload names (e.g., TPC-C, JOB)[1].

5 Knob Selector

Modern database systems offer hundreds of adjustable knobs; however, not all knobs are equally important for every workload. For example, working memory size is crucial for memory-intensive workloads, while maximum I/O concurrency is more relevant for I/O-intensive workloads. Therefore, considering the characteristics of both the database and the workload, the goal of the Knob Selector is to identify the most impactful knobs for the given workload. By narrowing the focus to these selected knobs, knob tuners can streamline the tuning process and enhance the quality of the recommended configuration.

Traditional approaches rely on ML-based algorithms to identify important knobs, typically requiring hundreds to thousands of training data samples across different workloads and configurations [37, 52]. Recently, emerging methods have leveraged LLMs to identify tunable knobs by utilizing external knob tuning knowledge, demonstrating improved performance over traditional ML techniques [26, 55]. However, these methods still impose a significant burden on users, requiring them to manually gather relevant knob tuning knowledge from sources like official documentation, user manuals, and community posts. This reliance on manual effort limits their ability to adapt effectively to new environments.

To eliminate this overhead, we propose a *workload-aware and training-free* approach. Specifically, we leverage LLMs as experienced DBAs, and incorporate several key elements into the prompt for selecting important knobs. As shown in Figure 4, the prompt for knob pruning includes the following components:

- **Task Description** specifies the LLM's objective, which is selecting the most critical knobs in the given context.
- **Candidate Knobs** provides detailed information on the candidate knobs available in the database engine, including their default ranges, types, and descriptions. These details are easily accessible from official documentation. To eliminate potential bias from LLM's pre-training, we anonymize all knob names used in our framework.
- **Workload and Database Information** includes essential features extracted by the *Workload Analyzer* module, as well as database kernel and hardware specifications.

[1]This strategy is effective in practice. For instance, when instructed to recommend knob values, the LLM does not attempt to infer or guess the underlying workload names; instead, it relies solely on the provided information and performs logical analysis. Illustrative LLM-generated sequences are available at: https://github.com/intlyy/AgentTune/tree/main/examples.

Knob Selector

Task Description:
Select the 20 most important knobs from the provided for the current tuning task in order to optimize the throughput metric.
Candidate Knobs:
{ “knob1”: { "max": 1000, "min": 0, "type": "integer", "description": "Defines the maximum number of threads permitted. }, ……}
Workload and Database Information:
- Workload Features: Query Semantic Features; Data Access Features; Execution Features
- Database Kernel: RDS MySQL 5.7.
- Hardware: 8 vCPUs and 16 GB RAM, Disk Type = HDD
Output Format:
Knobs should be formatted as follows:
{“knob_1”, …… }

Fig. 4. Prompt for the knob selector.

Range Pruner

Task Description:
Given the knobs along with their suggestion and tuning task information, your job is to offer intervals for each knob that may lead to the best performance of the system and meet the hardware resource constraints. In addition, if there is a special value (e.g., 0, -1, etc.), please mark it with “special value”.
Knob:
{“knob1": { "max": 1000, "min": 0, "type": "integer", “detailed description": "Defines the maximum number of threads."}, …… }
Workload and Database Information:
- Workload Features: Query Semantic Features; Data Access Features; Execution Features
- Database Kernel: RDS MySQL 5.7.
- Hardware: 8 vCPUs and 16 GB RAM, Disk Type = HDD
Output Format:
Knobs should be formatted as follows:
{ “knob_1”:{ "min_value": MIN_VALUE, "max_value": MAX_VALUE, “step": STEP, "special_value": SPECIAL_VALUE}, …… }

Fig. 5. Prompt for the range pruner.

- **Output Format** defines the required structure of the LLM’s response. Specifically, the model is instructed to list the serial numbers of the selected knobs in an organized JSON format.

These prompts are automatically generated and constructed without any human intervention. One of the key advantages of our approach is that it is *fully automated and works out of the box*, requiring minimal additional preparation effort.

6 Range Pruner

The goal of the *Range Pruner* module is to optimize the value range for each knob identified by the *Knob Selector* module. This step is crucial for improving both tuning efficiency and safety. To achieve effective pruning, the following challenges must be addressed:

(1) Overly Generic Default Ranges: Default ranges are typically designed to be as broad as possible. For instance, one of the most critical knobs in MySQL, `innodb_buffer_pool_size`, has a default range of $[0, 2^{64} - 1]$ bytes on 64-bit platforms. This knob controls the size (in bytes) of the buffer pool where InnoDB caches table and index data. Ranges beyond available memory are however invalid and may cause the DBMS to fail at startup.

(2) Ineffectiveness of Small Adjustments: For knobs with large ranges, small changes often have negligible impact on performance. Moreover, discrete knobs exposed by the DBMS can vary dramatically in scale. Some have only 10 possible values, while others span millions. For example, tuning `innodb_log_buffer_size` shows no meaningful performance difference between 67,108,864 bytes and 67,109,888 bytes, only a 1KB difference.

(3) Presence of Special Values: Some knobs support special values (e.g., -1 or 0) that trigger unique database behaviors. These are difficult to handle with traditional ML-based methods, which typically rely on smooth input-output mappings and often overlook such special semantics, as this knowledge is usually only available through manuals or expert experience.

To address these challenges, we introduce a *Range Pruner* module. This module leverages an LLM to refine the value ranges of selected important knobs, which are drawn from the *Knob Selector* module. The range pruner begins by presenting the LLM with all selected knobs and their corresponding default ranges as a unified input context. This design enables the LLM to reason holistically about potential interdependencies among knobs when suggesting refined value ranges. Based on the LLM's output, the search space is reconstructed with updated, narrower ranges. For each knob, the pruned range is defined using four elements: upper bound, lower bound, special value, and iteration granularity. The upper and lower bounds define the allowable value range, while the special value explicitly indicates any exceptional setting and its semantics. The iteration granularity specifies the recommended adjustment size for each knob tuning step, preventing the ineffectiveness of small adjustments in subsequent knob recommendations. We illustrate the specific prompt used for the range pruner module in Figure 5.

To ensure safety and correctness, we apply a rule-based white-box validation after LLM suggestions. This step filters out LLM-hallucinated knob names and invalid values, based on constraints derived from DBMS documentation and hardware specs. Additionally, it serves as a human-in-the-loop interface, allowing experts to incorporate domain knowledge or impose deployment-specific limits. These rules are configured once per machine or DBMS with minimal manual effort, enhancing both robustness and adaptability.

Overall, this pruning strategy not only enhances practical applicability but also aligns with how human DBAs operate: first identifying relevant knobs, then narrowing their ranges for tuning.

7 Configuration Recommender

In this section, we introduce our tree-based knob recommender module, powered by LLMs. Unlike traditional sequence-based iterative methods (e.g., RL- and BO-based approaches), we reformulate the tuning process as a tree search, leveraging LLMs to efficiently identify high-quality configurations based on feedback from the DBMS. This tree-based tuning paradigm enhances the capabilities of LLMs by fully utilizing their ability to generate multiple candidate outputs in parallel, further improving efficiency. We begin by presenting our tree-based tuning framework in Section 7.1, followed by a detailed discussion of the iterative LLM-based tuning method in Section 7.2.

7.1 Tree-Based Tuning Framework

Iteration is a critical component of most existing knob tuning systems. However, prior research has primarily focused on improving the tuning algorithms themselves (e.g., enhancing surrogate models for Bayesian optimization or designing novel neural networks for reinforcement learning), while the iterative process has received comparatively little attention. Traditional tuning methods typically follow a try-collect-adjust cycle, where samples are gathered to build Gaussian processes or train neural networks. This process often involves hundreds of iterations, many of which yield invalid configurations. As a result, the majority of the overall tuning time is consumed by repeatedly executing the workload [66].

