ESTune: Bayesian Uncertainty-Guided Early Stopping for Database Configuration Tuning

ZHONGWEI YUE, College of Software, Henan Normal University, China
JUN-PENG ZHU, East China Normal University, China
PENG CAI*, East China Normal University, China
XUAN ZHOU, East China Normal University, China
QUANQING XU, OceanBase, Ant Group, China
CHUANHUI YANG, OceanBase, Ant Group, China

Existing database knob tuning methods evaluate the actual performance of each configuration by fully executing the entire workload. However, our experimental analysis reveals that this exhaustive execution approach significantly limits tuning efficiency, particularly when dealing with underperforming configurations. To address this issue, we propose ESTune, which is designed to early-stop the execution of poorly performing configurations. ESTune approximates the actual performance of these configurations using *high-confidence predicted values* generated from partially executed workload data and configuration knob settings. This strategy significantly reduces the evaluation time for underperforming configurations while maintaining the overall tuning effectiveness. The high-confidence predicted values are produced by a Hybrid Bayesian Neural Network (HBNN), which models the performance distribution with respect to different knob configurations. To address the challenge of limited training data commonly encountered in database knob tuning, ESTune integrates a Model-Agnostic Meta-Learning (MAML), thereby enhancing the few-shot learning capability of the HBNN. Extensive evaluations on a wide range of workloads consistently demonstrate that ESTune improves the tuning efficiency of existing methods.

CCS Concepts: • **Information systems → Autonomous database administration**; • **Computing methodologies → Machine learning**.

Additional Key Words and Phrases: Database Knob Tuning, Early Stopping, Tuning Efficiency

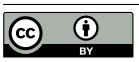
ACM Reference Format:
Zhongwei Yue, Jun-Peng Zhu, Peng Cai, Xuan Zhou, Quanqing Xu, and Chuanhui Yang. 2026. ESTune: Bayesian Uncertainty-Guided Early Stopping for Database Configuration Tuning. *Proc. ACM Manag. Data* 4, 1 (SIGMOD), Article 35 (February 2026), 26 pages. https://doi.org/10.1145/3786649

1 Introduction

Modern database management systems (DBMSs) are equipped with numerous tunable knobs (or parameters). The performance of a DBMS varies considerably with different configuration knob settings across a wide range of tuning scenarios, such as varying workloads [51, 52] and hardware configurations. Traditionally, database administrators (DBAs) have relied on their expertise to

*Peng Cai is the corresponding author.

Authors' Contact Information: Zhongwei Yue, yuezhongwei@htu.edu.cn, College of Software, Henan Normal University, Xinxiang, China; Jun-Peng Zhu, East China Normal University, Shanghai, China, zjp.dase@stu.ecnu.edu.cn; Peng Cai, East China Normal University, Shanghai, China, pcai@dase.ecnu.edu.cn; Xuan Zhou, East China Normal University, Shanghai, China, xzhou@dase.ecnu.edu.cn; Quanqing Xu, OceanBase, Ant Group, Hangzhou, China, xuquanqing.xqq@oceanbase.com; Chuanhui Yang, OceanBase, Ant Group, Hangzhou, China, rizhao.ych@oceanbase.com.



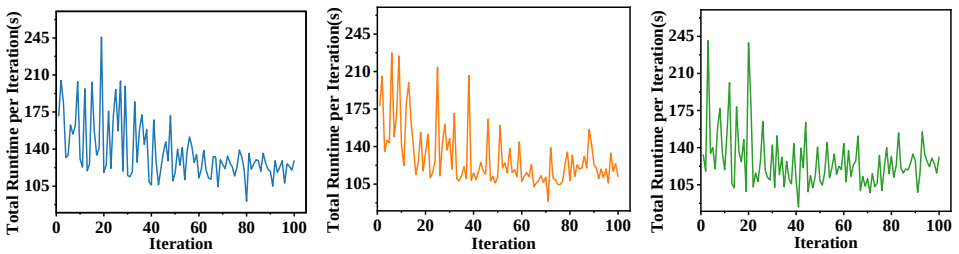
ACM 2836-6573/2026/2-ART35
https://doi.org/10.1145/3786649

manually configure these knobs to optimize database performance. However, the transition from on-premise systems to cloud computing, coupled with increasing workload complexity and a growing number of parameters, has rendered manual tuning increasingly challenging [11, 17, 24, 35, 42, 50]. As a result, the development of automated tools to assist DBAs in knob configuration optimization has become imperative.

Identifying the optimal configuration within the configuration knob space has been proven to be an NP-hard problem [33]. To address this challenge, heuristic iterative strategies are commonly employed to progressively optimize database performance. Various iterative tuning methods have been proposed, including LlamaTune (SMAC), where LlamaTune [17] is used to enhance the Sequential Model-based Algorithm Configuration (SMAC) approach [16], as well as OtterTune [1] and HUNTER [4]. Each of these methods follows a two-step iterative process, as detailed in Section 2. First, the given workload is fully executed to evaluate the database performance under the current configuration. Subsequently, new knob configurations are computed and applied based on the observed performance data. This iterative process continues until a predefined number of iterations is completed or the allocated tuning time is reached.

During the iterative tuning process, we observed that fully executing the workload in each iteration significantly limits the efficiency of iterative tuning methods. Two key findings support this observation.

(1) Exploration-exploitation trade-off causes non-monotonic tuning progress. Knob tuning typically involves numerous iterations; however, these iterations do not always result in performance improvement. As illustrated in Figure 1, the database performance achieved by three different tuning methods (i.e., OtterTune, HUNTER, and LlamaTune (SMAC)) is shown over 100 iterations, where the x-axis represents the iteration number and the y-axis denotes the total time taken to fully execute the workload in each iteration (i.e., total runtime per iteration). Experimental results reveal that the database performance obtained in each iteration exhibits noticeable fluctuations and does not show monotonic improvement. The lack of consistent performance improvement across iterations can be attributed to the inherent trade-off between exploration (i.e., acquiring new information) and exploitation (i.e., utilizing existing knowledge) in the tuning process.



(a) OtterTune (b) HUNTER (c) LlamaTune(SMAC)

Fig. 1. Per-iteration results of MySQL under the TPC-H

(2) Well-performing configurations drive tuning effectiveness despite noise. Minor inaccuracies in evaluating poorly performing configurations do not significantly impact the effectiveness of iterative tuning methods. To examine this effect, we developed variants of OtterTune, HUNTER, and LlamaTune (SMAC), denoted as V_OtterTune, V_HUNTER, and V_LlamaTune (SMAC), respectively. In these variants, the iterative process was modified as follows. If the performance of the current configuration was less than 15% of the median value from all previous iterations, it was considered a poorly performing configuration. In such cases, the performance was substituted

with a value randomly selected between 85% and 115% of the evaluation value obtained from fully executing the workload for that iteration. Figure 2 compares the final tuning results of the original methods and their variants, where the y-axis represents the best workload execution time observed over 100 iterations (i.e., total runtime). The results show that both the original methods and their variants achieve similar performance levels, suggesting that minor inaccuracies in evaluating poorly performing configurations have a negligible effect on overall tuning effectiveness. This limited impact occurs because subsequent recommendations are primarily guided by data from well-performing configurations.

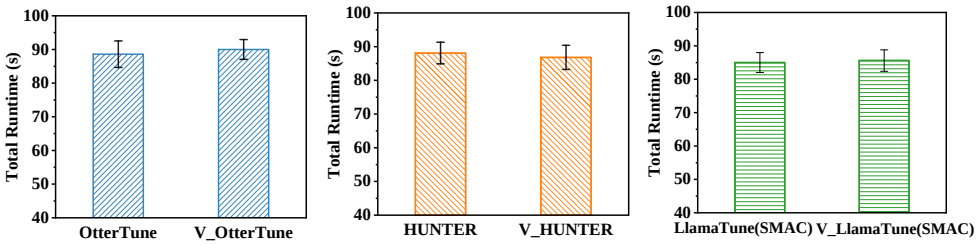


Fig. 2. The final results of MySQL under the TPC-H

Building on this robustness, a time-based trigger (i.e., terminating the workload execution upon reaching a predefined time threshold) combined with scaling the actual performance by a random factor within the interval [85%, 115%] might appear sufficient. However, this approach ultimately fails for two critical reasons. First, a time-based trigger can terminate execution only when a predefined threshold is reached, and thus cannot early-stop configurations that clearly underperform before that point, thereby incurring unnecessary time overhead. Second, and more critically, during tuning, the actual performance of an underperforming configuration is unknown, so scaling its actual performance by a random factor within [85%, 115%] cannot be implemented in practice. Moreover, the time-based trigger lacks any mechanism for predicting the performance of these early-stopped configurations. It is therefore forced to substitute their performance with random values, which can deviate substantially from the actual performance and, in turn, degrade the final tuning results.

The first observation reveals that not all evaluations contribute equally to tuning progress, suggesting that computational resources spent on thoroughly evaluating unpromising configurations could be better allocated. The second observation demonstrates that the tuning process is robust to evaluation noise for poor configurations, indicating that approximate evaluations may suffice for these cases. Building on this robustness, we can reasonably assume that prediction models, which typically achieve accuracy within similar error bounds for underperforming configurations, can serve as reliable substitutes for full evaluation. Therefore, we propose an early-stopping strategy for iterative tuning methods. Specifically, it is often unnecessary to fully execute the entire workload to evaluate the actual performance of underperforming configurations. Instead, the predicted performance of these configurations can serve as a reliable proxy for their actual performance. This strategy significantly improves tuning efficiency by reducing the evaluation time for underperforming configurations while maintaining overall effectiveness.

Effectively implementing the aforementioned early-stopping tuning strategy requires addressing the following challenges: **(C1)** timely termination of workload execution for underperforming configurations, which is essential for significantly improving tuning efficiency by avoiding unnecessary workload execution; **(C2)** accurate prediction of the performance of these underperforming configurations, which is critical for maintaining tuning effectiveness; and **(C3)** effective adaptation

of the predictive model to limited-sample scenarios, which are frequently encountered in database knob tuning.

To address challenge **C1**, we employ a segment performance monitoring mechanism. Segment performance refers to database performance periodically collected at fixed intervals (e.g., every 5 seconds during a 200-second workload evaluation). Leveraging new segment performance data as they become available during workload execution, the HBNN promptly estimates the performance of the current configuration. ESTune's early stopping mechanism halts the workload execution when underperformance is predicted, significantly reducing unnecessary computation time and improving tuning efficiency.

