Relation-Aware Bayesian Optimization of DBMS Configurations Guided by Affinity Scores*

Sein Kwon	Seulgi Baek	Hyunseo Yang	Youngwan Jo	Sanghyun Park
Computer Science	*Computer Science*	*Computer Science*	*Computer Science*	*Computer Science*
Yonsei Univercity	*Yonsei Univercity*	*Yonsei Univercity*	*Yonsei Univercity*	*Yonsei Univercity*
Seoul, Korea	Seoul, Korea	Seoul, Korea	Seoul, Korea	Seoul, Korea
seinkwon97@yonsei.ac.kr	seulgibaek@yonsei.ac.kr	hseo0608@yonsei.ac.kr	jyy1551@yonsei.ac.kr	sanghyun@yonsei.ac.kr

***Abstract*—Database Management Systems (DBMSs) are fundamental for managing large-scale and heterogeneous data, and their performance is critically influenced by configuration parameters. Effective tuning of these parameters is essential for adapting to diverse workloads and maximizing throughput while minimizing latency. Recent research has focused on automated configuration optimization using machine learning; however, existing approaches still exhibit several key limitations. Most tuning frameworks disregard the dependencies among parameters, assuming that each operates independently. This simplification prevents optimizers from leveraging relational effects across parameters, limiting their capacity to capture performance-sensitive interactions. Moreover, to reduce the complexity of the high-dimensional search space, prior work often selects only the top few parameters for optimization, overlooking others that contribute meaningfully to performance. Bayesian Optimization (BO), the most common method for automatic tuning, is also constrained by its reliance on surrogate models, which can lead to unstable predictions and inefficient exploration. To overcome these limitations, we propose RelTune, a novel framework that represents parameter dependencies as a Relational Graph and learns GNN-based latent embeddings that encode performance-relevant semantics. RelTune further introduces Hybrid-Score-Guided Bayesian Optimization (HBO), which combines surrogate predictions with an Affinity Score measuring proximity to previously high-performing configurations. Experimental results on multiple DBMSs and workloads demonstrate that RelTune achieves faster convergence and higher optimization efficiency than conventional BO-based methods, achieving state-of-the-art performance across all evaluated scenarios.**

***Index Terms*—Database parameter tuning, Bayesian Optimization, Database Management, Machine Learning**

I. Introduction

In recent years, the exponential growth of data has made the performance of database management systems (DBMSs) increasingly critical for efficiently storing and processing large-scale data [1]. To enhance DBMS performance, various optimization approaches have been actively studied [2]. Among them, database parameter tuning aims to optimize performance by appropriately adjusting the numerous configuration parameters within the DBMS [3]. Traditionally, a Database Administrator (DBA) manually tuned these parameters in a heuristic manner to achieve better performance. However, this manual process has several limitations. First, since modern DBMSs contain a large and diverse set of parameters, it is difficult for a DBA to fully understand all parameters and their complex interactions with system performance. Moreover, because different DBMSs have different types and semantics of parameters, it is challenging for a DBA to consider all possible combinations across systems. To address these challenges, recent research has explored automatic database parameter tuning using machine learning (ML) techniques, which automatically optimize database parameters based on observed configuration and performance data [4]–[12]. Although existing ML-based approaches, such as those leveraging Bayesian Optimization (BO) [13] or Reinforcement Learning (RL) [14], have shown high efficiency compared to heuristic search, they still suffer from several fundamental limitations.

One persistent limitation of existing studies is the neglect of dependencies among configuration parameters. Most existing approaches assume parameter independence and ignore inter-parameter relationships during optimization.