We propose our tree-based tuning framework to address the issues above. We transform the tuning process into a beam search [43]. Specifically, we treat each knob configuration in the tuning process as a node in the tree and expand it with multiple child nodes (*i.e.*, candidate configurations for the next round) through the tuning method. Details of the tuning flow are shown in Figure 6. Firstly, we initialize the beam and best-found configuration with the default configuration. Then the tuning

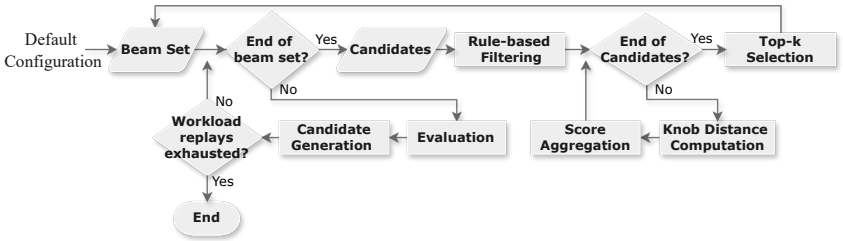

Fig. 6. AgentTune's tree-based tuning workflow.

process starts. In each layer, the tuning method recommends R candidate configurations for each node in the current beam, which are added to the candidate set. Similar to the rule-based white-box validation in *Range Pruner* module, we check the candidate configurations with pre-defined rules to remove duplicate, illegal, and unsafe candidates. Next, we employ a centroid-based ranking method to select the top-k configurations (where k is the beam width) from all candidate configurations for the next iteration. Each configuration is represented as a vector, where each element corresponds to the value of a particular knob. The selection process is as follows: (1) For each knob, compute the average value across all candidate configurations. (2) For each knob, rank all configurations according to how close their knob value is to the computed mean. (3) For each configuration $\boldsymbol{\theta}_i$, sum its ranks across all knobs to obtain an overall score D_i. (4) Sort all configurations by their overall scores in ascending order and select the top-k configurations as the beam for the next iteration[2]. This centroid-based ranking mechanism assumes that configurations closer to the center of the candidate set represent the dominant voting results, ensuring self-consistency among the selected candidates. This approach guides subsequent iterations toward optimizing configurations in a promising direction. Note that each layer involves one or more workload replays. In this paper, the maximum number of workload replays is treated as a hyperparameter. Once this limit is reached, the algorithm returns the best-found configuration from the search tree. In practice, early stopping strategies—such as terminating if no improvement is observed over the last 10 workload replays—can also be applied to further reduce resource and time consumption.

Such a tree-based search process offers several key advantages: **(1) Efficiency Improvement.** Compared to traditional sequence-based models, our tree-based approach explores a greater number of potential configurations in each round while preemptively filtering out invalid regions. In other words, it enhances both the sampling quantity and quality, thereby accelerating the overall tuning process. **(2) Universality.** This tree-based framework does not rely on a specific tuning algorithm and is compatible with a wide range of existing methods, including Bayesian optimization, reinforcement learning, and large language models. We will detail our iterative LLM-based tuning method in Section 7.2. Moreover, thanks to its training-free and plug-and-play design, our framework can easily adapt to various tuning tasks and environments.

7.2 Knob Recommendation with LLM

The knob recommendation component is a pivotal part of the tuning system, tasked with suggesting high-quality configurations for specific workloads to enhance performance metrics. Recent studies have explored leveraging LLMs to recommend promising configurations [17, 23]. However, these approaches predominantly adopt a one-shot strategy, relying solely on LLMs without incorporating feedback from the DBMS to iteratively refine the configurations, which can result in suboptimal performance.

[2]By default, we set the beam width $k = 2$ to achieve the best trade-off between tuning efficiency and performance. We also tested other values, with results detailed in Section 8.4.4.

Among the algorithms discussed, LLMs integrate seamlessly with our tree-based tuning framework. By leveraging their inherent ability to generate diverse outputs through adjustable generation parameters (such as temperature), LLMs are particularly well-suited for iterative, tree-based processes. Furthermore, their advanced understanding of database-related knowledge (e.g., the description of knobs and the execution feedback from DBMS) and strong exploration-exploitation capabilities make them highly effective for knob-tuning tasks [32]. Building on these strengths, we propose an iterative LLM-driven knob-tuning approach that incrementally refines configurations on a beam search tree. Specifically, for each node (representing the current configuration) in the beam, LLMs generate R child nodes (i.e., R refined configurations) using nucleus sampling [20], guided by feedback from the DBMS[3].

Formally, the LLM-based tuning method (i.e., the configuration refinement process) is defined as:

$$LLM(P_r, \boldsymbol{\theta}_{t,i}, \mathcal{D}, \mathcal{W}, \mathcal{S}, \boldsymbol{\Theta}_p, F_{t,i}, M) \rightarrow \{\boldsymbol{\theta}^1_{t+1,i}, \boldsymbol{\theta}^2_{t+1,i}, ..., \boldsymbol{\theta}^R_{t+1,i}\} \tag{1}$$

Here, t represents t-th iteration (i.e., the current iteration), P_r denotes the knob recommendation prompt, and $\boldsymbol{\theta}_{t,i}$ signifies i-th configuration in the current beam. $\mathcal{D}$ and $\mathcal{W}$ represent the database and workload features, respectively. $F_{t,i}$ refers to the feedback from the DBMS under the configuration $\boldsymbol{\theta}_{t,i}$. $\mathcal{S}$ and $\boldsymbol{\Theta}_p$ represents the selected important knobs and their ranges. M represents the memory window we introduce, which stores a number of top-performing configurations from the beginning of the tuning process to the current t-th step. These records are continuously updated to prevent the prompt length from growing excessively. The output $\{\boldsymbol{\theta}^1_{t+1,i}, \boldsymbol{\theta}^2_{t+1,i}, ..., \boldsymbol{\theta}^R_{t+1,i}\}$ represents the LLM refined multiple configurations based on $\boldsymbol{\theta}_{t,i}$.

After expanding all nodes in the current beam using the LLM, the candidate configuration set is formed as $\{\boldsymbol{\theta}^r_{t+1,i} \mid r = 1, 2, \ldots, R; i = 1, 2, \ldots, k\}$, where k represents the beam width. Next, rule-based checking and the centroid-based ranking mechanism are applied to form the new beam, consisting of k refined nodes $\{\boldsymbol{\theta}_{t+1,i} \mid i = 1, 2, \ldots, k\}$. This process initiates a new iteration to further optimize $\boldsymbol{\theta}_{t+1,i}$. Initially, the beam contains only a single configuration, corresponding to the default settings. The refinement process proceeds iteratively until a predefined stopping criterion is satisfied.

As illustrated in Figure 7, we construct the prompt for knob recommendation in the following format:

- **Task Description** outlines the objective of the LLM.
- **Knobs** consists of the selected knobs from the *Knob Selector* module and their corresponding ranges from the *Range Pruner* module, along with detailed knob descriptions for LLM reference.
- **Workload and Database Information** contains essential details about the workload features obtained from the *Workload Analyzer* module, as well as information about the database kernel and hardware specifications.
- **Memory Window** includes a few knob refinement instances, which serve as few-shot examples within the prompt for demonstration purposes. Each instance includes the current configuration, internal metrics, and a refined configuration.
- **Current Configuration** displays the current knob values.
- **Database Feedback** showcases the performance and execution features of the database when executing the given workload under the current configuration. Incorporating this feedback is essential, as DBAs often rely on these metrics to assess the database's status and

[3]In practice, we generate three refined configurations for each node in the beam ($R = 3$). While experimenting with larger values, we observed no significant performance improvements, making $R = 3$ an optimal choice for balancing LLM inference costs and knob-tuning performance.

Configuration Recommender

Task Description:
Recommend optimal knob configuration based on the given information in order to optimize the throughput metric.
Knobs:
{ “knob1”: { "min_value": MIN_VALUE, "max_value": MAX_VALUE, “step": STEP, "special_value": SPECIAL_VALUE, "description": "Defines the maximum number of threads."}, …… }
Workload and Database Information:
- Workload Features: Query Semantic Features; Data Access Features
- Database Kernel: RDS MySQL 5.7.
- Hardware: 8 vCPUs and 16 GB RAM, Disk Type = HDD
Memory Window:
Set of optimal historical records
Current Configuration:
{ “knob1”: 10, …… }
Database Feedback:
- Throughput : 100
- Execution Features : Execution Features
Output Format:
Strictly utilize the aforementioned knobs, ensuring that the generated configuration are formatted as follows:
{ “knob_serial number”: knob_value, …… }

Fig. 7. Prompt for the configuration recommender.

implement necessary adjustments. For example, when confronted with a low cache hit rate, DBAs typically choose to increase the cache size to improve database performance.

- **Output Format** specifies the format for the LLM responses.

8 Experimental Evaluation

This section presents the experimental results comparing AgentTune against the state-of-the-art tuning approaches. The AgentTune implementation is available as open-source[4].

8.1 Experimental Setup

Workloads. To evaluate AgentTune's performance, we utilize four widely adopted public benchmarks: two OLAP benchmarks (TPC-DS [54] and JOB [27]) and two OLTP benchmarks (TPC-C [53] and SYSBENCH [25]). Specifically, for OLAP, we use the TPC-DS benchmark with a scaling factor of 1, resulting in 1 GB of data and 99 complex queries. The JOB benchmark includes 113 complex queries on 9 GB of data. For OLTP, we utilize the TPC-C benchmark with 10 warehouses and 32 connections. Additionally, we employ the SYSBENCH benchmark with the OLTP-Read-Write workload, loading 50 tables with 1,000,000 rows each, resulting in a total dataset size of approximately 10 GB.