To address challenge **C2**, we develop a tailored HBNN (Hybrid Bayesian Neural Network) specifically for the task of performance prediction. It incorporates a Feedforward Neural Network (FNN) to encode knob configurations and a Gated Recurrent Unit (GRU) [6] to process segment performance data with sequential characteristics. The outputs of the FNN and GRU are fused via a Bayesian Neural Network (BNN) layer [12, 34], which enables accurate performance prediction and principled uncertainty quantification. As a result, only predictions with low uncertainty are used in early-stopping decisions, ensuring both reliability and accuracy.

To address challenge **C3**, we incorporate MAML to endow HBNN with few-shot adaptation capability, enabling efficient tuning in limited-sample scenarios by leveraging meta-learning across diverse tuning tasks.

Building on the aforementioned techniques, we propose ESTune, which is designed to stop workload execution early for configurations that exhibit poor performance. It is important to note that, although ESTune shares a common motivation with early stopping mechanisms in hyperparameter optimization (HPO) for machine learning [19, 22, 26], it differs in two key aspects.

(1) The targets of early stopping are different. In HPO, model training on large datasets is extremely time-consuming. As a result, all configurations are initially trained with limited resources, such as smaller datasets or fewer training iterations, and underperforming configurations are eliminated through early stopping. The remaining configurations are then iteratively evaluated with progressively increased resources until the best configuration is identified using the full resource allocation. In contrast, early stopping in ESTune is applied to database workloads. Because there can be substantial performance differences between partial and full workload executions, filtering out underperforming configurations based on partial workload results, as is commonly done in HPO, is not feasible. Furthermore, the fundamental units within a database workload, namely SQL statements, exhibit interdependencies. To capture these interdependencies, ESTune employs a GRU model to process database segment performance data, which captures the evolution of system performance during workload execution.

(2) The approach to performance evaluation for early-stopped configurations also differs. In HPO, performance is evaluated on a separate test set. However, in database tuning, the performance of a configuration can only be determined after the full execution of the workload. When early stopping is applied to database workloads, the ground-truth performance evaluation is not directly obtainable. Therefore, ESTune leverages both configuration data and partial segment performance data before early stopping, using HBNN to predict the overall performance of each configuration.

Key Contributions. Our contributions are summarized as follows.

(1) We introduce ESTune, which improves the efficiency of iterative tuning methods by incorporating early stopping in workload executions for underperforming configurations.

(2) We utilize HBNN to predict database performance across configurations and quantify uncertainties, both of which effectively support early stopping.

(3) To address the challenge of limited training data, we leverage MAML to enhance the few-shot learning capabilities of HBNN.

(4) We conduct comprehensive experiments on various workloads and databases, demonstrating that ESTune significantly improves the tuning efficiency of existing iterative methods.

2 Problem Statement

Definition 2.1. Database Performance is defined as the overall performance of the database when the entire workload is executed to completion. The primary objective of database knob tuning is to enhance database performance.

Definition 2.2. Database Segment Performance refers to the performance observed during short intervals of workload execution, such as the average throughput calculated every 5 seconds during a 200-second workload execution, or the total runtime measured for a group of 5 queries in a sequence of 200.

Consider a scenario involving n database knobs to be tuned, denoted as $knob_1, knob_2, \ldots, knob_n$. Each knob $knob_i$ is associated with a value domain Θ_i, which may be continuous, discrete, or categorical depending on the knob type. The configuration space is thus defined as $\Theta = \Theta_1 \times \Theta_2 \times \ldots \times \Theta_n$. Let y denote a performance metric of interest (e.g., throughput or total runtime), quantified by an objective function $f : \Theta \rightarrow \mathbb{R}$ that maps each configuration $x \in \Theta$ to its corresponding performance value. Given a specified metric, the goal of knob tuning is to identify the optimal configuration x_* that maximizes or minimizes $f(x)$, formally:

$$x_* = \arg\max_{x \in \Theta} f(x). \tag{1}$$

Each iteration in iterative tuning methods typically follows a structured process. Formally, in the i-th iteration, the method $Meth$ determines the next configuration x_{i+1} based on the current configuration x_i, its corresponding performance metric y_i, and the historical dataset D_{i-1}, which comprises configuration–performance tuples from all previous iterations:

$$\begin{aligned} x_{i+1} &= Meth\left(D_{i-1} \cup \{(x_i, y_i)\}\right), \\ \text{s.t.} \quad y_i &= f_w(x_i), \end{aligned} \tag{2}$$

where $y_i = f_w(x_i)$ denotes the database performance y_i obtained by fully executing the entire workload w under the configuration x_i.

Motivated by the two observations detailed in Section 1, we propose ESTune to enhance the efficiency of existing tuning methods by enabling early termination of workload executions for underperforming configurations. Specifically, in each iteration, ESTune predicts the database performance distribution, denoted as $p_d(x_i, D_s)$, using the current configuration x_i and the segment performance dataset D_s collected during the current iteration. If this distribution indicates that the current configuration is underperforming (i.e., $I_{\text{poor}}(p_d(x_i, D_s))$ = true), ESTune terminates the workload execution. The mean of the predicted distribution (i.e., $\mu(p_d(x_i, D_s))$) is then recorded as the performance of that configuration. Otherwise, workload execution continues and new segment data are incorporated into D_s. This updated dataset is then used to re-evaluate the performance distribution and determine whether to terminate execution. If the predicate $I_{\text{poor}}(p_d(x_i, D_s))$ evaluates to false at every prediction stage, it signifies that the early stopping criterion was not met. The workload thus runs to completion, and its final indicator I_{poor} is recorded as $false$ (i.e., I_{poor} =

false). This procedure is formalized as follows:

$$x_{i+1} = Meth\left(D_{i-1} \cup \{(x_i, y_i)\}\right),$$
$$\text{s.t.} \quad y_i = \begin{cases} f_w(x_i), & I_{\text{poor}} = \text{false} \\ \mu(p_d(x_i, D_s)), & I_{poor}(p_d(x_i, D_{rt})) = \text{true} \end{cases} \tag{3}$$

3 Overview of ESTune

The overall architecture of ESTune is shown in Figure 3. ESTune consists of a client and a server. The client, deployed in the user's environment, collects knob configurations and segment performance data, and transmits the collected data to the server. The server, located in the tuning environment, predicts the performance distribution of the current configuration. If the predicted performance distribution indicates that the configuration is underperforming, the server instructs the client to terminate workload execution and records the mean of the predicted distribution as the database performance of that configuration. Upon completion of either the full workload execution or early stopping, the server generates a new configuration based on the related data and sends it to the client. The client then applies the new configuration, executes the workload, and initiates the next tuning iteration. Next, we detail the core components of the architecture.

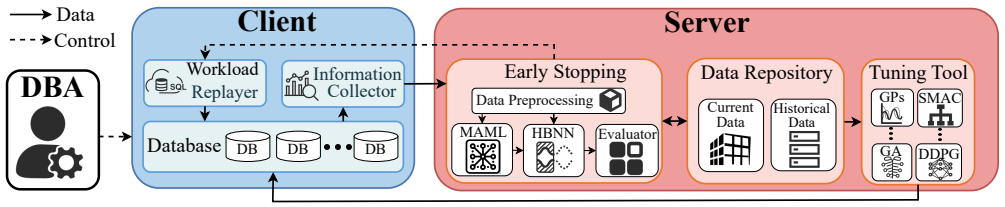


Fig. 3. Architecture of ESTune

Workload Replayer. Upon receiving a tuning signal, the client activates the workload replayer to execute the target workload on the database instance. If the early stopping components issue a termination command, the execution is promptly halted, thus preventing unnecessary time consumption on underperforming configurations.

Information Collector. The information collector is responsible for aggregating all necessary data for the server-side components. For early stopping, it collects the current knob configurations and segment performance data. For the tuning models, it collects additional contextual information, such as the runtime status of the database system. This collected data enables the server to make informed decisions regarding both early stopping and subsequent configuration recommendations.

Early Stopping. The early stopping component utilizes the HBNN to predict the performance distribution for each configuration. If the predicted distribution suggests that a configuration is underperforming, the component issues a termination signal to the workload replayer and subsequently records the mean of the predicted distribution as the performance metric. This strategy not only preserves the effectiveness of the tuning method, but also substantially reduces evaluation time, thereby improving overall tuning efficiency. Furthermore, ESTune leverages the MAML to learn robust initialization parameters for the HBNN, thereby enabling the HBNN to rapidly adapt to new tuning scenarios with limited data.

Data Repository. The data repository stores both current and historical data. Current data are generated during the ongoing tuning task, while historical data are retained from previous tuning tasks. During tuning, the repository serves as the primary data source for the early stopping component and the tuning model component, thereby supporting data-driven decision-making.

Tuning Model. The tuning model incorporates various knob configuration recommendation algorithms that have been adopted by state-of-the-art tuning methods, including genetic algorithms (GA) [18] in HUNTER, Gaussian process (GP) [32] in OBTune, OtterTune [1] and iTuned [32], SMAC [16] in LlamaTune (SMAC) [16], and Deep Deterministic Policy Gradient (DDPG) [38] in CDBTune [38]. These algorithms guide the search for optimal knob settings by leveraging performance feedback and historical data.

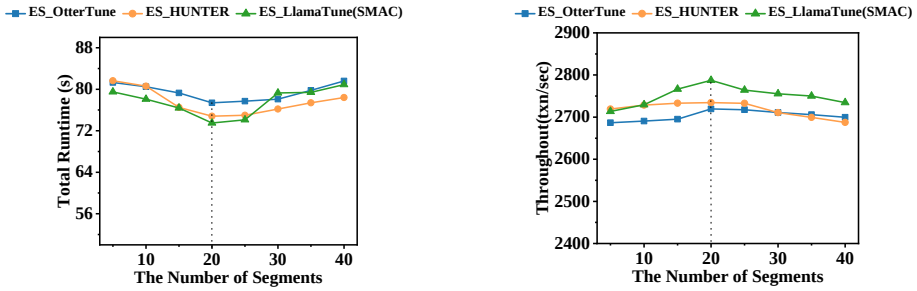
4 Segment-Aware Mechanism of the Information Collector

In this section, we elaborate on the process through which the information collector gathers the relevant data.

ESTune predicts database performance by jointly leveraging configuration data and segment performance data. This joint utilization provides two primary benefits. First, it enriches the model feature space, thereby improving prediction accuracy. Second, it enables continuous, fine-grained performance prediction for each configuration as new segment data becomes available. This, in turn, facilitates timely early stopping for underperforming configurations and reduces unnecessary time overhead. Notably, ESTune deliberately avoids using SQL statements as model features for several key reasons. First, incorporating query-level features would decrease the adaptability of HBNN to unseen workloads. Second, including SQL statements would substantially increase the amount of required training data, which is often scarce in practical tuning scenarios. Third, the heterogeneity and syntactic complexity of SQL queries across types (e.g., SELECT, INSERT, UPDATE, DELETE) make it challenging to effectively encode them as model features.