However, in practice, many parameters are interdependent, exhibiting conditional constraints or value-related dependencies as documented in official DBMS manuals [15]. For example, the MySQL Manual states that *"The performance effect of innodb_buffer_pool_instances is only significant when innodb_buffer_pool_size is large enough."* indicating that the influence of one parameter is conditionally dependent on the level of another. Table I summarized multiple examples of such parameter dependencies described in the MySQL 5.7 Manual, demonstrating that these relationships are difficult to capture under optimization frameworks that assume parameter independence. Figure 1 compares the throughput performance of MySQL under two different parameter combinations. The first heatmap (left) illustrates the interaction between `innodb_buffer_pool_size` and `innodb_log_file_size`, revealing a clear conditional dependency between the two parameters. For instance, when `innodb_log_file_size` is fixed at 128 MB, increasing `innodb_buffer_pool_size` from 1 GB to 2 GB improves throughput, whereas under a different condition where `innodb_log_file_size` is set to 512 MB, the same increase results in performance degradation.

In contrast, the right heatmap shows the combination of `innodb_buffer_pool_size` and `max_connections`. In this case, throughput remains nearly constant across different configurations, and no monotonic trend is observed along either

TABLE I: Representative MySQL parameter dependencies. All parameters refer to InnoDB system variables.

Param A	Param B	Manual Evidence
buffer_pool_size	buffer_pool_instances	"... The number of buffer pool instances has no effect unless the pool is large."
io_capacity	max_dirty_pages_pct	"... The rate at which InnoDB flushing rate depends on I/O capacity."
doublewrite	flush_method	"... Doublewrite can bottleneck when using O_DIRECT."

Sources: MySQL 5.7 manual.

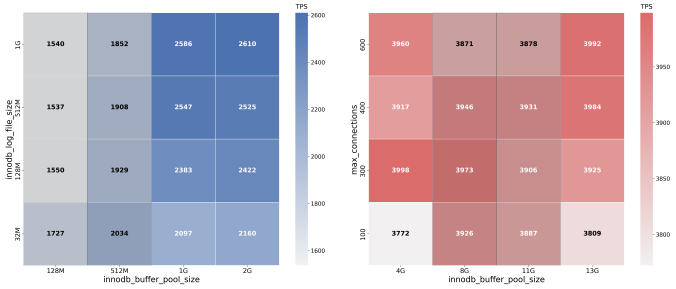


Fig. 1: Throughput comparison of correlated (left) and independent (right) parameter pairs.

parameter axis. The variation range is small, indicating that changes in one parameter do not noticeably influence the effect of the other. This suggests that these two parameters act largely independently without significant interaction.

This comparison demonstrates that database parameters can operate both independently and conditionally dependently, depending on their functional relationships. Consequently, optimization methods assuming parameter independence may overlook critical global optima, highlighting the importance of explicitly modeling parameter relationships for reliable and efficient configuration tuning.

Another limitation of existing studies lies in their handling of the curse of dimensionality in high-dimensional optimization [16], [17]. To alleviate this issue, prior studies typically avoid optimizing the entire configuration space. Instead, they identify and tune only a subset of parameters that are believed to have the greatest impact on performance, often selected using algorithms such as LASSO [18] or SHapley Additive exPlanations (SHAP) [19]. However, such feature selection based approaches may ignore valuable parameters that are meaningful but less dominant. Moreover, the selected parameters often differ across algorithms, resulting in inconsistency. Table II compares the top-10 parameters identified by LASSO and SHAP for two MySQL workloads. The results show that only three or five parameters overlap between the two algorithms for each workload. This inconsistency suggests that the definition of top parameters is highly sensitive to the workload characteristics and the feature selection method. Consequently, the final optimized configurations derived from these different parameter subsets can lead to noticeably different performance outcomes, undermining the reliability of partial-parameter optimization.

A further limitation arises from the inherent characteristics of BO, which is fundamentally a black-box optimization method that heavily depends on the performance of its surrogate model [20], [21]. This dependency becomes particularly problematic in noisy or high-dimensional search spaces, where the surrogate tends to produce unreliable predictions, ultimately leading to inefficient optimization outcomes. Moreover, this issue often leads to instability during the early exploration phase, causing the surrogate to repeatedly explore irrelevant regions of the search space, thereby reducing overall optimization efficiency.