To assess the practical applicability of AgentTune, we include two real-world benchmarks from a company: SSAG (58 GB) and AMPS (25 GB). SSAG is an OLAP benchmark designed for slow SQL analysis and governance, focusing on complex analytical queries for tasks such as slow SQL template analysis and logical database analysis. AMPS, on the other hand, is an OLTP benchmark tailored for AI platform services, encompassing tasks like user management, permission control, model management, and task scheduling.

Hardware and Software. Unless otherwise specified, all experiments (excluding real-world benchmarks) are conducted on a machine equipped with an Intel i7-7700 processor (8 cores, 3.60GHz) and 16 GB RAM, running RDS MySQL 5.7 by default. Evaluations of real-world benchmarks are performed in a separate production environment.

[4]https://github.com/intlyy/AgentTune

Baselines. We compare AgentTune against several state-of-the-art methods: (1) DDPG++ [58], an RL-based method from CDBTune [65]. (2) VBO, a BO-based method used in iTuned [13] and OtterTune [2]. (3) SMAC [66], a BO-based method using a random forest as the surrogate model. (4) GPTuner [26], an LLM-assisted BO-based tuner that leverages LLMs to interpret manuals and reduce the search space. (5) DB-BERT [55], an influential NLP-based tuning approach. Additionally, we include widely-used rule-based tuners: MySQLTuner [19] for MySQL and PgTune [60] for PostgreSQL. We also use the default configuration as a reference baseline.

Tuning Settings. In all experiments, the LLM is configured with a temperature of 1.0 and a top-p of 0.98 to balance output diversity and determinism. Following the settings used in prior work [66], we set the number of important knobs to 20. Since traditional techniques such as DDPG+, SMAC, and VBO do not include a knob selection module, we apply SHAP [37]—a knob selection method validated in prior work—to preselect 20 important knobs for these baselines. While traditional baselines typically perform one workload replay (i.e., stress test) per iteration, our tree-based search framework may require multiple workload replays within a single iteration. Since workload replay is the most time-consuming operation in the tuning process, we define a single execution of a workload replay as a "step". To ensure fair comparisons, all methods are limited to a maximum of 100 steps in our evaluation. To ensure knob settings take effect, the database is restarted after applying a new configuration. We use GPT-4 as the LLM for both our method and GPTuner. For DB-BERT, we adopt the same training settings as described in the extended version of the paper published in the VLDB Journal [56]. For BO-based methods, we follow the setting of iTuned [13] and OtterTune [2] by executing 10 configurations generated by Latin Hypercube Sampling (LHS) [38] to initialize the surrogate model. For RL-based methods, consistent with recent studies [5, 55], we do not pre-train the neural network due to overfitting concerns [66]. For rule-based tuning methods, PGTune and MySQLTuner, we strictly follow their official usage instructions and deployment procedures. We provide the required system specifications (e.g., memory size, CPU count) as inputs to ensure accurate recommendations.

Evaluation Metric. We evaluate our method and baselines across three key dimensions: best-found database performance, tuning efficiency, and tuning safety. The best-found database performance metric equals the maximum achievable *throughput* (OLTP benchmark) or minimum *latency* (OLAP benchmark) during the tuning procedure. Then, because the main portion of the tuning time is spent on workload replays, we propose T_{opt}, the number of workload replays needed to reach peak performance, to quantify tuning speed. Building upon this, we introduce a new metric, *Performance Improvement Efficiency* (PIE), to measure the tuning efficiency, which can be calculated as follows:

$$\text{PIE} = \frac{1}{T_{\text{opt}}} \times \frac{P_{\text{gap}}}{P_{\text{default}}} \tag{2}$$

where

$$P_{\text{gap}} = \begin{cases} P_{\text{default}} - P_{\text{opt}}, & \text{if the benchmark is OLAP} \\ P_{\text{opt}} - P_{\text{default}}, & \text{if the benchmark is OLTP} \end{cases}$$

Here, $P_{default}$ and P_{opt} denote the default and optimized database performance (throughput or latency). To assess safety in the tuning process, we define the *Invalid Times* metric, which counts the number of configurations that perform worse than the default setting or are considered illegal. To avoid counting minor fluctuations as errors, we use a threshold of 10% degradation relative to the default configuration to determine invalid configurations. Throughput and PIE are "higher is better" metrics, whereas latency, T_{opt}, and Invalid Times are "lower is better" metrics.

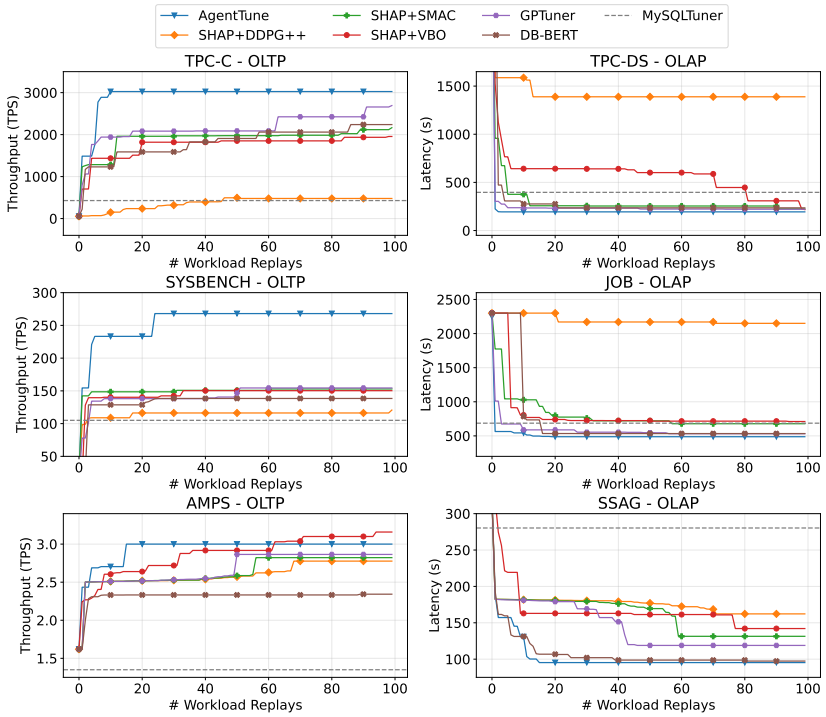

Fig. 8. Performance comparison of best-found configurations throughout the tuning process across six different benchmarks.

8.2 Main Results

The experimental results prove that our method outperforms existing state-of-the-art in *Performance*, *Efficiency*, and *Safety* aspects.

AgentTune discovers superior knob configurations. As shown in Table 1 and Figure 8, AgentTune consistently finds the best configurations across nearly all evaluated benchmarks (except for AMPS). For instance, in TPC-C (OLTP), it achieves 56× higher throughput over the default, and 16× in SYSBENCH ($\frac{3025.70}{54.30} = 56$, $\frac{268.03}{17.25} = 16$). Compared to GPTuner, AgentTune outperforms it by 11.0% in TPC-C and 42.5% in SYSBENCH ($\frac{3025.70-2692.22}{3025.70} = 11.0\%$, $\frac{268.03-154.43}{268.03} = 42.5\%$). AgentTune achieves the best performance improvements on TPC-C and SYSBENCH and the second-best on AMPS. Similar trends hold for OLAP benchmarks: AgentTune reduces latency by 94.2% (TPC-DS), 78.8% (JOB), and 70.2% (SSAG) ($\frac{3338.38-193.88}{3338.38} = 94.2\%$, $\frac{2300.25-487.88}{2300.25} = 78.8\%$, $\frac{319.89-95.27}{319.89} = 70.2\%$), outperforming all baseline methods. These gains are primarily attributed to AgentTune's agent-based design. The analyzer extracts key workload-specific features, which the selector and pruner use to identify important knobs and ranges. Finally, the tree-based configuration recommender leverages DBMS feedback to perform a structured search. Together, these efforts effectively enhance the upper bound of tuning performance.

AgentTune offers the highest tuning efficiency. It can be observed from both Table 1 and Figure 8 that **AgentTune** converges rapidly, achieving the minimal T_{opt} across all benchmarks. This indicates that it requires the fewest workload replays to reach peak performance. In contrast, traditional methods often need significantly more samples to train their models, whether based on Bayesian optimization or Reinforcement learning.

Table 1. Experimental results on different benchmarks. We measure database performance using throughput for OLTP benchmarks (higher is better) and latency for the OLAP benchmarks (lower is better). IT denotes "Invalid Times".