For database segment performance collection, the information collector applies tailored segmentation strategies according to the execution mode of the workload. Workloads are generally categorized as either time-based or volume-based [1]. Time-based workloads, which run for a predetermined duration, are typical in Online Transaction Processing (OLTP) scenarios characterized by numerous short queries. For such workloads, the collector divides execution into fixed time intervals and records performance metrics for each segment. Volume-based workloads, which execute a fixed set of queries that are typically longer in duration, are common in Online Analytical Processing (OLAP) environments. In this case, the collector divides the workload by a specified number of query statements and records metrics for each segment. For example, consider a TPC-C workload with a total execution time of 500 seconds: if the workload is divided into 10 time-based segments, the collector records throughput at every 50-second interval. Similarly, for a TPC-H workload consisting of 22 queries, the workload is segmented into 22 parts, with each segment corresponding to a single query. The collector then records segment performance metrics, such as latency, for each individual query.

The choice of segmentation granularity (num_{seg}) for each workload is crucial, as it must balance prediction accuracy and tuning efficiency. Setting num_{seg} too high produces excessively fine-grained segments, which increases the input sequence length for the GRU model and may introduce noise or overfitting, thereby diminishing prediction accuracy. Conversely, setting num_{seg} too low leads to overly coarse segmentation, potentially delaying the early termination of underperforming configurations and thus reducing tuning efficiency. We empirically evaluated this trade-off, as shown in Figure 4, where ES_OtterTune, ES_HUNTER, and ES_LlamaTune(SMAC) represent ESTune applied to OBTune, HUNTER, and LlamaTune (SMAC), respectively. The results demonstrate that a segmentation granularity of 20 achieves an optimal balance between accuracy and efficiency. Therefore, we set num_{seg} to 20 in all subsequent experiments.



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

Fig. 4. Results for different num_{seg} values on MySQL

5 The Design and Implementation of Early Stopping

In this section, we provide a detailed description of the four modules that comprise the early stopping component: data preprocessing, HBNN, evaluator, and MAML. The data preprocessing module enhances the quality of input features for subsequent prediction tasks. The HBNN module predicts both the mean and variance of the performance distribution for each configuration. The evaluator determines whether to terminate workload execution early based on these predicted results. Finally, the MAML algorithm enables the HBNN to adapt to new tuning scenarios with limited training samples rapidly.

5.1 Data Preprocessing

The values of configuration knobs typically have heterogeneous ranges, and distinct configurations can often induce significant disparities in observed segment performance. To ensure robust model training and accurate performance prediction, it is crucial to standardize all input features before modeling. Specifically, we apply min-max normalization to each feature, mapping its values to the interval [0,1]. This normalization enhances training stability and predictive accuracy, improving model generalization and enabling effective adaptation to diverse workloads.

5.2 Implementation of HBNN

The HBNN module consists of three units: the FNN, the GRU, and the BNN. Specifically, the FNN encodes the configuration features, the GRU captures dependencies from the sequence of segment performance metrics, and the BNN fuses the outputs of the FNN and GRU to predict the performance distribution for the given database configuration.

5.2.1 FNN for Configuration Encoding. In ESTune, the FNN is specifically employed to encode knob configuration data. Given that knob configuration can be naturally represented as a fixed-length feature vector, the FNN is well-suited for extracting informative representations from such structured inputs with high training efficiency. Concretely, the FNN consists of an input layer and an output layer, with one or more hidden layers in between. For a given configuration vector $\mathbf{V}$, the pre-activation value of the j-th neuron is computed as follows:

$$z_j = \sum_i w_{ij} v_i + b_j \tag{4}$$

where v_i denotes the i-th feature in the configuration vector, w_{ij} is the weight for the connection from the i-th input to the j-th neuron, and b_j is the bias term of the j-th neuron. The output z_j is

subsequently passed through a sigmoid activation function, whose smooth and bounded characteristics facilitate stable training and enable the nonlinear transformation of knob configuration features.

5.2.2 GRU for Encoding Segment Performance Data. Accurately predicting database performance requires the incorporation of fine-grained segment performance data generated during workload execution. This data provides localized and timely evidence of how the database responds to specific configuration settings, thereby playing a critical role in performance forecasting. Unlike fixed-length configuration vectors, segment performance data are naturally represented as variable-length sequences, since the number of segments depends on how much of the workload has been executed. In addition, strong dependencies are often present within these sequences. For example, SQL queries executed early in the workload may warm the cache, which may in turn accelerate subsequent queries that access the same data. Therefore, segment performance data is organized as variable-length sequential data to explicitly represent the interdependencies among queries.

Numerous neural network architectures have been proposed for modeling variable-length sequential data, such as Recurrent Neural Networks (RNNs)[10, 31], Long Short-Term Memory Networks (LSTMs)[14], GRUs[6], and Transformers[40]. Among these, GRUs offer an effective trade-off between modeling capacity and computational efficiency. Specifically, GRUs employ gating mechanisms to mitigate vanishing and exploding gradient problems inherent in traditional RNNs, thereby facilitating the capture of long-term dependencies in sequential data. Compared to LSTMs, GRUs simplify the architecture by using only two gates (i.e., update gates and reset gates), thereby reducing model complexity and computational overhead. Although Transformers excel at parallel sequence modeling for large-scale datasets, their substantial computational and memory requirements make them less practical for resource-constrained scenarios. Furthermore, database knob tuning typically generates relatively small-scale datasets, where Transformers are more likely to overfit.

In the context of segment performance data, GRUs regulate the flow of information across the sequence of performance measurements using two key gating mechanisms: the update gate and the reset gate. The update gate z_t controls how much of the previous hidden state h_{t-1}, which encodes the previous segment performance data, is carried over to the current hidden state h_t. This enables the GRU to balance historical segment performance data with newly acquired segment performance data, capturing both long-term dependencies and real-time changes in system performance as new segment performance data becomes available. It is computed as follows:

$$z_t = \sigma_{act}(W_z[h_{t-1}, s_t] + b_z) \tag{5}$$

where s_t represents the segment performance data at time step t, σ_{act} is the sigmoid activation function, W_z is the weight matrix, and b_z is the bias term for the update gate.

The reset gate r_t controls the extent to which the previous hidden state h_{t-1} is incorporated into the candidate hidden state $\tilde{h}_t$. This candidate state captures interdependencies and sequential patterns within the segment performance data. This mechanism is crucial for selectively discarding irrelevant or outdated performance data from previous segments, while allowing the model to incorporate new, relevant segment performance data as subsequent segments are processed. The reset gate is computed as follows:

$$r_t = \sigma_{act}(W_r[h_{t-1}, s_t] + b_r) \tag{6}$$

where W_r and b_r are the weight matrix and bias term for the reset gate, respectively. This gate helps discard irrelevant information when r_t is close to 0, while retaining past information when r_t approaches 1.

The candidate hidden state $\tilde{h}_t$ is computed based on the current segment's performance input s_t and the previous hidden state h_{t-1}, modulated by the reset gate r_t. This step is essential for modeling how previous segment performance influences the current segment performance, allowing ESTune to capture dynamic dependencies between consecutive segments in a database workload. It is calculated as follows:

$$\tilde{h}_t = \tanh(W_h[r_t \odot h_{t-1}, s_t] + b_h) \tag{7}$$

where W_h and b_h represent the weight matrix and bias term for the candidate hidden state, respectively, and $\odot$ denotes the Hadamard product. The nonlinear activation function tanh enables the model to capture complex, nonlinear dependencies within the segment performance data.

Finally, the output hidden state h_t is calculated by combining the previous segment's hidden state h_{t-1} and the candidate hidden state $\tilde{h}_t$, with their respective contributions regulated by the update gate z_t:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t \tag{8}$$

This process enables ESTune to integrate performance data from multiple segments and make accurate predictions about database performance, leveraging both historical patterns and new segment information.

5.2.3 BNN for Performance Distribution Prediction. In ESTune, we employ BNNs to combine the outputs from the GRU and FNN, where the model weights and biases are treated as probability distributions rather than fixed values. This probabilistic framework enables principled quantification of predictive uncertainty and provides inherent regularization, thus reducing overfitting and improving generalization in performance prediction.

BNNs extend traditional neural networks by assigning probability distributions to the network parameters (e.g., weights and biases). Each parameter is associated with a prior distribution, typically chosen as a Gaussian distribution due to its mathematical convenience and connection to the central limit theorem. This approach enables explicit modeling of uncertainty in performance predictions by accounting for uncertainties in both segment performance data and configuration data encoding. As new segment performance data become available, the model updates the posterior distributions of the parameters using Bayes' theorem:

$$p(\theta|D_{BNN}) = \frac{p(D_{BNN}|\theta)p(\theta)}{p(D_{BNN})} \tag{9}$$

where $p(\theta)$ is the prior, $p(D_{BNN}|\theta)$ is the likelihood, and $p(\theta|D_{BNN})$ is the posterior distribution after observing the training data D_{BNN}. This Bayesian updating process refines the model's uncertainty estimates, providing more robust and reliable performance predictions by continuously adapting to new segment performance data encoding.

5.2.4 Architecture of HBNN. The architecture of HBNN is illustrated in Figure 5. The GRU encodes the segment performance sequence $(s_0, s_1, \ldots, s_l)$, where l denotes the last index of the sequence (i.e., the sequence has $l + 1$ elements). Simultaneously, the FNN extracts features from the n-dimensional configuration vector $(v_1, v_2, \ldots, v_n)$. The outputs of the GRU and FNN are concatenated and fed into the BNN, which models the distribution of database performance by predicting both its mean and variance. Notably, before the start of each iteration, s_0 is set to zero to represent the scenario where predictions are made solely based on configuration parameters before any workload execution.