To address the aforementioned limitations, we propose **RelTune**, a relation-aware Bayesian Optimization framework that integrates parameter dependencies and affinity-based guidance for efficient DBMS configuration tuning. To overcome the limitation of prior studies that overlooked such parameters relationships, RelTune leverages a Large Language Model (LLM) to extract descriptions of parameters, covering functionalities and interactions, from official DBMS documentation. Based on these descriptions, RelTune constructs a parameter dependency graph that captures complex relationships among parameters, thereby enabling the use of domain knowledge during the tuning process. To further mitigate the curse of dimensionality and the information loss caused by optimizing only a small subset of parameters, RelTune employs a Graph Neural Network (GNN) [22] to learn representations from the constructed graph. The GNN encodes the relationships among parameters and compresses them into a low-dimensional latent space. This latent representation not only reduces dimensionality but also preserves the structural dependencies among parameters, enabling efficient optimization across the entire configuration space. Additionally, to overcome the exploration inefficiency of conventional BO, we introduce an Affinity Score that enhances the acquisition process by incorporating the structural information of the latent space. This score guides the search toward regions that are close to previously high-performing latent vectors, thereby concentrating the optimization process on promising areas. We evaluate RelTune on two DBMSs, MySQL [23] and PostgreSQL [24], across four and two workloads, respectively. Experimental results demonstrate that RelTune achieves substantial performance improvements over state-of-the-art methods.

Our main contributions are summarized as follows:

- We construct a relational graph that captures structural dependencies among DBMS parameters based on descriptions generated by LLMs.
- We train a GNN to jointly encode parameter relations and performance metrics into a compact latent space, enabling holistic optimization over all parameters.
- We propose a Hybrid Score-Guided Bayesian Optimization that incorporates an Affinity Score to enhance exploration efficiency and tuning stability.
- Experimental results demonstrate that RelTune achieves superior performance through comparison with various

TABLE II: Top-10 parameters using SHAP and LASSO for MySQL Read50Update50, Read95Update5 workloads. Overlapped parameters are highlighted in bold.

Read50Update50	
SHAP (top-10)	**Lasso (top-10)**
range_alloc_block_size	**range_alloc_block_size**
innodb_buffer_pool_size	**innodb_buffer_pool_size**
innodb_spin_wait_delay	**innodb_spin_wait_delay**
innodb_purge_threads	**innodb_stats_persistent**
innodb_log_file_size	optimizer_prune_level
innodb_stats_persistent	innodb_optimize_filltext_only
innodb_old_blocks_time	innodb_read_io_threads
innodb_ft_min_token_size	**innodb_log_file_size**
binlog_cache_size	max_error_count
inodb_compression_failure_threshold_pct	innodb_ft_max_token_size

Read95Update5	
SHAP (top-10)	**Lasso (top-10)**
range_alloc_block_size	**range_alloc_block_size**
innodb_buffer_pool_size	**innodb_spin_wait_delay**
innodb_spin_wait_delay	innodb_optimize_fulltext_only
innodb_random_read_ahead	innodb_file_per_table
innodb_purge_threads	binlog_cache_size
sort_buffer_size	innodb_print_all_deadlocks
innodb_read_ahead_threshold	**innodb_random_read_ahead**
innodb_lru_scan_depth	innodb_read_io_threads
table_open_cache	query_cache_wlock_invalidate
innodb_adaptive_hash_index_parts	innodb_online_alter_log_max_size

baseline models.

II. PRELIMINARIES

A. Automatic Database Parmeter Tuning

Database parameter tuning aims to maximize the performance of a DBMS by optimizing its configuration parameters. Let the set of parameters be denoted as P, where $P_1, \ldots, P_k$ represent K parameters. A collection of these parameters, $conf = P_1, \ldots, P_k$, is defined as a configuration. The objective function is defined based on two key DBMS performance metrics: throughput and latency. Throughput measures the amount of work completed per unit time and should be maximized, while latency represents the delay per operation and should be minimized. Automatic parameter tuning approaches can be broadly categorized into search-based methods [25] and ML-based methods [4]–[12].