Benchmark	Method	Performance	T_{opt}	PIE (%)	IT
TPC-C	Default	54.30	-	-	-
	MySQLTuner	428.73	-	-	**0**
	SHAP + DDPG++	478.79	51	15.60	41
	SHAP + SMAC	2164.72	100	38.85	15
	SHAP + VBO	1955.56	98	35.72	21
	GPTuner	2692.22	100	48.57	7
	DB-BERT	2239.03	86	46.78	29
	AgentTune	**3025.71**	**11**	**497.35**	**0**
SYSBENCH	Default	17.25	-	-	-
	MySQLTuner	105.47	-	-	**0**
	SHAP + DDPG++	120.71	99	6.06	91
	SHAP + SMAC	152.91	60	13.11	7
	SHAP + VBO	150.31	33	23.38	27
	GPTuner	154.43	51	15.59	12
	DB-BERT	138.40	26	27.01	4
	AgentTune	**268.03**	**23**	**63.10**	**0**
AMPS Real-world OLTP	Default	1.62	-	-	-
	MySQLTuner	1.35	-	-	1
	SHAP + DDPG++	2.78	70	1.07	10
	SHAP + SMAC	2.82	58	1.30	6
	SHAP + VBO	**3.16**	94	1.01	7
	GPTuner	2.86	52	1.56	3
	DB-BERT	2.34	88	0.51	2
	AgentTune	3.00	**15**	**5.68**	**0**
TPC-DS	Default	3338.38	-	-	-
	MySQLTuner	396.51	-	-	**0**
	SHAP + DDPG++	1388.89	14	4.17	88
	SHAP + SMAC	224.49	92	1.01	12
	SHAP + VBO	215.70	100	0.93	33
	GPTuner	221.43	76	1.18	6
	DB-BERT	235.63	22	4.22	3
	AgentTune	**193.88**	**3**	**31.41**	**0**
JOB	Default	2300.25	-	-	-
	MySQLTuner	686.37	-	-	**0**
	SHAP + DDPG++	2150.59	72	0.09	90
	SHAP + SMAC	678.24	59	1.5	2
	SHAP + VBO	710.75	94	0.82	5
	GPTuner	531.81	56	1.37	13
	DB-BERT	534.01	28	2.68	6
	AgentTune	**487.88**	**19**	**4.15**	**0**
SSAG Real-world OLAP	Default	319.89	-	-	-
	MySQLTuner	280.29	-	-	**0**
	SHAP + DDPG++	162.23	73	0.70	3
	SHAP + SMAC	131.39	61	1.02	**0**
	SHAP + VBO	142.08	79	0.79	2
	GPTuner	118.89	48	1.40	1
	DB-BERT	98.73	39	1.78	**0**
	AgentTune	**95.27**	**17**	**5.02**	**0**

Furthermore, AgentTune achieves the best PIE score across all benchmarks (Table 1), being 2× to 46× more efficient than traditional baselines ($\frac{4.15}{2.68} = 2$, $\frac{4.15}{0.09} = 46$). This stems from two factors: (1) the knob selector and the range pruner drastically narrows the configuration space, and (2) AgentTune incorporates feedback from the DBMS during the iterative process, which speeds up convergence. BO-based and RL-based methods need many iterations to perform the "try-collect-adjust" loops. Although GPTuner uses LLMs in its tuning process, it still relies on BO, resulting in

Table 2. Experimental results on different database scales.

Database Scale (GB)	Throughput (Default)	Throughput (Optimized)	T_{opt}	PIE (%)	Invalid Times
1	85.34	1046.06	19	59.25	0
10	17.25	268.03	23	63.10	0
50	3.26	12.79	8	36.54	0

sub-optimal tuning efficiency. Among all baselines, DB-BERT is the most competitive, achieving the second-highest PIE scores on SYSBENCH, TPC-DS, JOB, and SSAG. Both AgentTune and DB-BERT aim to incorporate DBA expertise into the tuning process; however, DB-BERT still partially relies on reinforcement learning to guide decisions, whereas AgentTune leverages pre-trained LLMs for decision-making.

AgentTune ensures safe tuning. Safety is critical in tuning, as invalid configurations can crash the system or degrade performance significantly. As shown in Table 1, AgentTune introduces no failures or unsafe recommendations, while other tuning methods often produce unsafe configurations during the tuning process. This robustness stems from AgentTune's LLM-guided range pruning and rule-based safety checks during tree-based search. Another method with very few failures is MySQLTuner, a widely used rule-based tool. However, its recommendations are limited to a fixed set of parameters and do not adapt to dynamic workloads. Additionally, many suggestions are non-specific (e.g., "`table_open_cache` (> 2000)"), requiring users to interpret and fine-tune them.

AgentTune generalizes well to real-world workloads. When deployed in a production environment, our method consistently delivered strong performance. Under the AP workload (SSAG), AgentTune successfully reduces the latency from 319.89 s (default configuration) to 95.27 s, achieving a 3.4× improvement, while also outperforming all baselines in tuning efficiency ($\frac{319.89}{95.27} = 3.4$). For AMPS, although VBO slightly outperforms AgentTune in final throughput (3.16 vs. 3.00), AgentTune achieves a 5.6× speedup in tuning efficiency ($\frac{5.68}{1.01} = 5.6$). These results demonstrate that AgentTune can effectively generalize to real-world tuning tasks, even in real-world production environments that LLMs have never encountered before, highlighting its strong transferability and practical utility. More importantly, AgentTune does not recommend any invalid configurations throughout the entire tuning process — a critical property for safe deployment in real-world DBMSs.

8.3 Scalability Study

In this section, we delve into the generalizability of AgentTune by expanding our evaluation framework to encompass new database scales, engines, and hardware environments. These scalability studies are conducted using the SYSBENCH benchmark.

8.3.1 Database Scaling. To evaluate the scalability of AgentTune across different database scales, we conduct experiments on three different data sizes in SYSBENCH: 1GB, 10GB, and 50GB. The results are shown in Table 2. As the database size increases, the absolute performance of the database decreases, both for the default and optimized configurations. Nevertheless, AgentTune demonstrates excellent performance across all scales, with an obvious performance improvement compared to the default configuration. Furthermore, it consistently maintains high efficiency and exceptional safety, proving the robustness of our approach in handling workloads of varying sizes.

8.3.2 Hardware Scaling. As detailed in Section 8.1, our primary evaluations are conducted using the MySQL engine on a server with an 8-core CPU and 16 GB of RAM. To investigate the impact of hardware on tuning performance, we deploy the MySQL database on a separate machine equipped with 40 cores (Intel Xeon Gold 5118 @ 2.30GHz) and 256 GB of RAM. The experimental outcomes are presented in Table 3 and Figure 9 (a).

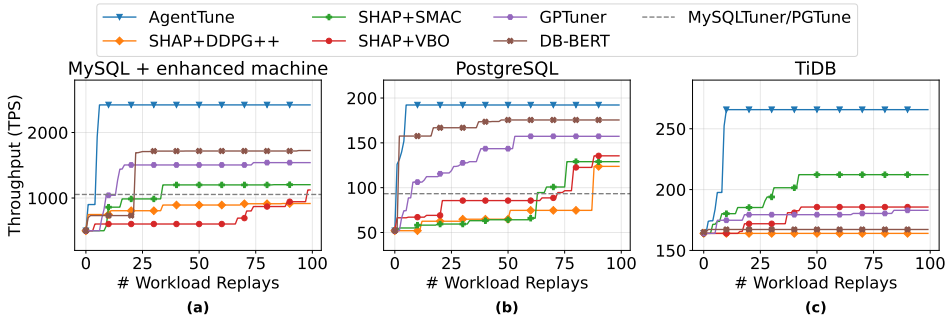

Fig. 9. Performance comparison of best-found configurations during the tuning process on SYSBENCH, measured across (a) MySQL on an enhanced machine, (b) PostgreSQL, and (c) TiDB.

Table 3. Experimental results for "SYSBENCH + MySQL + enhanced machine (40-core CPU with 256GB RAM)".

Hardware	Method	Throughput	T_{opt}	PIE (%)	Invalid Times
40 CPUs 256GB RAM	Default	500.42	-	-	-
	MySQLTuner	1054.32	-	-	0
	SHAP + DDPG++	912.39	86	1.19	89
	SHAP + SMAC	1203.45	83	1.70	10
	SHAP + VBO	1166.23	98	1.27	21
	GPTuner	1540.23	74	2.82	11
	DB-BERT	1724.98	94	2.60	8
	AgentTune	**2421.34**	**6**	**64.05**	**0**

The enhanced CPU capabilities and increased memory require a broader range for certain knobs, thereby complicating the task of identifying the best-performing configuration. Experimental results show that AgentTune significantly outperforms existing knob tuning methods on this upgraded hardware. Notably, the strongest baseline, DB-BERT, achieves a throughput of 1724.98, while AgentTune reaches 2421.34 with considerably better efficiency (2.60 vs. 64.05 PIE) and no unsafe configurations. These findings highlight the adaptability of our method to hardware changes, primarily due to the workload analyzer and range pruner, which dynamically adjust the search space based on the tuning task.