5.3 Implementation of Evaluator

The evaluator determines whether to stop workload execution based on the mean and variance predicted by the HBNN. At each iteration, it first sorts the historical database performance metrics

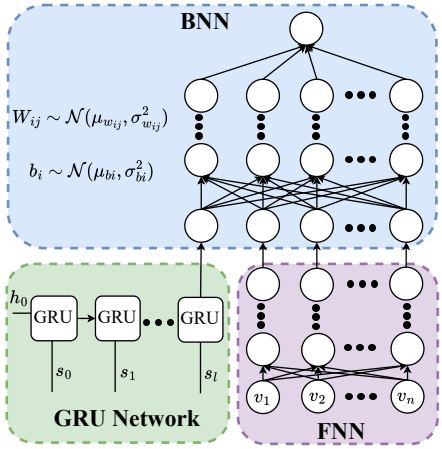
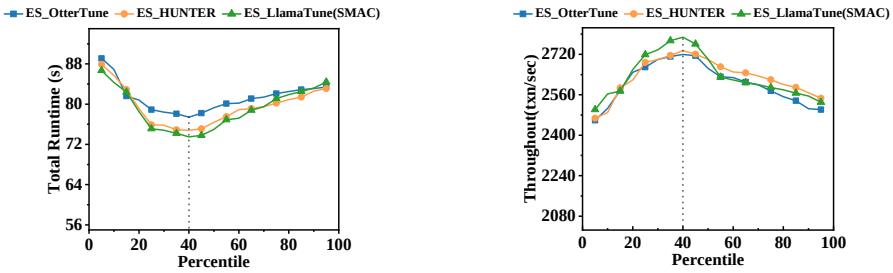


Fig. 5. The HBNN Architecture

from previous iterations within the same tuning task. For metrics where higher values indicate better performance (e.g., throughput), the values are sorted in descending order; for metrics where lower values are preferred (e.g., total runtime), they are sorted in ascending order. The resulting sequence is denoted as $p_1, p_2, \ldots, p_\ell$. A threshold value is then selected from this ordered set, specifically $p_{\lceil \ell \cdot pct \rceil}$, where pct denotes the chosen percentile. The evaluator compares the predicted database performance against this threshold: if the predicted performance is worse than the threshold, the current workload execution is terminated, and the predicted mean is recorded as the performance for the current configuration. The choice of pct is critical. For example, when throughput is the tuning metric, setting pct too low may erroneously filter out promising configurations, thereby reducing overall tuning effectiveness. Conversely, setting pct too high may fail to improve tuning efficiency due to insufficient early stopping of underperforming configurations. To determine an appropriate value, we conducted experiments depicted in Figure 6; the results demonstrate that setting pct to 40% yields optimal tuning performance. Therefore, pct is set to 40% in all subsequent experiments.



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

Fig. 6. Results for different pct values on MySQL

Since the HBNN predicts the performance distribution for each configuration in terms of both mean and variance, the evaluator employs the cumulative distribution function (CDF) to compare the predicted distribution against the performance threshold. Specifically, let μ_g and σ_g^2 denote the

predicted mean and variance of the current configuration's performance. The evaluator computes the probability that the predicted performance is worse than the threshold $p_{\lceil \ell \cdot pct \rceil}$ using the CDF:

$$F(y_i) = \begin{cases} \int_{-\infty}^{p_{\lceil \ell \cdot pct \rceil}} \frac{1}{\sqrt{2\pi}\sigma_g} e^{-\frac{(t-\mu_g)^2}{2\sigma_g^2}} dt, & \text{for maximization} \\ \int_{p_{\lceil \ell \cdot pct \rceil}}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma_g} e^{-\frac{(t-\mu_g)^2}{2\sigma_g^2}} dt, & \text{for minimization} \end{cases} \tag{10}$$

Based on the experimental results in Figure 14 (detailed in Section 6.10) and Table 4 (detailed in Section 6.7), ESTune adopts a 95% CDF t for early stopping decisions. According to this criterion, a configuration is classified as underperforming if the evaluator determines that the probability $F(y_i)$ of its predicted performance being worse than the threshold is at least 95%. This approach prevents unwarranted early termination in cases where predictive uncertainty is substantial, thereby improving the reliability of predicted values used as surrogates for actual database performance.

5.4 The Workflow of ESTune

The workflow of ESTune is illustrated in algorithm 1. At the initial iteration, the HBNN model is constructed, and a performance queue Q is initialized to store database performance metrics in an ordered manner ($line$ 1 ∼ 3). The segment performance queue S is initially populated with 0, enabling ESTune to make performance predictions solely based on configuration parameters ($line$ 9). During workload execution, newly generated segment performance metrics are sequentially appended to S ($line$ 11). The HBNN then uses the current S and configuration knob set V to predict the performance distribution ($line$ 21). If the predicted distribution satisfies the early stopping criterion ($line$ 22), ESTune signals the database to halt further workload execution ($line$ 23), and appends the predicted mean μ_g to Q ($line$ 24). The updated queue Q is subsequently used in the next iteration. Once all segment metrics have been collected ($line$ 12), the HBNN model is updated using all prefix subsequences of S, the knob set V, and the performance p_i obtained from fully executing the workload ($line$ 15 ∼ 17). This update enhances predictive accuracy in future iterations.

Label Handling and Update Protocol. Early-stopped trials are *never* used as training labels for the HBNN. The predictive mean is used only to (i) evaluate the 95% CDF-based stopping test and (ii) supply a surrogate performance value to the recommender for subsequent configuration selection; it is *never* fed back as a ground-truth label. The HBNN parameters are updated *exclusively* using data from fully executed runs. This protocol prevents prediction leakage into training and mitigates exploration bias.

Sampling Policy and Imbalance. ESTune operates downstream of the sampling policy implemented by the underlying tuner (e.g., OtterTune, HUNTER) and therefore does not control which configurations are sampled. To mitigate the impact of imbalance on prediction accuracy, ESTune continually updates its estimate of overall performance whenever new segment-level performance (i.e., s_i in Figure 5 of Section 5.2) is observed, thereby improving estimation accuracy by leveraging partial-run data. Furthermore, ESTune employs few-shot meta-learning to improve generalization across diverse tuning scenarios. Empirically, Table 4 (detailed in Section 6.7) shows that a 95% CDF threshold keeps MaxAPE at ≤ 15%, and Figure 14 (detailed in Section 6.10) demonstrates that MaxAPE ≤ 15% does not materially affect the final tuning outcome. Therefore, ESTune adopts 95% as the CDF threshold, improving tuning efficiency without sacrificing effectiveness.

5.5 MAML-Enhanced Few-Shot Learning

Given the practical requirement for database knob tuning to yield satisfactory results within a few hours, the number of tuning iterations is typically limited. This limitation, in turn, constrains the amount of training data available for the HBNN. Neural networks often underperform in such

Algorithm 1: The workflow of HBNN and evaluators during each iteration.

Input: i: the iteration number; Q: set of historical performance metrics; num_{seg}: the number of database segment performances.

```
if i = 1 then
    Construct_HBNN()
    Q ← Sort_queue()
V ← the set of knob values for the i-th iteration
S ← Queue()
while True do
    L ← Length(S)
    if L = 0 then
        S.add(0)
    else
        S.add(get_segment_performance())
    if num_seg = L then
        p_i ← get_performance()
        tmp_s ← Queue()
        for element ∈ S do
            tmp_s.add(element)
            update_HBNN(tmp_s, V, p_i)
        Q.add(p_i)
        break
    else
        (μ_g, σ_g^2) = HBNN_predict(S, V)
        if eval(μ_g, σ_g^2, Q.index(⌈L · pct⌉) > 95% then
            send_stop_signal()
            Q.add(μ_g)
            break
```

data-limited scenarios, as they typically require abundant training data to optimize parameters, avoid overfitting, and achieve strong generalization. To address this limitation, we incorporate MAML into HBNN, which enables the model to rapidly adapt to new tasks with limited data by learning a well-generalized initialization parameters from a set of related historical tuning tasks.

In the context of database knob tuning, each historical tuning task corresponds to a distinct dataset, which is partitioned into a support set and a query set. The support set is used for inner-loop adaptation, while the query set evaluates the model's post-adaptation performance. Prior to meta-training, the global initialization parameters of the HBNN model, denoted by θ, are shared across all historical tasks, allowing the model to learn from a diverse range of past tuning experiences. For each historical task, the model first performs a forward pass on the support set using the current global parameters θ to compute the task-specific loss. This loss is then used to adapt the parameters via gradient descent, yielding the adapted parameters for the i-th task:

$$\theta_i' = \theta - \alpha \nabla \theta L_{T_i}(\theta) \tag{11}$$

where α is the learning rate and L_{T_i} denotes the loss function for the i-th historical task T_i.

The query set for each task is then used to evaluate the adapted parameters θ_i', and the resulting loss is used to compute the meta-gradient with respect to the global parameters θ. The meta-update step is as follows:

$$\theta \leftarrow \theta - \beta \sum T_i \nabla_\theta L_{T_i}(\theta_i') \tag{12}$$

where β denotes the meta-learning rate. This iterative cycle of task-specific adaptation and global meta-update progressively optimizes the initialization parameters, enabling rapid adaptation to new tuning tasks.

To make the initialization of optimization parameters better suited to the characteristics of the current tuning task, MAML utilizes historical data from the same workload category (e.g., time-based or volume-based) when learning the initialization parameters of the HBNN. This targeted selection strategy incorporates domain-specific knowledge into the initialization process, thereby accelerating convergence and improving prediction accuracy.

6 Experimental Evaluation

In this section, we empirically evaluate the design of ESTune using two widely-used benchmarks.

6.1 Experimental Setup

6.1.1 Experimental Settings. All experiments were conducted on a host equipped with an 8-core 2.10 GHz CPU, 16 GB of RAM, and a 150 GB HDD. MySQL 8.0 and PostgreSQL 12.12 were used as the target DBMSs for knob tuning. We use a preselected subset of 100 knobs for MySQL and 90 for PostgreSQL. The HBNN was composed of a single-layer GRU network, a three-layer FNN, and a four-layer BNN. We employed two representative workloads from OLTPBench [8]: TPC-C and TPC-H. The TPC-C workload, simulating a time-based operational environment, was set up with 320 warehouses, a data size of approximately 20 GB, and 500 terminals. The TPC-H workload, representing a volume-based scenario, was configured with a scale factor of 25, corresponding to a data size of approximately 25 GB. The total tuning duration was set to 240 minutes, with each iteration of the TPC-C workload lasting 200 seconds.

Notably, the aforementioned subsets of 100 knobs for MySQL and 90 knobs for PostgreSQL were derived by selecting knobs that met at least one of the following two criteria: (i) **Performance impact**, knobs known to significantly affect database performance (e.g., innodb_buffer_pool_size); and (ii) **Configuration representativeness**, knobs that cover various modules of the database system (such as storage, lock management, and caching), so as to enable the database to exhibit performance reflective of real-world scenarios. These knobs are frequently studied in database tuning research, including prior work such as OtterTune and CDBTune, as well as industry tuning guides. This selection yields a representative and effective knob set for evaluating the performance of the tuning methods.

6.1.2 Baseline Methods. Database tuning methods are typically categorized into six classes: search-based, GP-based, SMAC-based, reinforcement learning (RL)-based, hybrid methods, and large language model (LLM)-based methods, as detailed in Section 7. To comprehensively evaluate the performance improvements introduced by ESTune, we select one or two methods from each category, namely BestConfig [53], OBTune [43], OtterTune [1], LlamaTune (SMAC) [17], GPTuner [23], HUNTER [4], OpAdviser [48], and E2ETune [15].