Search-based methods aim to identify optimal database configurations by following predefined rules or heuristics. A representative approach is BestConfig [25], which collects configuration and metric pairs and employs a sampling strategy called DDS (Divide and Diverge Sampling) to efficiently explore the high-dimensional parameter space while maintaining broad applicability. After identifying the configuration that yields the best performance, BestConfig further refines the search by proposing new parameter settings within a bounded region around the current optimum using the RBS (Recursive Bound and Search) algorithm. However, such rule-based tuning approaches are inherently data dependent. Their performance can become unstable when the collected data are noisy or limited in quantity. In addition, since they rely on heuristic exploration rather than learned models, these methods often require substantial time to converge to an optimal configuration.

ML-based methods leverage learned models to predict performance and identify optimal parameter configurations. A representative approach is OtterTune [6], which stores configuration and metric pairs generated during the optimization process in a data repository. These data are then used during the workload mapping stage to enable optimization across different workloads. To alleviate the inefficiency of exploring high-dimensional spaces, OtterTune employs LASSO to select the most influential parameters and trains a Gaussian Process (GP)–based [26] surrogate model only on this selected subset. However, since only a subset of parameters is optimized, valuable parameters may be omitted, and parameter relationships are not considered. Moreover, as OtterTune relies on BO, its performance remains highly dependent on the predictive accuracy of the surrogate model.

CDBTune [8] adopts a RL approach that learns database configurations through a try-and-error process. To enable optimization in high-dimensional parameter spaces, CDBTune employs the Deep Deterministic Policy Gradient (DDPG) algorithm, which combines the advantages of the Deep Q-Network (DQN) [27] and Actor–Critic [28] methods. However, RL-based tuning methods such as CDBTune require a large amount of high-quality data and numerous iterations to achieve convergence. Moreover, since each configuration must be evaluated through actual benchmarking, the overall tuning process becomes time-consuming and computationally expensive.

RGPE (Ranking-weighted Gaussian Process Ensemble) [10] represents one of the most effective algorithms among existing ML-based database tuning approaches, as demonstrated in comparative studies evaluating various methodologies such as knowledge transfer and workload mapping. RGPE is an ensemble model designed for BO-based optimization, which combines both workload-specific information and historical knowledge to guide the search process. However, similar to OtterTune, RGPE is unable to efficiently handle high-dimensional parameter spaces and therefore performs optimization only on a selected subset of parameters after a parameter selection step. In addition, the model requires careful tuning of hyper-parameters, which further increases the complexity of the optimization process.

CSAT (Confidence-Aware Surrogate-Assisted Tuning) [11] is a BO-based framework designed to improve the efficiency and stability of optimization by incorporating the confidence of the surrogate model. To address the issue in conventional BO methods that often ignore surrogate uncertainty and consequently explore unreliable regions, CSAT introduces an acquisition function that integrates the surrogate's confidence, thereby guiding the search toward more reliable regions of

the parameter space. This approach enhances the stability of exploration, particularly under noisy environments. However, CSAT still fails to explicitly capture the inter-dependencies among parameters. While it contributes to improving the reliability of the surrogate model, it does not account for the structural relationships or dependencies among configurations, making it incapable of leveraging relational knowledge for more efficient exploration.

GPTuner [12] is an LLM-based automatic database tuning framework that aims to overcome the limitations of conventional ML-based methods, which fail to fully leverage domain knowledge. To achieve this, GPTuner collects and refines domain knowledge using a LLM (GPT-4) [29] and transforms it into a structured representation that can be utilized during the optimization process. It further introduces a Coarse-to-Fine BO strategy, which progressively explores a knowledge-guided, reduced search space to achieve efficient optimization. While GPTuner effectively reduces tuning cost by structurally incorporating domain knowledge through LLMs, it still does not explicitly model the dependencies among parameters. Specifically, GPTuner relies on LLM-generated explanatory, parameter-wise guidelines, but it does not represent parameter interactions in a graph-based form or learn the structural correlations within the search space.