8.3.3 Database Engine Generalization. To assess the versatility of LLMs across different database engines, we additionally evaluate AgentTune on two open-source systems: PostgreSQL and TiDB [21]. As a newer database with limited public documentation, TiDB serves as an effective test case for evaluating AgentTune's adaptability to emerging systems. In our experiments, we use TiDB version 8.2 and deploy a full cluster simulating a production environment. The setup includes 3 TiKV instances (row-based storage engine), 1 TiFlash instance (column-based storage engine), 1 TiDB instance (distributed database server), 1 PD instance (placement driver), and 1 monitor. To apply DB-BERT to TiDB, we follow the approach in the extended DB-BERT paper [56]. Specifically, we use the query "TiDB performance tuning hints" on Google to collect the top 100 relevant documents (totaling 3.85 MB of text) and convert them into the required DB-BERT input format.

After replacing MySQL with PostgreSQL and TiDB, we present the experimental results in Table 4 and Figure 9. For PostgreSQL, we observe results consistent with MySQL: both baseline methods and AgentTune identify significantly better configurations than the default. Moreover,

Table 4. Experimental results on different database engines.

DB Engine	Method	Throughput	T_{opt}	PIE (%)	Invalid Times
PostgreSQL	Default	52	-	-	-
	PGTune	93.36	-	-	0
	SHAP + DDPG++	123.75	88	1.56	79
	SHAP + SMAC	140.03	76	1.94	6
	SHAP + VBO	135.52	88	1.82	17
	GPTuner	157.37	53	3.82	9
	DB-BERT	175.53	47	5.05	11
	AgentTune	**191.77**	**6**	**44.80**	**0**
TiDB	Default	164	-	-	-
	SHAP + DDPG++	109.59	71	-0.47	100
	SHAP + SMAC	212.27	42	0.70	23
	SHAP + VBO	184.31	56	0.22	45
	GPTuner	182.95	84	0.13	43
	DB-BERT	167.26	**3**	0.66	84
	AgentTune	**265.71**	10	**6.2**	**3**

AgentTune can identify the best configurations with much higher efficiency. For TiDB, its ability to automatically configure knobs based on hardware specifications [45] gives it a notable advantage in the default setting, achieving a throughput of 164.00 compared to PostgreSQL's 52.00. This built-in optimization poses a greater challenge for tuning methods, as the baseline is already highly competitive. As shown in Table 4, configurations recommended by the DDPG++ algorithm even fall short of TiDB's default. DB-BERT, while converging rapidly, also yields only limited improvements in performance on TiDB. We attribute this to the relatively scarce availability of high-quality, TiDB-specific tuning experience on the web compared to more mature systems such as MySQL and PostgreSQL. As a result, the number of useful hints that DB-BERT can extract from the collected documents is substantially constrained. Nonetheless, through iterative refinement and effective use of database feedback, AgentTune surpasses the default configuration. In addition, AgentTune also demonstrates strong results in terms of tuning efficiency and safety. These findings highlight AgentTune's robustness across heterogeneous database systems. By incorporating knob descriptions and internal metrics into prompts, AgentTune effectively leverages the linguistic and database-related knowledge implied in LLMs, enabling it to generalize to unfamiliar database engines.

8.4 Ablation Study

To further evaluate the effectiveness of each component in our framework, we conduct comprehensive ablation studies throughout the entire tuning process. In addition, we examine the impact of different LLMs on tuning performance. Unless otherwise specified, all ablation experiments are conducted on SYSBENCH.

8.4.1 Ablation study on the Workload Analyzer. First, we perform ablation studies to evaluate the effectiveness of the workload analyzer. Specifically, we consider three scenarios: (1) completely removing the workload analyzer, thereby eliminating all workload-related features from the prompts to the LLM-driven agents; (2) removing only the static features from the workload analyzer; and (3) removing only the dynamic features. The results are shown in Table 7 ("w/o WA", "w/o WA's static features", and "w/o WA's dynamic features"). Eliminating all workload features results in substantial decreases in both performance and efficiency, and also leads to 33 invalid configurations during tuning, making the process significantly more error-prone and unreliable. Similarly, removing static

Table 5. Ablation study on the knob size.

Knob Size (n)	Throughput	T_{opt}	PIE (%)	Invalid Times
5	119.31	42	14.09	**0**
10	173.85	68	15.65	**0**
20	268.03	**23**	**63.10**	**0**
40	**269.60**	**23**	50.44	**0**

features produces suboptimal results due to the absence of critical workload and hardware context. In contrast, removing dynamic features results in a relatively slight performance and efficiency degradation. These findings indicate that static features primarily determine the lower bound of performance, while dynamic features help improve the upper bound.

8.4.2 Ablation study on the Knob Selector. Since the subsequent pruning and tuning stages depend on the selected knobs, it is challenging to fully decouple the knob selector from the framework. As an alternative, we utilize a learning-based method—SHAP [37]—as a competitive baseline. SHAP is currently considered one of the most effective learning-based methods for assessing the importance of knobs [66]. As shown in Table 7 ("replace KS with SHAP"), our knob selection method significantly improves the optimization upper bound, with an increase of approximately 75.26%.

Another important consideration is the number of knobs being tuned. For this analysis, we use 5, 10, 20, and 40 knobs, as shown in Table 5. We observe that optimization performance improves with an increasing number of knobs, but the rate of improvement diminishes from 20 to 40 knobs. Additionally, more knobs lead to longer prompts and higher computational costs and tuning time. A set of 20 knobs offers a favorable trade-off between performance, efficiency, and LLM inference overhead. We do not explore larger knob spaces due to GPT-4's context length limitations. However, with advancements in models [62] supporting up to 1 million tokens, we anticipate handling larger knob spaces in the future.

8.4.3 Ablation study on the Range Pruner. We conduct an ablation study to evaluate the effectiveness of the range pruner. Specifically, we remove the range pruner as a baseline and compare it against the full version, where the module is enabled. As shown in Table 7 ("w/o RP"), the range pruner significantly improves both performance and efficiency. This improvement stems from its ability to constrain knob ranges based on workload features while handling special values that are often difficult for traditional methods to manage. More importantly, the range pruner almost prevents the occurrence of invalid configurations, which is crucial for real-world deployment.

8.4.4 Ablation study on the Knob Recommender. We remove certain components of the tree-based knob recommender in AgentTune to understand their contribution to the overall system. We compare the following versions: (1) AgentTune; (2) AgentTune-w/o Iteration, which suggests configurations without any iterations; (3) AgentTune-w/o TS, which updates using a traditional sequence-based method instead of our tree search framework by setting $R = 1$ and $k = 1$; (4) AgentTune-w/o MW, which removes the memory window from the prompt, meaning the tuning process no longer retains any historical best-performing configurations for reference; and (5) AgentTune-w/o LLM, which replaces the LLM-based tuning methods with a traditional method, SMAC.

Table 7 shows the results, and we make the following observations: (1) AgentTune-w/o iteration performs the worst in improving database throughput, suggesting that our well-designed iterative LLMs do benefit from iterations. (2) AgentTune-w/o TS exhibits much lower efficiency, and unsafe configurations emerge during the tuning process. This demonstrates the advantages of our tree-based iteration framework over traditional sequential approaches in terms of both reliability and

Table 6. Ablation study on the beam size.

Beam Size	Throughput	T_{opt}	PIE (%)	Invalid Times
1	208.69	**15**	**73.99**	**0**
2	268.03	23	63.10	**0**
4	281.31	33	46.39	**0**
8	**283.78**	39	39.62	2

Table 7. Ablation studies. KS, RP, WA, and IT represent the abbreviations of "Knob Selector", "Range Pruner", "Workload Analyzer", and "Invalid Times", respectively.

Method	Throughput	T_{opt}	PIE (%)	IT
AgentTune	**268.03**	**23**	**63.10**	**0**
- w/o WA	111.72	50	10.95	33
- w/o WA's static features	145.12	54	13.73	25
- w/o WA's dynamic features	251.35	30	45.24	**0**
- replace KS with SHAP	152.91	24	26.89	**0**
- w/o RP	200.14	29	36.82	17
- w/o iteration	154.48	-	-	**0**
- w/o TS	192.16	30	33.80	7
- w/o MW	235.67	43	29.45	23
- w/o LLM	187.25	69	14.37	**0**
- w/o kS and RP	155.30	27	34.81	**0**
- merge KS and RP	214.95	31	36.93	**0**
- w/o checks	263.43	41	34.81	6

speed of convergence. (3) After removing the memory window, we observe a notable decline in tuning performance. Without the ability to retain and refer to historical best configurations, AgentTune is constrained to react only to the most recent feedback, limiting its learning capacity over iterations. This limitation also contributes to the emergence of more invalid configurations; a closer inspection reveals that the LLM frequently makes repetitive errors on similar parameter sets due to the absence of historical experience. (4) AgentTune-w/o LLM performs worse than when using LLMs. LLMs can identify comparable or superior configurations with only a few iterations, showcasing their tuning efficiency. On the other hand, we observe that SMAC, under our tree-based tuning framework, shows improvement on all metrics, including better best-found performance, higher efficiency, and complete elimination of unsafe configurations, indicating that our tuning framework can be combined with other methods to enhance their performance.