Among these, the first seven methods employ iterative tuning frameworks, allowing for direct integration with ESTune, which results in their enhanced versions: ES_BestConfig, ES_OBTune, ES_OtterTune, ES_LlamaTune (SMAC), ES_GPTuner, ES_HUNTER, and ES_OpAdviser. In contrast,

E2ETune recommends an initial configuration for new tuning environments, and additional performance improvements are achieved through subsequent iterative tuning methods. To evaluate its effectiveness, we use E2ETune to identify an initial configuration, then apply BestConfig, OtterTune, and OpAdviser to tune this configuration. The methods utilizing the initial configuration recommended by E2ETune are denoted as BestConfig+, OtterTune+, and OpAdviser+. The versions enhanced with ESTune for improved efficiency are referred to as ES_BestConfig+, ES_OtterTune+, and ES_OpAdviser+, respectively.

It is worth noting that, among these methods, BestConfig and BestConfig+ focus on identifying promising regions within the configuration space as a means to optimize the knob configuration, and these promising regions are identified using Latin Hypercube Sampling (LHS). Accordingly, ESTune aims to accelerate the discovery of such regions during the LHS process. Its early stopping strategy is determined solely based on the performance of the current set of LHS-sampled configurations, without considering the performance of previously sampled configurations within the current task.

6.1.3 Data Repository. The HBNN parameters were initialized using MAML with historical tuning task data. Specifically, for the volume-based tuning category, data from ten tuning tasks were collected for both the TPC-DS and SSB workloads. Similarly, for the time-based tuning category, historical data from ten tuning tasks were gathered for both the Wikipedia and YCSB workloads.

6.2 Practical Overhead of ESTune Integration

Each iteration of tuning methods typically consists of two primary phases: workload execution and configuration recommendation. ESTune introduces two additional steps: configuration performance prediction, and HBNN updating. Configuration performance prediction is performed concurrently with workload execution, where the HBNN model is used to evaluate the performance of each configuration knob. The overhead of this process is negligible, with each prediction requiring no more than 0.01 seconds.

HBNN updating is part of the configuration recommendation phase and occurs only after the workload has been fully executed. This update is not performed in every iteration but is triggered after the entire workload execution. To quantify the computational overhead of model updates, we compared the configuration recommendation time between the original tuning method and its ESTune-enhanced counterpart with model updates enabled, as shown in Table 1. The results indicate that, although HBNN updating slightly increase recommendation time, the additional overhead is negligible for long-running workloads. The configuration recommendation times for ES_BestConfig+, ES_OtterTune+, and ES_OpAdviser+ are similar to those for ES_BestConfig, ES_OtterTune, and ES_OpAdviser, respectively. Similarly, the configuration recommendation times for BestConfig+, OtterTune+, and OpAdviser+ are comparable to those for BestConfig, OtterTune, and OpAdviser, respectively. This is due to the fact that their iterative processes follow the same underlying principles.

6.3 Efficiency Comparison

We evaluated both the original and ESTune-enhanced versions on MySQL under the TPC-C workload, with results presented in Figure 7. Our experiments consistently demonstrate that ESTune improves tuning efficiency. For example, ES_BestConfig achieved the final result of BestConfig in just 160 minutes, a 31% reduction in tuning time from the original 232-minute result. Tuning methods such as BestConfig+, OtterTune+, and OPTuner+, which utilize initial configurations recommended by E2ETune, achieved convergence within the 240-minute tuning window. Notably, ESTune further accelerates these methods' convergence process, leading to the faster attainment of high-performing configurations.

Table 1. Time spent in recommended configuration stages.

Methods	Original Time	Enhanced Time
BestConfig	< 0.01s	2.04s
OtterTune	7.01s	7.61s
HUNTER	0.02s	2.05s
LlamaTune (SMAC)	1.67s	3.21s
OpAdviser	6.01s	6.78s
OBTune	7.22s	7.85s
GPTuner	1.68s	3.22s
BestConfig+	< 0.01s	2.03s
OtterTune+	7.00s	7.62s
OpAdviser+	6.02s	6.77s

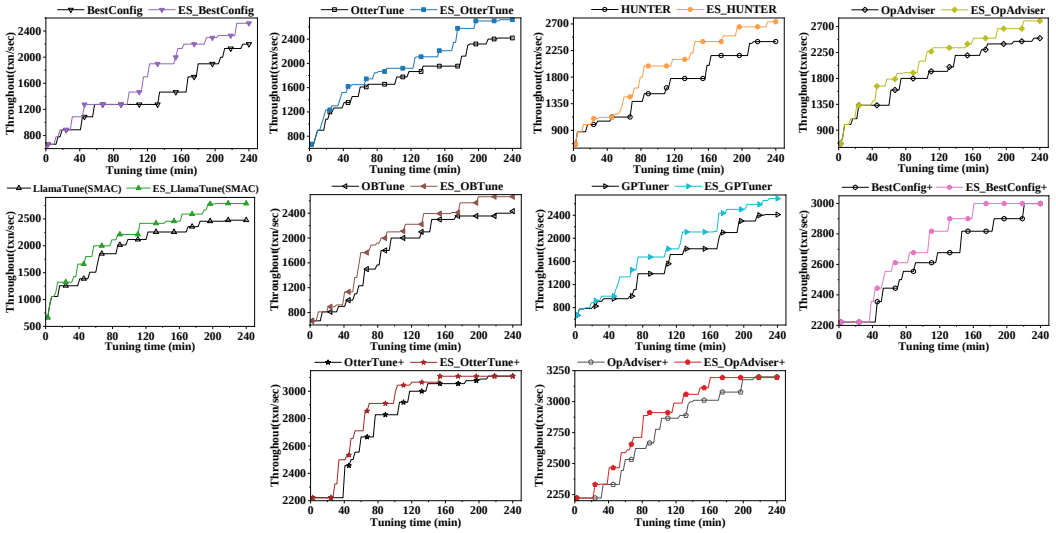


Fig. 7. Tuning results of MySQL on TPC-C workload

The primary advantage of ESTune lies in its ability to accelerate tuning by enabling early termination of underperforming configurations, thereby allowing more iterations within a fixed time budget. This, in turn, increases the likelihood of discovering superior configurations. This analysis is empirically confirmed by counting the number of iterations completed within 240 minutes for each tuning method, as shown in Table 2. The statistical results indicate that, within 240 minutes, the ESTune-enhanced versions performed over 40 more iterations than their original counterparts.

We conducted further experiments on the MySQL database using the TPC-H benchmark workload, as detailed in Figure 8. The experimental results continue to demonstrate that ESTune significantly improves the tuning efficiency of the original methods. The similarity between ES_OtterTune+ and OtterTune+ in Figure 8 is attributable to the fact that both the tuning has converged and that early stopping does not materially affect final outcome of OtterTune+. Minor discrepancies of ES_OtterTune+ relative to OtterTune+ are expected mainly due to performance fluctuation between

Table 2. The number of iterations within 240 minutes

Methods	Original Iterations	Enhanced Iterations
BestConfig	71	122
OtterTune	69	115
HUNTER	71	119
LlamaTune (SMAC)	71	121
OpAdviser	69	116
OBTune	69	113
GPTuner	71	117
BestConfig+	71	122
OtterTune+	69	116
OpAdviser+	69	115

different runs under the same configuration; these differences are small, fall within normal statistical variation, and are negligible. The same holds for other workloads: when both methods converge, any residual gap is likewise insignificant.

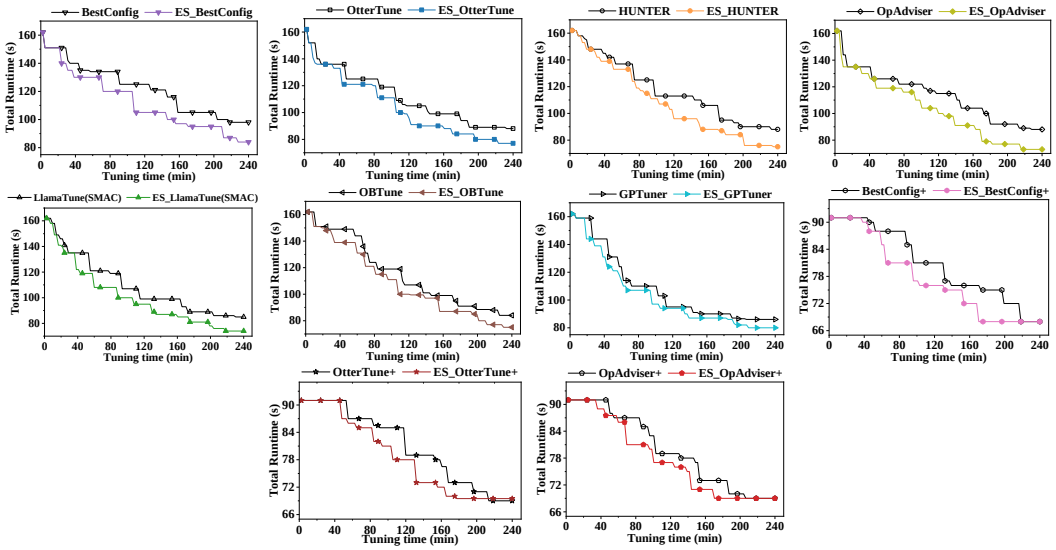


Fig. 8. Tuning results of MySQL on TPC-H workload

6.4 Evaluation on Generalization

To assess the generalizability of ESTune, we evaluated both the original and ESTune-enhanced tuning methods on PostgreSQL under the TPC-C workload (shown in Figure 9). The results show that the ESTune-enhanced methods consistently outperform the original methods in terms of tuning efficiency. OpAdviser+, BestConfig+, and their corresponding ESTune-enhanced variants all achieved convergence within the 240-minute evaluation window. In contrast, OtterTune+ and ES_OtterTune+ did not converge within the allotted time. This limitation is primarily due to their susceptibility to getting trapped in local optima during the iterative tuning process, which impedes

both convergence speed and final performance. It is noteworthy that, for the TPC-C workload, the original methods exhibited nearly identical numbers of iterations on both MySQL and PostgreSQL. This was mainly due to the fixed workload execution time per iteration.