B. Bayesian Optimization

BO [13]is widely used for black-box function optimization and is particularly effective for problems with expensive evaluation costs, such as database parameter tuning. BO learns a surrogate model based on the observed configuration–metric pairs (X, Y) to approximate the underlying objective function. An acquisition function, denoted as $a(x)$, is then used to select the next candidate configuration for evaluation. In most cases, a GP [26] is adopted as the surrogate model, which models the objective function as a probabilistic distribution defined by a mean function $\mu(x)$ and a covariance (kernel) function $k(x, x')$. This formulation allows the GP to estimate both the expected value and the uncertainty of each configuration. The acquisition function utilizes this predictive distribution to balance exploration and exploitation. Typical acquisition functions, such as Expected Improvement (EI) [30] and Upper Confidence Bound (UCB) [31], use this predictive distribution to guide the selection of the next configuration x_t, which is determined as follows:

$$x_t = \arg\max_{x \in X} a(x) \tag{1}$$

At each iteration, BO updates the surrogate model and selects a new candidate configuration to gradually move toward the global optimum. The dataset is then updated as

$$D_t = D_{t-1} \cup \{(x_t, y_t)\}, \tag{2}$$

where x_t and y_t denote the newly explored configuration and its corresponding performance metric, respectively. However, BO is highly sensitive to the predictive accuracy of its surrogate model. When the surrogate model fails to accurately approximate the underlying objective function, owing to limited training samples, measurement noise, or irregular performance landscapes, the acquisition function can lead the optimization toward suboptimal or unstable regions. These limitations directly motivate the design of RelTune.

III. Method

A. Overview

We propose RelTune, a novel framework that efficiently explores the entire parameter space by representing semantic relationships among parameters as a graph, encoding them into a Graph Neural Network (GNN)-based [22] latent representation, and performing Hybrid Score–Guided Bayesian Optimization (Hybrid BO). The overall architecture of RelTune is illustrated in Figure 2, which consists of three main components.

Step 1: Relational Graph Construction. In the first stage, we construct a parameter relationship graph that captures semantic dependencies among DBMS configuration parameters. Specifically, we first use a Large Language Model (LLM; GPT-4 [29]) to generate textual descriptions that capture the relationships among parameters. These descriptions are then transformed into embedding vectors through the LLM API, and pairwise cosine similarity between the embeddings is computed to form edges in the relational graph.

Step 2: Graph-Based Latent Representation. In the second stage, we inject configuration values into the nodes of the relational graph to generate multiple samples and train a GNN. The GNN learns to encode both parameter dependencies and performance metrics into a latent representation. Additionally, a decoder is trained to reconstruct the original configuration from the latent representation, ensuring that the learned embedding preserves the structural information of the original parameter space.

Step 3: Latent Space Optimization via Hybrid-Score-Guided Bayesian Optimization. In the third stage, optimization is performed over the latent representations generated in Step 2. Instead of applying Vanilla Bayesian Optimization (VBO), we adopt a Hybrid-Score-Guided BO (HBO) that integrates an Affinity Score, which reflects the relational proximity between candidate points and high-performing configurations. This approach enables more efficient exploration by focusing the search on promising regions of the latent space associated with superior performance.