In our tree-based tuning framework, beam width is a crucial hyperparameter that controls the number of nodes at each level of the tree. Specifically, it determines how many configurations are selected for workload replay during each iteration. We conducted tests for various values of $k = 1, 2, 4, 8$, with results shown in Table 6. As k increases, the best-found performance gradually improves, while efficiency decreases. We think this trend as analogous to the exploration vs. exploitation dilemma in reinforcement learning: a larger k allows AgentTune to explore more possible directions, increasing the likelihood of finding a better configuration, but also reducing efficiency due to some of the less meaningful trials. When $k \geq 2$, the performance improvement levels off, and the benefits are far outweighed by the loss in efficiency. Specifically, at $k = 8$, the extensive exploration even led to the occurrence of invalid configurations. Based on these findings, we choose $k = 2$ as the beam width for experiments.

8.4.5 Ablation study on the combination design of Knob Selector and Range Pruner. To evaluate the effectiveness of the knob selector and range pruner combination designs, we conduct two additional ablation studies: (1) completely removing the knob selector and range pruner from our framework while increasing the maximum number of workload replays from 100 to 200, and (2)

Table 8. Deterministic evaluation of AgentTune.

Benchmark	Performance	T_{opt}	PIE (%)	Invalid Times
SYSBENCH	282.08 ± 19.41	24 ± 0.82	64.02 ± 4.04	0 ± 0
JOB	483.49 ± 5.00	17.67 ± 1.25	4.49 ± 0.33	0 ± 0

merging the knob selector and range pruner into a single LLM call, where the LLM is instructed to first select important knobs and then determine their meaningful ranges within one call. The first ablation examines how our agent-based design compares to a baseline that does not use multiple agents but instead increases iterations during the configuration recommendation stage. The second ablation assesses the importance of separating knob selection and range pruning as distinct steps. As shown in Table 7 ("w/o KS and RP" and "merge KS and RP"), both simplified approaches result in noticeable performance degradation. Removing the knob selector and range pruner significantly increases the search space in the configuration recommendation stage, making it much harder for the LLM to find suitable configurations. Similarly, merging the two subtasks into a single LLM call reduces performance, likely because it overloads the LLM and diminishes its ability to accurately complete both tasks in a sequential, focused manner. These results highlight the necessity of both agent-based design and step-wise task decomposition for effective LLM-driven knob tuning.

8.4.6 Ablation study on the rule-based constraint. We conduct an ablation study to evaluate the effectiveness of AgentTune's rule-based constraint strategy. Specifically, we disable all rule-based white-box validations in AgentTune and compare the results to the full version. As shown in Table 7 ("w/o checks"), removing rule-based checks causes only a slight drop in performance, but tuning efficiency declines significantly, requiring 41 workload replays to find the best configuration and decreasing the PIE score from 63.10 to 34.81. Furthermore, the absence of rule-based checks leads to six invalid configurations, negatively affecting system safety. These findings highlight the crucial role of rule-based checks in LLM-driven AgentTune. On the other hand, to verify that AgentTune's performance advantage is not simply due to its use of rule-based checks, we also integrated these checks into GPTuner's tuning process. Experiments on SYSBENCH show that rule-based checks have negligible impact on GPTuner's performance: throughput drops slightly from 154.43 to 150.25, and PIE decreases from 15.59 to 14.55. Additionally, GPTuner with checks still produces 12 invalid configurations during tuning. This is because GPTuner's most invalid configurations are due to poor performance (worse than the default configuration), rather than structural infeasibility, and thus cannot be prevented by rule-based constraints.

8.4.7 Deterministic evaluation of AgentTune. During LLM inference, we use a temperature of 1.0 to generate responses, which introduces a degree of nondeterminism into the tuning process. To evaluate the stability of AgentTune, we repeat each experiment five times under identical settings on two representative benchmarks, SYSBENCH and JOB. We report the mean and standard deviation of key evaluation metrics, denoted as "mean ± std". As shown in Table 8, AgentTune achieves consistently strong mean performance and low standard deviations across runs, highlighting its robustness despite the intrinsic stochasticity of LLM outputs.

8.4.8 Effect of different LLMs. We analyze the effect of LLMs on tuning performance. Specifically, AgentTune is tested with GPT-3.5 [42] , GPT-4 [40], GPT-4o [41] and Claude-3-Opus [4]. As shown in Table 9, all versions of AgentTune outperform GPTuner, indicating that AgentTune's success is not reliant on any specific LLM.

Table 9. Effect of different LLMs.

Method	Throughput	T_{opt}	PIE (%)	Invalid Times
GPTuner	154.42	51	15.59	12
AgentTune + GPT-3.5	196.62	21	48.67	**0**
AgentTune + GPT-4	**268.03**	23	63.10	**0**
AgentTune + GPT-4o	206.61	16	69.49	**0**
AgentTune + Claude-3-Opus	228.51	**12**	**103.43**	**0**

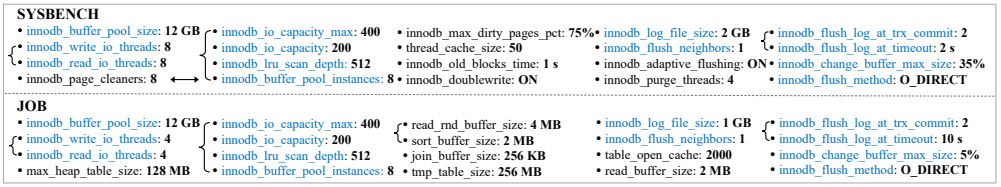

Fig. 10. AgentTune-recommended configurations on SYSBENCH and JOB. Identical selected knobs are highlighted in blue; interdependent knobs are marked with brackets or arrows.

8.5 Case Study

As shown in Figure 10, we present the best-found configurations by AgentTune for two representative workloads: SYSBENCH (OLTP) and JOB (OLAP). A detailed case study analysis follows below.

Key knob decisions drive performance improvements. For SYSBENCH (OLTP), several knobs significantly boost throughput. Increasing `innodb_buffer_pool_size` from 128M (default) to 12GB (75% of available memory) improves cache hit rates and reduces disk access. Setting `innodb_write_io_threads` and `innodb_page_cleaners` to 8 (from defaults of 4 and 1) alleviates write bottlenecks through greater parallelism. For JOB (OLAP), performance gains are achieved by enlarging `read_rnd_buffer_size`, `sort_buffer_size`, and `tmp_table_size`, which directly reduce the disk-based temporary tables during large joins and sorting.

Knob selector can capture knob correlations. We observe that the knob selector often chooses functionally related knobs together, suggesting that LLMs can capture implicit dependencies through the knob descriptions. For SYSBENCH (OLTP), the knob selector jointly selects `innodb_io_capacity`, `innodb_io_capacity_max`, and `innodb_lru_scan_depth`, along with `innodb_page_cleaners` and `innodb_buffer_pool_instances`, reflecting a systemic approach to page flushing, buffer management, and dirty page tracking. Similarly, for JOB (OLAP), the co-selection of `read_rnd_buffer_size` and `sort_buffer_size` shows AgentTune's ability to recognize their coordinated role in optimizing read-heavy operations.

AgentTune recommends different configurations for different workloads. Although certain knobs, such as `innodb_buffer_pool_size`, are consistently selected due to their fundamental importance, other knobs show strong workload-specific relevance. For SYSBENCH (OLTP), AgentTune selects knobs related to concurrency, flushing (`innodb_page_cleaners`, `innodb_purge_threads`), and durability (`innodb_max_dirty_pages_pct`) to improve throughput. In contrast, for JOB (OLAP), it prioritizes knobs such as `read_rnd_buffer_size`, `tmp_table_size`, and `sort_buffer_size` that benefit read-heavy queries. Notably, some knobs are tuned differently based on workload: for instance, `innodb_change_buffer_max_size` is set to 35% in SYSBENCH to accelerate index changes, but reduced to 5% in JOB, where insert/update operations are infrequent. This adaptive knob tuning enables AgentTune's robust, generalizable performance across workloads.

8.6 Cost Analysis

There are two main types of overhead in DBMS knob tuning: (1) initial profiling overhead and (2) runtime overhead. Initial profiling overhead refers to the time required to collect training data or pre-train models before tuning begins (*e.g.*, OtterTune [2] requires "over 30k trials per DBMS"; GPTuner spends several hours building database-related knowledge [26]). Runtime overhead is the time the tuner takes to recommend a new configuration for evaluation.