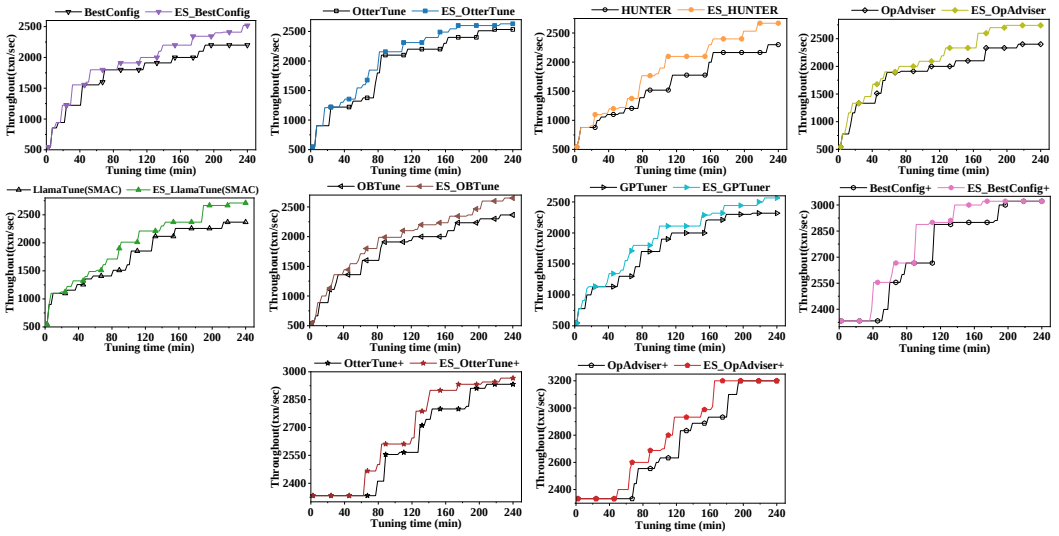


Fig. 9. Tuning results of PostgreSQL on TPC-C workload

By comparing Figure 7 and Figure 9, we observe that early stopping yields different magnitudes of improvement on MySQL and PostgreSQL. Although the gains are generally larger on MySQL in these experiments, the ES_OpAdviser case is a notable counterexample. These differences arise from database-specific characteristics that cause the same tuning algorithm to explore distinct configuration sequences on the two systems. Specifically, the underperforming configurations encountered in the MySQL sequence exhibited poor-performance patterns that provided a clearer and more predictable signal to the HBNN model. This allowed ESTune to quickly form high-confidence predictions and thus trigger the early-stopping criterion more frequently. Conversely, the poor-performance patterns on PostgreSQL exhibited higher variance, which caused ESTune's model to behave more conservatively. As a result, the corresponding predictive intervals were wider and satisfied the early-stopping criterion less often, leading to smaller observed efficiency gains.

6.5 Analysis of Iteration Progress Over Time

To elucidate how early stopping affects tuning progress over time, we conducted an experiment that analyzes the cumulative number of completed iterations as a function of elapsed tuning time, as shown in Figure 10. The results indicate that gains in tuning efficiency (i.e., faster growth in cumulative iterations relative to the baseline without early stopping) are *non-linear*, emerging predominantly in the middle-to-late stages of the tuning process rather than uniformly over time. For instance, within the first 120 minutes (i.e., 50% of the total time budget), ES_BestConfig completes only 37% of its total iterations. By contrast, the BestConfig baseline completes exactly 50% of its iterations within the first 120 minutes.

This phenomenon is intrinsically linked to the confidence-bound mechanism of the HBNN model. In the initial tuning phase, the scarcity of historical data leads to high predictive variance in the

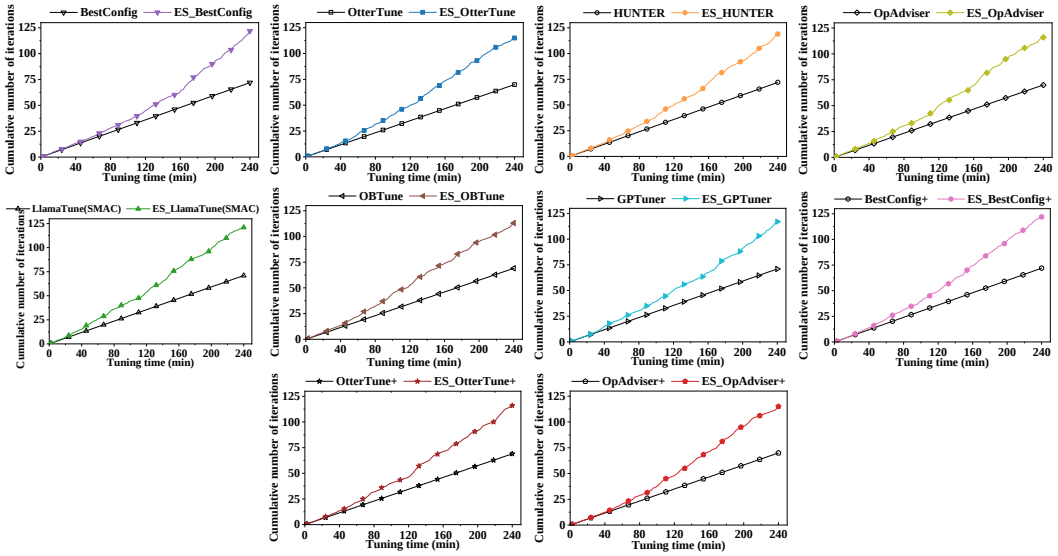


Fig. 10. Relationship between cumulative iterations and elapsed time for MySQL tuning under TPC-C

model's outputs. This, in turn, results in wider confidence intervals, making it difficult for the early-stopping condition to be satisfied. Conversely, as the tuning process progresses, the HBNN accumulates sufficient data, leading to a substantial reduction in predictive variance. Consequently, the confidence intervals narrow, enabling the early-stopping strategy to identify and terminate underperforming configurations more frequently.

6.6 Evaluation in Cold Start Scenarios

The cold start problem, characterized by the absence of historical data, is commonly encountered in database knob tuning. To assess the effectiveness of ESTune under such conditions, we conducted comparative experiments by comparing the ESTune-enhanced methods to the original tuning methods.

In the cold-start scenario, the tuning procedures of LlamaTune (SMAC), BestConfig, OBTune, and GPTuner do not rely on historical tuning task data and are thus unaffected by cold start. Consequently, their experimental results are omitted. In contrast, OpAdviser requires historical data to select an appropriate optimizer, making it inapplicable under cold-start conditions, and therefore it is excluded from the experiments. Additionally, E2ETune relies on relevant historical data for training, so its results are also excluded in the cold-start setting. Therefore, in the absence of historical data, we evaluated the remaining tuning methods on MySQL using the TPC-C benchmark, as illustrated in Figure 11. The experimental results demonstrate that ESTune significantly improves both the final tuning results and tuning efficiency, even without historical data. Specifically, ES_HUNTER achieved the optimal result that HUNTER reached in 158 minutes within only 112 minutes, representing a 29% improvement in tuning efficiency.

Even in the absence of historical data, ESTune can still effectively terminate underperforming configurations early. This capability enables more iterations within a fixed time budget, thereby enhancing the tuning efficiency of the original methods. To validate this, we measured the number of iterations completed by each method without historical data, as shown in Table 3. The results indicate that within 240 minutes, the enhanced methods completed over 24 additional iterations

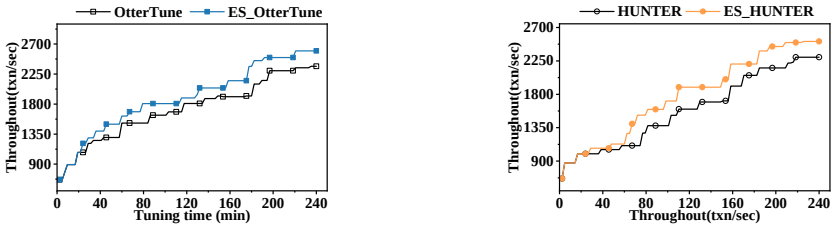


Fig. 11. Tuning results of MySQL on TPC-C workload without historical data

compared to their original counterparts. It is important to note that although the computational overhead for OtterTune and HUNTER to select the next sample is reduced under cold-start compared to the historical data scenario, this saving is negligible compared to the dominant, fixed cost of workload execution. Consequently, their total number of completed iterations remains identical to that of the historical data scenario.

Table 3. The number of iterations without historical data

Methods	Original Iterations	Enhanced Iterations
OBTune	69	95
HUNTER	71	96

6.7 Analysis of the CDF threshold

To determine an appropriate CDF threshold, we evaluated several candidate thresholds across all ESTune-enhanced tuning methods. For each threshold, we compared the predicted performance with the actual performance for all early-stopped configurations and computed the resulting Maximum Absolute Percentage Error (MaxAPE) and Mean Absolute Percentage Error (MAPE). Table 4 summarizes the relationship between the CDF threshold and the corresponding error metrics.

Table 4. Comparison of MaxAPE and MAPE across different methods and CDF threshold.

Methods	CDF 97% (Max/MAPE)	CDF 95% (Max/MAPE)	CDF 93% (Max/MAPE)	CDF 91% (Max/MAPE)
ES_BestConfig	11.0% / 3.3%	14.9% / 5.2%	18.1% / 8.5%	23.9% / 11.7%
ES_OtterTune	11.1% / 3.5%	14.5% / 5.6%	19.0% / 8.8%	23.1% / 11.6%
ES_HUNTER	11.6% / 3.6%	15.1% / 5.7%	18.7% / 8.8%	23.3% / 11.8%
ES_OpAdviser	11.2% / 3.3%	15.2% / 5.9%	18.3% / 8.9%	23.2% / 11.4%
ES_LlamaTune	11.5% / 3.4%	14.4% / 5.3%	18.8% / 8.2%	23.5% / 11.5%
ES_OBTune	11.3% / 3.2%	14.7% / 5.1%	18.6% / 8.1%	23.0% / 11.7%
ES_GPTuner	11.1% / 3.4%	15.7% / 5.6%	18.5% / 8.5%	23.4% / 11.8%
ES_BestConfig+	11.1% / 3.5%	14.8% / 5.4%	18.8% / 8.4%	23.8% / 12.1%
ES_OtterTune+	11.2% / 3.5%	15.1% / 5.7%	18.7% / 8.9%	23.3% / 11.6%
ES_OpAdviser+	11.4% / 3.6%	14.8% / 5.9%	18.5% / 8.8%	23.4% / 11.9%

The results demonstrate that smaller CDF thresholds tend to yield larger MaxAPE and MAPE. Crucially, a 95% CDF threshold keeps MaxAPE within 15%. As shown in Figure 14 (detailed in Section 6.10), MaxAPE $\leq$ 15% does not materially affect the final tuning outcome, whereas larger errors can severely compromise the effectiveness of the tuning method. Therefore, we adopt a 95% CDF threshold, which strikes an effective balance between tuning efficiency and predictive robustness.