B. Relational Graph Construction

1) ***Parameter Embedding:*** To extract semantic relationships among parameters across various DBMSs, we leverage extensive domain knowledge sources such as official DBMS manuals and community forum discussions through a Large Language Model (LLM), GPT-4 [29]. We prompt the model with queries of the form:

```
For a ''given DBMS (specify
the version)'', explain the
functionality of each parameter
and list related parameters that
affect its behavior.
```

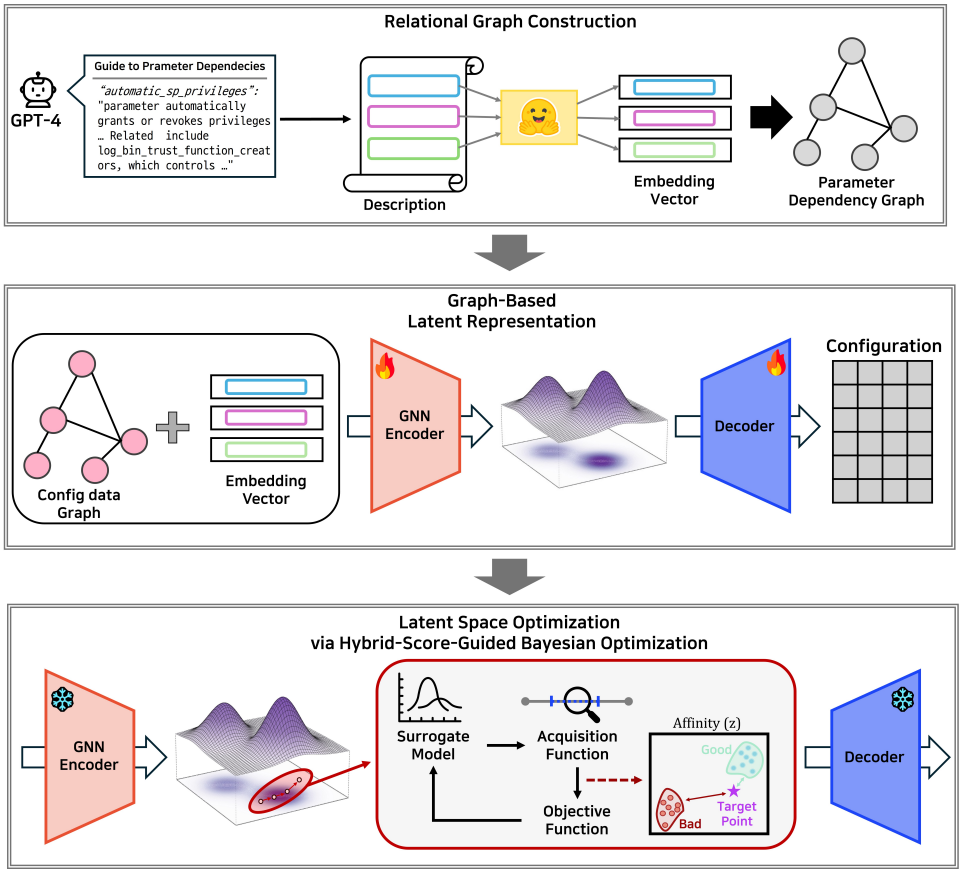


Fig. 2: Overview Architecture of RelTune

This prompt is designed to elicit functional descriptions and inter-parameter dependencies directly from domain texts, allowing the LLM to capture semantically grounded relationships among configuration parameters. Including the specific DBMS version in the prompt helps ensure that the extracted descriptions and parameter relationships reflect version-specific behaviors. Although most parameter dependencies are generally consistent across versions, version-level context provides more precise and up-to-date information, thereby improving the reliability of the extracted knowledge.

This process automatically generates textual descriptions for each parameter. For example, when queried about the parameter `automatic_sp_privileges`, the model produces an output in the following format:

Listing 1: Example LLM Output

```
"automatic_sp_privileges": "parameter automatically
grants or revokes privileges for stored procedures
when they are created or dropped. It simplifies
stored procedure privilege management. Related
include log_bin_trust_function_creators, which
controls whether function creators must have SUPER
privilege."
```

The generated descriptions serve as sentence-level data that encapsulate the semantic characteristics and relational information of each parameter. To embed these descriptions, we employ the paraphrase-MiniLM-L6-v2 [32] model from the Sentence-Transformers library. This model, fine-tuned on paraphrase datasets, is designed to capture contextual semantic similarity rather than mere lexical overlap [33], making it well-suited for measuring functional relationships among DBMS parameters. In particular, the Sentence Transformer architecture is optimized such that cosine similarity directly reflects the semantic distance between embeddings, which aligns naturally with our graph construction approach that defines edges based on similarity thresholds. Moreover, the lightweight MiniLM structure ensures high computational efficiency, enabling scalable embedding even for large sets of parameters.