AgentTune offers out-of-the-box flexibility with minimal initial profiling overhead compared to previous methods. We evaluate the runtime overhead by presenting the number of tokens consumed, monetary costs, and time required using GPT-4. In the main experiments (Section 8.2), AgentTune incurs 669.20K tokens, costs 20.17 USD, and takes 898 seconds in total. Configuration recommendation time is generally negligible compared to workload replay time. Therefore, AgentTune's LLM-related costs are feasible and practical, making it suitable for real-world large-scale deployment.

9 Discussion

Enhanced Feedback Signals. As described in Section 7.2, we currently leverage performance metrics and execution features as feedback signals, which are both effective and aligned with common DBA practices. Beyond these, other sources of feedback—such as changes in query execution plans—may offer valuable insights, as they directly impact query behavior and reflect the influence of configuration settings. Incorporating richer database feedback is therefore a promising direction to further enhance the convergence and effectiveness of the configuration recommender.

Workload Compression. In this paper, workload replay is performed using the entire original workload. However, for complex workloads with many queries, often only a subset of representative queries substantially affects the database performance. Therefore, a promising direction is to leverage LLMs for workload compression, either by identifying these representative queries or by generating shorter-running query variants. Employing a compressed workload for replay can accelerate each tuning iteration in the configuration recommender module and further enhance overall efficiency.

10 Conclusion

Database knob tuning is a complex task due to the high-dimensional configuration space and system variability. While ML- and LLM-based methods show promise, they often struggle with efficiency, reliability, and adaptability. To address these challenges, we proposed **AgentTune**, an LLM-driven multi-agent framework that decomposes knob tuning into four subtasks: workload analysis, knob selection, range pruning, and configuration recommendation. AgentTune leverages a tree-based iterative search with centroid-distance ranking to efficiently search for high-performing configurations. It requires minimal setup and generalizes well across different environments. Extensive experiments show that AgentTune consistently outperforms existing methods in both tuning efficiency and performance, demonstrating the value of a structured multi-agent framework for practical database tuning.

Acknowledgments

This work was supported by the National Key Research & Develop Plan(2023YFB4503600) and National Natural Science Foundation of China (U23A20299, U24B20144, 62172424, 62276270, 62322214). Renata Borovica-Gajic gratefully acknowledges support from the Australian Research Council Discovery Early Career Researcher Award DE230100366.

References

[1] Roee Aharoni, Melvin Johnson, and Orhan Firat. 2019. Massively Multilingual Neural Machine Translation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 3874–3884. doi:10.18653/V1/N19-1388

[2] 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, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu (Eds.). ACM, 1009–1024. doi:10.1145/3035918.3064029

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

[4] Anthropic. 2024. Introducing the next generation of Claude. (2024). Available at: https://www.anthropic.com/news/claude-3-family.

[5] Baoqing Cai, Yu Liu, Ce Zhang, Guangyu Zhang, Ke Zhou, Li Liu, Chunhua Li, Bin Cheng, Jie Yang, and Jiashu Xing. 2022. HUNTER: An Online Cloud Database Hybrid Tuning System for Personalized Requirements. In *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, 646–659. doi:10.1145/3514221.3517882

[6] Surajit Chaudhuri and Vivek R. Narasayya. 2007. Self-Tuning Database Systems: A Decade of Progress. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, Christoph Koch, Johannes Gehrke, Minos N. Garofalakis, Divesh Srivastava, Karl Aberer, Anand Deshpande, Daniela Florescu, Chee Yong Chan, Venkatesh Ganti, Carl-Christian Kanne, Wolfgang Klas, and Erich J. Neuhold (Eds.). ACM, 3–14. http://www.vldb.org/conf/2007/papers/special/p3-chaudhuri.pdf

[7] Haifeng Chen, Wenxuan Zhang, and Guofei Jiang. 2011. Experience Transfer for the Configuration Tuning in Large-Scale Computing Systems. *IEEE Trans. Knowl. Data Eng.* 23, 3 (2011), 388–401. doi:10.1109/TKDE.2010.121

[8] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. *CoRR* abs/2107.03374 (2021). arXiv:2107.03374 https://arxiv.org/abs/2107.03374

[9] Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George F. Foster, Llion Jones, Mike Schuster, Noam Shazeer, Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. 2018. The Best of Both Worlds: Combining Recent Advances in Neural Machine Translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, Iryna Gurevych and Yusuke Miyao (Eds.). Association for Computational Linguistics, 76–86. doi:10.18653/V1/P18-1008

[10] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024. Teaching Large Language Models to Self-Debug. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. https://openreview.net/forum?id=KuPixIqPiq

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL] https://arxiv.org/abs/1810.04805

[12] Harish Doraiswamy, Pooja N. Darera, and Jayant R. Haritsa. 2008. Identifying robust plans through plan diagram reduction. *Proc. VLDB Endow.* 1, 1 (2008), 1124–1140. doi:10.14778/1453856.1453976

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

[14] Chongjiong Fan, Zhicheng Pan, Wenwen Sun, Chengcheng Yang, and Wei-Neng Chen. 2024. LATuner: An LLM-Enhanced Database Tuning System Based on Adaptive Surrogate Model. In *Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2024, Vilnius, Lithuania, September 9–13, 2024, Proceedings, Part V* (Vilnius, Lithuania). Springer-Verlag, Berlin, Heidelberg, 372–388. doi:10.1007/978-3-031-70362-1_22

[15] Ju Fan, Zihui Gu, Songyue Zhang, Yuxin Zhang, Zui Chen, Lei Cao, Guoliang Li, Samuel Madden, Xiaoyong Du, and Nan Tang. 2024. Combining Small Language Models and Large Language Models for Zero-Shot NL2SQL. *Proc. VLDB Endow.* 17, 11 (2024), 2750–2763. https://www.vldb.org/pvldb/vol17/p2750-fan.pdf

[16] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *Proc. VLDB Endow.* 17, 5 (2024), 1132–1145. https://www.vldb.org/pvldb/vol17/p1132-gao.pdf

[17] Victor Giannakouris and Immanuel Trummer. 2025. λ-Tune: Harnessing Large Language Models for Automated Database System Tuning. *Proc. ACM Manag. Data* 3, 1, Article 2 (Feb. 2025), 26 pages. doi:10.1145/3709652

[18] Yuxing Han, Ziniu Wu, Peizhi Wu, Rong Zhu, Jingyi Yang, Liang Wei Tan, Kai Zeng, Gao Cong, Yanzhao Qin, Andreas Pfadler, Zhengping Qian, Jingren Zhou, Jiangneng Li, and Bin Cui. 2021. Cardinality Estimation in DBMS: A Comprehensive Benchmark Evaluation. *Proc. VLDB Endow.* 15, 4 (2021), 752–765. doi:10.14778/3503585.3503586

[19] Major Hayden. 2024. MySQLTuner — A script to review and tune your MySQL installation. https://github.com/major/MySQLTuner-perl. Accessed: 2025-07-22.

[20] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The Curious Case of Neural Text Degeneration. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=rygGQyrFvH

[21] Dongxu Huang, Qi Liu, Qiu Cui, Zhuhe Fang, Xiaoyu Ma, Fei Xu, Li Shen, Liu Tang, Yuxing Zhou, Menglong Huang, Wan Wei, Cong Liu, Jian Zhang, Jianjun Li, Xuelian Wu, Lingyu Song, Ruoxi Sun, Shuaipeng Yu, Lei Zhao, Nicholas Cameron, Liquan Pei, and Xin Tang. 2020. TiDB: A Raft-based HTAP Database. *Proc. VLDB Endow.* 13, 12 (2020), 3072–3084. doi:10.14778/3415478.3415535

[22] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. 2023. A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions. *CoRR* abs/2311.05232 (2023). doi:10.48550/ARXIV.2311.05232 arXiv:2311.05232

[23] Xinmei Huang, Haoyang Li, Jing Zhang, Xinxin Zhao, Zhiming Yao, Yiyan Li, Tieying Zhang, Jianjun Chen, Hong Chen, and Cuiping Li. 2025. E2ETune: End-to-End Knob Tuning via Fine-tuned Generative Language Model. arXiv:2404.11581 [cs.AI] https://arxiv.org/abs/2404.11581

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

[25] Alexey Kopytov. 2024. Scriptable database and system performance benchmark. (2024). Available at: https://github.com/akopytov/sysbench/.

[26] Jiale Lao, Yibo Wang, Yufei Li, Jianping Wang, Yunjia Zhang, Zhiyuan Cheng, Wanghu Chen, Mingjie Tang, and Jianguo Wang. 2024. GPTuner: A Manual-Reading Database Tuning System via GPT-Guided Bayesian Optimization. *Proc. VLDB Endow.* 17, 8 (2024), 1939–1952. https://www.vldb.org/pvldb/vol17/p1939-tang.pdf

[27] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? *Proc. VLDB Endow.* 9, 3 (2015), 204–215. doi:10.14778/2850583.2850594

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

[29] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023. RESDSQL: Decoupling Schema Linking and Skeleton Parsing for Text-to-SQL. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, Brian Williams, Yiling Chen, and Jennifer Neville (Eds.). AAAI Press, 13067–13075. doi:10.1609/AAAI.V37I11.26535

[30] 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. *Proc. ACM Manag. Data* 2, 3 (2024), 127.