6.8 Ablation Study on MAML

To enhance the few-shot adaptation capability of HBNN, ESTune integrates MAML. To assess the contribution of MAML, we conducted an ablation study by removing the MAML component from the ESTune-enhanced methods. The resulting variants, denoted as ES_LlamaTune(SMAC)_M, ES_BestConfig_M, ES_BestConfig+_M, ES_OBTune_M, ES_OtterTune_M, ES_OtterTune+_M, ES_GPTuner_M, ES_HUNTER_M, ES_OpAdviser_M, and ES_OpAdviser_M, represent the corresponding ESTune-enhanced methods without MAML.

The tuning results of the enhanced and reduced-MAML versions correspond to those presented in Figure 12. Experimental results consistently show that the enhanced versions outperform their reduced-MAML counterparts in tuning efficiency. This performance gain is primarily attributed to the MAML module's ability to optimize initial parameters effectively, thereby accelerating convergence to optimal configurations for new tuning tasks.

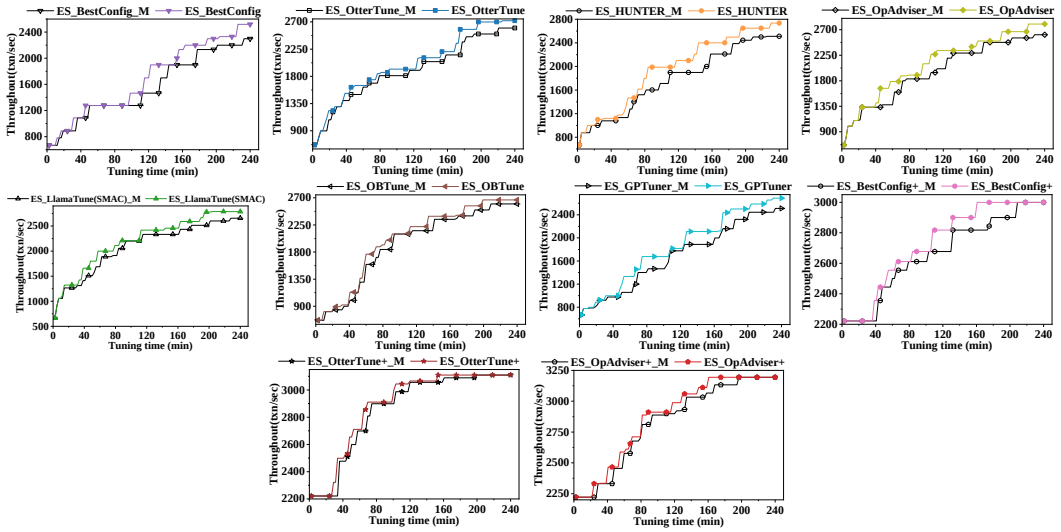


Fig. 12. Ablation study on MAML

6.9 Ablation Study on GRU

In the context of database performance prediction, HBNN leverages both configuration data and segment performance data, with the latter providing crucial local information about workload execution. To evaluate the contribution of segment performance data, we conducted an ablation study by removing the GRU. The resulting variants, denoted as ES_BestConfig_G, ES_BestConfig+_G, ES_LlamaTune(SMAC)_G, ES_OpAdviser_G, ES_OpAdviser+_G, ES_OtterTune_G, ES_OtterTune+_G, ES_GPTuner_G, ES_OBTune-G, and ES_HUNTER_G, represent the ESTune-enhanced methods with the GRU module excluded.

We conducted configuration knob tuning experiments on the MySQL database using the TPC-C benchmark workload to compare the tuning performance of ESTune-enhanced versions with their reduced-GRU counterparts, as shown in Figure 13. The results indicate that the enhanced versions achieved an improvement in tuning efficiency. This improvement can be attributed to the incorporation of segment performance data, which substantially enhances the prediction accuracy of HBNN and increases the opportunities for early stopping. To validate this, we recorded the total number of iterations completed within 240 minutes, as shown in Table 5. The enhanced versions completed over 13 additional iterations compared to the reduced-GRU variants, demonstrating the

importance of leveraging segment performance data in improving tuning efficiency for database knob tuning.

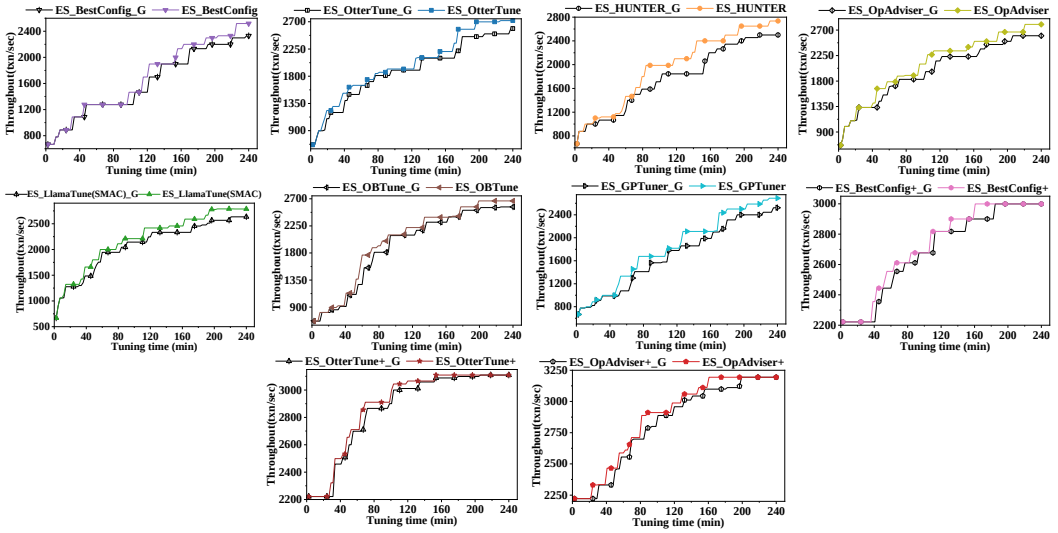


Fig. 13. Ablation study on GRU

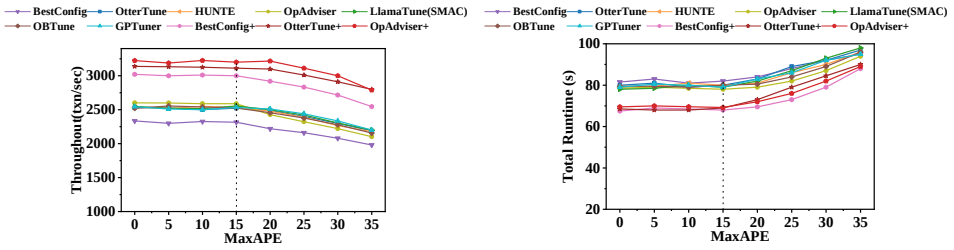
Table 5. The number of iterations without GRU

Methods	Without GRU	With GRU
ES_BestConfig	98	122
ES_OtterTune	101	115
ES_HUNTER	100	119
ES_LlamaTune (SMAC)	99	121
ES_OpAdviser	96	116
OBTune	98	113
GPTuner	100	117
BestConfig+	97	122
OtterTune+	99	116
OpAdviser+	101	115

6.10 Ablation Study on Evaluation Inaccuracy

To assess how inaccuracies in performance evaluation affect the final tuning outcome, we injects controlled noise into the actual performance A_i of configurations flagged as underperforming by the evaluator. Specifically, for each such configuration, we set $\tilde{A}_i = A_i \cdot u$, where u is drawn at random from the interval $[1 - \text{MaxAPE}, 1 + \text{MaxAPE}]$. We vary MaxAPE over {0%, 5%, 10%, 15%, 20%, 25%, 30%, 35%} and measure the final tuning performance after 100 iterations. As shown in Figure 14, the final tuning outcome remains stable when MaxAPE $\leq$ 15%. Beyond this range, the final tuning performance degrades significantly, which is consistent with the expectation that large perturbations can severely compromise the effectiveness of the tuning method. This ablation study shows that when the injected evaluation error for underperforming configurations is limited to $\leq$ 15%, the tuning method's effectiveness is preserved and the results remain uncompromised.

Table 4 (detailed in Section 6.7) shows that setting the CDF threshold to 95% keeps the approximation error within the 15% tolerance. Consequently, we adopt 95% as the CDF threshold.



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

Fig. 14. Results for different *MaxAPE* on MySQL

6.11 Discussion on Tuning Settings

To systematically compare these settings, we use ES_OtterTune as a baseline and perform controlled comparisons. With historical data (i.e., in the warm-start setting), ES_OtterTune in Figure 11 achieves higher final performance than cold-start ES_OtterTune in Figure 7, primarily because MAML leverages historical runs to meta-learn better initializations, thereby accelerating adaptation and convergence on new tuning tasks. Figure 12 further conducts a MAML ablation by comparing the full ESTune-enhanced ES_OtterTune with its MAML-ablated variant ES_OtterTune_M, demonstrating that removing MAML consistently slows convergence and reduces final performance. Finally, Figure 13 evaluates the effect of the GRU component on ES_OtterTune and shows that it yields higher tuning efficiency than its GRU-free variant ES_OtterTune_G. This improvement occurs because modeling segment-level performance sequences with a GRU improves the predictive accuracy for early-stopped configurations and increases opportunities for early stopping. Overall, both MAML and GRU are critical for improving ESTune's tuning efficiency.

7 Related Works

Existing tuning approaches fall into six classes: search-based methods, GP-based methods [32], SMAC-based methods [16], RL-based methods, hybrid methods, and LLM-based methods [27, 30].

Search-based Methods. BestConfig [53] tunes configuration knobs by iteratively narrowing the parameter space. It first applies Latin Hypercube Sampling (LHS) [28] over the global space, evaluates all candidates, and selects the best-performing configuration. The configuration's neighborhood serves as the local search region for LHS resampling. If an improved configuration is found, the local search repeats; otherwise, the search restarts from the global space. Owing to the stochastic nature of sampling, BestConfig's tuning outcomes can vary substantially.

GP-based Methods. iTuned [36] pioneered the application of GPs for parameter tuning, initializing a preliminary model using LHS-sampled data. OtterTune [1] initializes its GP with historical data, uses Lasso [37] for knob selection, and adopts an incremental strategy [7]. ResTune [46] focuses on minimizing resource usage while maintaining performance. CGPTuner [5] considers contextual factors (e.g., operating system and Java Virtual Machine) and uses Contextual GP Bandit Optimization [21] to tune knobs of the entire IT stack. ONLINETUNE [47] introduces an online framework [9] for adaptive tuning in dynamic cloud environments. OBTune [43] addresses multi-objective (e.g., functionality-level) tuning using multi-task GPs [3, 13, 20]. While GPs are effective in balancing exploration and exploitation, they face scalability issues when applied to high-dimensional knob-tuning tasks.