[31] Yujia Li, David H. Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin 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). doi:10.48550/ARXIV.2203.07814 arXiv:2203.07814

[32] Yiyan Li, Haoyang Li, Zhao Pu, Jing Zhang, Xinyi Zhang, Tao Ji, Luming Sun, Cuiping Li, and Hong Chen. 2024. Is Large Language Model Good at Database Knob Tuning? A Comprehensive Experimental Evaluation. arXiv:2408.02213 [cs.DB] https://arxiv.org/abs/2408.02213

[33] Zhaodonghui Li, Haitao Yuan, Huiming Wang, Gao Cong, and Lidong Bing. 2024. LLM-R2: A Large Language Model Enhanced Rule-based Rewrite System for Boosting Query Efficiency. *CoRR* abs/2404.12872 (2024). doi:10.48550/ARXIV.2404.12872 arXiv:2404.12872

[34] Chen Lin, Junqing Zhuang, Jiadong Feng, Hui Li, Xuanhe Zhou, and Guoliang Li. 2022. Adaptive Code Learning for Spark Configuration Tuning. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur,*

Malaysia, May 9-12, 2022. IEEE, 1995–2007. doi:10.1109/ICDE53745.2022.00195

[35] Jie Liu and Barzan Mozafari. 2024. Query Rewriting via Large Language Models. *CoRR* abs/2403.09060 (2024). doi:10.48550/ARXIV.2403.09060 arXiv:2403.09060

[36] Junling Liu, Peilin Zhou, Yining Hua, Dading Chong, Zhongyu Tian, Andrew Liu, Helin Wang, Chenyu You, Zhenhua Guo, Lei Zhu, and Michael Lingzhi Li. 2023. Benchmarking Large Language Models on CMExam - A comprehensive Chinese Medical Exam Dataset. 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/a48ad12d588c597f4725a8b84af647b5-Abstract-Datasets_and_Benchmarks.html

[37] Scott M. Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 4765–4774. https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html

[38] Michael D. McKay. 1992. Latin Hypercube Sampling as a Tool in Uncertainty Analysis of Computer Models. In *Proceedings of the 24th Winter Simulation Conference, Arlington, VA, USA, December 13-16, 1992*, Robert C. Crain (Ed.). ACM Press, 557–564. doi:10.1145/167293.167637

[39] Seyed-Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. 2024. GSM-Symbolic: Understanding the Limitations of Mathematical Reasoning in Large Language Models. *CoRR* abs/2410.05229 (2024). doi:10.48550/ARXIV.2410.05229 arXiv:2410.05229

[40] OpenAI. 2023. GPT-4 Technical Report. *CoRR* abs/2303.08774 (2023). doi:10.48550/ARXIV.2303.08774 arXiv:2303.08774

[41] OpenAI. 2024. Hello gpt-4o. (2024). Available at: https://openai.com/index/hello-gpt-4o/.

[42] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems* 35 (2022), 27730–27744.

[43] PENG SI OW and THOMAS E. MORTON and. 1988. Filtered beam search in scheduling†. *International Journal of Production Research* 26, 1 (1988), 35–62. doi:10.1080/00207548808947840 arXiv:https://doi.org/10.1080/00207548808947840

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

[45] PingCAP. 2024. TiDB v8.2 documentation. https://docs-archive.pingcap.com/zh/tidb/v8.2/ Version 8.2.

[46] 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/72223cc66f63ca1aa59edaec1b3670e6-Abstract-Conference.html

[47] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. https://openreview.net/forum?id=dHng2O0Jjr

[48] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[49] 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, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal V. Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Févry, Jason Alan Fries, Ryan Teehan, Teven Le Scao, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M. Rush. 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. https://openreview.net/forum?id=9Vrb9D0WI4

[50] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. 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/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html

[51] 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. *Proc. VLDB Endow.* 12, 10 (2019), 1221–1234. doi:10.14778/3339490.3339503

[52] Robert Tibshirani. 1996. Regression Shrinkage and Selection via the Lasso. *Journal of the royal statistical society series b-methodological* 58 (1996), 267–288. https://api.semanticscholar.org/CorpusID:16162039

[53] Transaction Processing Performance Council (TPC). 2010. TPC Benchmark C Standard Specification. http://www.tpc.org/tpcc/ Version 5.11.0.

[54] Transaction Processing Performance Council (TPC). 2024. TPC Benchmark DS Standard Specification. https://www.tpc.org/tpcds/default5.asp Version 4.0.0.

[55] 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.

[56] Immanuel Trummer. 2024. DB-BERT: making database tuning tools “read” the manual. *The VLDB Journal* 33 (2024), 1085–1104. doi:10.1007/s00778-023-00831-y Published: 27 December 2023.

[57] Kostas Tzoumas, Amol Deshpande, and Christian S. Jensen. 2011. Lightweight Graphical Models for Selectivity Estimation Without Independence Assumptions. *Proc. VLDB Endow.* 4, 11 (2011), 852–863. http://www.vldb.org/pvldb/vol4/p852-tzoumas.pdf

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

[59] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5998–6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[60] Oleksii Vasyliev. 2024. Pgtune - tuning PostgreSQL config by your hardware. https://github.com/le0pard/pgtune. Accessed: 2025-07-22.

[61] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. 2023. The Rise and Potential of Large Language Model Based Agents: A Survey. *CoRR* abs/2309.07864 (2023). doi:10.48550/ARXIV.2309.07864 arXiv:2309.07864

[62] An Yang, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoyan Huang, Jiandong Jiang, Jianhong Tu, Jianwei Zhang, Jingren Zhou, Junyang Lin, Kai Dang, Kexin Yang, Le Yu, Mei Li, Minmin Sun, Qin Zhu, Rui Men, Tao He, Weijia Xu, Wenbiao Yin, Wenyuan Yu, Xiafei Qiu, Xingzhang Ren, Xinlong Yang, Yong Li, Zhiying Xu, and Zipeng Zhang. 2025. Qwen2.5-1M Technical Report. *CoRR* abs/2501.15383 (2025). doi:10.48550/ARXIV.2501.15383 arXiv:2501.15383

[63] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d’Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 5754–5764. https://proceedings.neurips.cc/paper/2019/hash/dc6a7e655d7e5840e66733e9ee67cc69-Abstract.html

[64] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net. https://openreview.net/forum?id=WE_vluYUL-X

[65] Ji Zhang, Ke Zhou, Guoliang Li, Yu Liu, Ming Xie, Bin Cheng, and Jiashu Xing. 2021. CDBTune: An efficient deep reinforcement learning-based automatic cloud database tuning system. *VLDB J.* 30, 6 (2021), 959–987. doi:10.1007/S00778-021-00670-9

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

[67] 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 *SIGMOD ’21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 2102–2114. doi:10.1145/3448016.3457291

[68] Xinyi Zhang, Hong Wu, Yang Li, Jian Tan, Feifei Li, and Bin Cui. 2022. Towards Dynamic and Safe Configuration Tuning for Cloud Databases. 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, 631–645. doi:10.1145/3514221.3526176

[69] 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.

[70] Xinyang Zhao, Xuanhe Zhou, and Guoliang Li. 2023. Automatic Database Knob Tuning: A Survey. *IEEE Trans. Knowl. Data Eng.* 35, 12 (2023), 12470–12490. doi:10.1109/TKDE.2023.3266893

[71] Yue Zhao, Gao Cong, Jiachen Shi, and Chunyan Miao. 2022. QueryFormer: a tree transformer model for query plan representation. *Proc. VLDB Endow.* 15, 8 (April 2022), 1658–1670. doi:10.14778/3529337.3529349

[72] Xuanhe Zhou, Guoliang Li, Zhaoyan Sun, Zhiyuan Liu, Weize Chen, Jianming Wu, Jiesi Liu, Ruohang Feng, and Guoyang Zeng. 2024. D-Bot: Database Diagnosis System using Large Language Models. *Proc. VLDB Endow.* 17, 10 (2024), 2514–2527. https://www.vldb.org/pvldb/vol17/p2514-li.pdf

[73] Yuqing Zhu, Jianxun Liu, Mengying Guo, Yungang Bao, Wenlong Ma, Zhuoyue Liu, Kunpeng Song, and Yingchun Yang. 2017. BestConfig: tapping the performance potential of systems via automatic configuration tuning. In *Proceedings of the 2017 Symposium on Cloud Computing, SoCC 2017, Santa Clara, CA, USA, September 24-27, 2017.* ACM, 338–350. doi:10.1145/3127479.3128605

Received April 2025; revised July 2025; accepted August 2025