SMAC-based Methods. As demonstrated by experimental results in [45], the SMAC algorithm can outperform GPs in high-dimensional configuration spaces. LlamaTune (SMAC) [17] leverages SMAC and improves sample efficiency via HeSBO [29] for dimensionality reduction, biased sampling, and bucketing. GPTuner [23] leverages SMAC within a coarse-to-fine framework to navigate the trade-off between search efficiency and solution quality. However, the tree-based model in SMAC may not provide consistently high predictive accuracy.

RL-based Methods. CDBTune [44] introduced an end-to-end RL approach using DDPG [38], mapping internal database states to knob actions based on performance rewards. QTune [25] also leverages RL but incorporates query statements to support multi-level tuning. A common challenge is the cold start problem, as pretrained models struggle to generalize. HUNTER [4] addresses this by hybridizing GA with RL to improve adaptation. It is worth noting that RL-based methods typically require substantially more training data to achieve satisfactory tuning results [49].

Hybrid Methods. Recognizing that no single optimizer is universally optimal [2, 41, 45], OpAdviser [48] leverages historical data to dynamically select optimizers and employs transfer learning [54] to reduce the search space. While this improves adaptability, it may also introduce additional system complexity.

LLM-based Methods. LLMs are used to extract tuning hints (e.g., DB-BERT [39]), facilitate knob selection (e.g., GPTuner [23]), or recommend initial configurations (e.g., E2ETune [15]). However, these approaches often still require subsequent iterative tuning. They may also generate irrelevant suggestions and pose privacy or significant deployment overheads [27].

8 Conclusion

To improve the efficiency of iterative database tuning methods, we propose ESTune. Specifically, ESTune leverages HBNN to predict the performance distribution for each configuration and dynamically determines whether to stop the workload execution based on these predictions. If the early stopping criterion is satisfied, the workload execution is terminated, and the predicted mean is used as the estimated performance; otherwise, execution continues. To enhance adaptability in data-limited scenarios, ESTune employs MAML to initialize the parameters of the HBNN. Comprehensive experiments consistently demonstrate that ESTune significantly enhances iterative tuning efficiency.

Acknowledgments

We thank the anonymous reviewers for their insightful comments and feedback. This work is supported by grants from the National Natural Science Foundation of China No.92270202, No.U22B2020 and OceanBase. Peng Cai is the corresponding author.

References

[1] Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. 2017. Automatic Database Management System Tuning Through Large-scale Machine Learning. In *SIGMOD*. 1009–1024.

[2] 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.

[3] Edwin V. Bonilla, Kian Ming Adam Chai, and Christopher K. I. Williams. 2007. Multi-task Gaussian Process Prediction. In *NeurIPS*. 153–160.

[4] 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*. 646–659.

[5] Stefano Cereda, Stefano Valladares, Paolo Cremonesi, and Stefano Doni. 2021. CGPTuner: a Contextual Gaussian Process Bandit Approach for the Automatic Tuning of IT Configurations Under Varying Workload Conditions. *Proc. VLDB Endow.* 14, 8 (2021), 1401–1413.

[6] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *CoRR* abs/1412.3555 (2014).

[7] Emilie Danna and Laurent Perron. 2003. Structured vs. Unstructured Large Neighborhood Search: A Case Study on Job-Shop Scheduling Problems with Earliness and Tardiness Costs. In *CP*. 817–821.

[8] Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudré-Mauroux. 2013. OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases. *Proc. VLDB Endow.* 7, 4 (2013), 277–288.

[9] Marcello Fiducioso, Sebastian Curi, Benedikt Schumacher, Markus Gwerder, and Andreas Krause. 2019. Safe Contextual Bayesian Optimization for Sustainable Room Temperature PID Control Tuning. In *IJCAI*. 5850–5856.

[10] Yarin Gal and Zoubin Ghahramani. 2016. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. In *Neural*. 1019–1027.

[11] Jia-Ke Ge, Yanfeng Chai, and Yunpeng Chai. 2021. WATuning: A Workload-Aware Tuning System with Attention-Based Deep Reinforcement Learning. *J. Comput. Sci. Technol.* 36, 4 (2021), 741–761.

[12] Ethan Goan and Clinton Fookes. 2020. Bayesian Neural Networks: An Introduction and Survey. *CoRR* abs/2006.12024 (2020).

[13] Daniel Hernández-Lobato, Jose Hernandez-Lobato, Amar Shah, and Ryan Adams. 2016. Predictive entropy search for multi-objective bayesian optimization. In *ICML*. 1492–1501.

[14] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (1997), 1735–1780.

[15] Xinmei Huang, Haoyang Li, Jing Zhang, Xinxin Zhao, Zhiming Yao, Yiyan Li, Tieying Zhang, Jianjun Chen, Hong Chen, and Cuiping Li. 2024. E2ETune: End-to-End Knob Tuning via Fine-tuned Generative Language Model. *CoRR* abs/2404.11581 (2024).

[16] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential Model-Based Optimization for General Algorithm Configuration. In *LION*. 507–523.

[17] 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

[18] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. 2021. A review on genetic algorithm: past, present, and future. *Multim. Tools Appl.* 80, 5 (2021), 8091–8126.

[19] Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. 2017. Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets. In *AISTATS*, Vol. 54. 528–536.

[20] Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. 2017. Fast bayesian optimization of machine learning hyperparameters on large datasets. In *AISTATS*. 528–536.

[21] Andreas Krause and Cheng Soon Ong. 2011. Contextual Gaussian Process Bandit Optimization. In *NeurIPS*. 2447–2455.

[22] Tammo Krueger, Danny Panknin, and Mikio L. Braun. 2015. Fast cross-validation via sequential testing. *Journal of Machine Learning Research* 16 (2015), 1103–1155.

[23] 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.

[24] Guoliang Li, Xuanhe Zhou, and Lei Cao. 2021. Machine Learning for Databases. *Proc. VLDB Endow.* 14, 12 (2021), 3190–3193.

[25] 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.

[26] Lisha Li, Kevin G. Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: Bandit-Based Configuration Evaluation for Hyperparameter Optimization. In *ICLR*.

[27] Zheheng Luo, Qianqian Xie, and Sophia Ananiadou. 2023. ChatGPT as a Factual Inconsistency Evaluator for Abstractive Text Summarization. *CoRR* abs/2303.15621 (2023).

[28] Michael D. McKay, Richard J. Beckman, and William J. Conover. 2000. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output From a Computer Code. *Technometrics* 42, 1 (2000), 55–61.

[29] Amin Nayebi, Alexander Munteanu, and Matthias Poloczek. 2019. A Framework for Bayesian Optimization in Embedded Subspaces. In *ICML*. 4752–4761.

[30] OpenAI. 2023. GPT-4 Technical Report. *CoRR* abs/2303.08774 (2023).

[31] Hojjat Salehinejad, Julianne Baarbe, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. 2018. Recent Advances in Recurrent Neural Networks. *CoRR* abs/1801.01078 (2018).

[32] Matthias W. Seeger. 2004. Gaussian Processes For Machine Learning. *International Journal of Neural Systems* 14, 2 (2004), 69–106.

[33] David G. Sullivan, Margo I. Seltzer, and Avi Pfeffer. 2004. Using probabilistic reasoning to automate software tuning. In *SIGMETRICS*. ACM, 404–405.

[34] Shengyang Sun, Changyou Chen, and Lawrence Carin. 2017. Learning Structured Weight Uncertainty in Bayesian Neural Networks. In *AISTATS*, Vol. 54. PMLR, 1283–1292.
[35] 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.
[36] Vamsidhar Thummala and Shivnath Babu. 2010. iTuned: a tool for configuring and visualizing database parameters. In *SIGMOD*. 1231–1234.
[37] Robert Tibshirani. 2018. Regression Shrinkage and Selection Via the Lasso. *Journal of the Royal Statistical Society: Series B (Methodological)* 58, 1 (2018), 267–288.
[38] Yegor Tkachenko. 2015. Autonomous CRM Control via CLV Approximation with Deep Reinforcement Learning in Discrete and Continuous Action Space. *CoRR* abs/1504.01840 (2015).
[39] Immanuel Trummer. 2022. DB-BERT: A Database Tuning Tool that "Reads the Manual". In *SIGMOD*, Zachary G. Ives, Angela Bonifati, and Amr El Abbadi (Eds.). 190–203.
[40] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *CoRR* abs/1706.03762 (2017).
[41] David H. Wolpert and William G. Macready. 1997. No free lunch theorems for optimization. *IEEE Trans. Evol. Comput.* 1, 1 (1997), 67–82.
[42] Tiannuo Yang, Wen Hu, Wangqi Peng, Yusen Li, Jianguo Li, Gang Wang, and Xiaoguang Liu. 2024. VDTuner: Automated Performance Tuning for Vector Data Management Systems. In *ICDE*. 4357–4369.
[43] Zhongwei Yue, Shujian Peng, Peng Cai, Xuan Zhou, Huiqi Hu, Rong Zhang, Quanqing Xu, and Chuanhui Yang. 2024. Functionality-Aware Database Tuning via Multi-Task Learning. In *ICDE*. 83–95.
[44] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, Minwei Ran, and Zekang Li. 2019. An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning. In *SIGMOD*. 415–432.
[45] 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.
[46] 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*. 2102–2114.
[47] 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*. 631–645.
[48] 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. *Proc. VLDB Endow.* 17, 3 (2023), 539–552.
[49] Xinyang Zhao, Xuanhe Zhou, and Guoliang Li. 2023. Automatic Database Knob Tuning: A Survey. *IEEE Trans. Knowl. Data Eng.* 35, 12 (2023), 12470–12490.
[50] Xuanhe Zhou, Chengliang Chai, Guoliang Li, and Ji Sun. 2023. Database Meets Artificial Intelligence: A Survey (Extended Abstract). In *ICDE*. 3901–3902.
[51] Jun-Peng Zhu, Zhiwei Ye, Peng Cai, Donghui Wang, Fengyan Zhang, Dunbo Cai, and Ling Qian. 2024. Log Replaying for Real-Time HTAP: An Adaptive Epoch-Based Two-Stage Framework. In *ICDE*. 2096–2108.
[52] Jun-Peng Zhu, Zhiwei Ye, Xiaolong He, Peng Cai, Xuan Zhou, Aoying Zhou, Dunbo Cai, Ling Qian, Kai Xu, Liu Tang, et al. 2025. SylphDB: An Active and Adaptive LSM Engine for Update-Intensive Workloads. In *ICDE*. 4360–4372.
[53] 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 *SoCC*. 338–350.
[54] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. 2019. A Comprehensive Survey on Transfer Learning. *CoRR* abs/1911.02685 (2019).

Received July 2025; revised October 2025; accepted November 2025