*2) **Graph Construction:*** For the embedding vectors generated in the previous step, let $\mathbf{e}_i$ and $\mathbf{e}_j$ denote the embedding representations of parameters i and j, respectively. The semantic similarity between two parameters is defined using cosine similarity, as follows:

$$S_{ij} = \frac{\mathbf{e}_i \cdot \mathbf{e}_j}{\|\mathbf{e}_i\| \, \|\mathbf{e}_j\|} \tag{3}$$

This similarity represents the probabilistic proximity of two parameters belonging to the same functional context, and it is used to define the edges of the relational graph. The relational graph $G = (V, E)$ consists of a set of parameters V and their relationship E. Each node corresponds to a parameter, and an

edge between parameters i and j is established based on their cosine similarity S_{ij}. The binary adjacency matrix $A \in \mathbb{R}^{n \times n}$ is defined as:

$$A_{ij} = \begin{cases} 1, & \text{if } S_{ij} \geq \tau, \\ 0, & \text{otherwise,} \end{cases} \tag{4}$$

where τ is the similarity threshold controlling edge creation. If the similarity exceeds τ, an edge is created between the corresponding nodes; otherwise, no edge is added.

In this study, the cosine similarity threshold τ is set to 0.75, which was empirically found to provide the best balance between graph density and modularity (IV-C). The graph constructed with this threshold prevents excessive connectivity while encouraging semantically related parameters to form clusters within the same subsystem (e.g., InnoDB, Query Optimizer). Consequently, the resulting relational graph structurally encodes the semantic dependencies and functional correlations among parameters, providing a solid foundation for the GNN-based encoder to effectively model parameter relationships.

C. Graph Based Latent Representation

1) ***Relational Graph Representation Learning:*** To compress the high-dimensional relational graph of parameters into a low-dimensional latent space, we employ a GNN–based encoder. To reconstruct the original high-dimensional configuration from this latent representation, an autoencoder-style decoder is additionally trained. (III-C2)

Based on the relational graph constructed in the previous stage, we inject the parameter values of each configuration sample into the corresponding nodes. The embedding vectors of all nodes are then concatenated to jointly reflect both the numerical and semantic characteristics of the parameters. Through this design, each node simultaneously encodes a parameter's quantitative and semantic properties, while all configuration samples share the same edge structure E but differ in their feature distributions. As a result, each configuration sample can be represented as an instance-level graph.

To learn latent representations from the constructed graphs, we employ an encoder based on the Graph Attention Network (GAT) [34] architecture. GAT enables each node to selectively aggregate information from its neighbors using attention weights, rather than a simple average, allowing the model to learn the relative importance of parameter interactions. The node representation at layer l is defined as follows:

$$\mathbf{e}_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}^{(l)} \, \mathbf{W}^{(l)} \mathbf{e}_j^{(l)}\right) \tag{5}$$

The $W^{(l)}$ denotes a trainable linear transformation, and $\sigma(\cdot)$ represents a nonlinear activation function (e.g., ELU). The attention coefficient $\alpha_{ij}^{(l)}$, which indicates the importance of node j information when being aggregated into node i, is computed using a softmax-based normalization as follows:

$$\alpha_{ij}^{(l)} = \mathrm{softmax}_j\Big(\mathrm{LeakyReLU}\Big(\mathbf{a}^\top \big[\mathbf{W}^{(l)}\mathbf{e}_i^{(l)} \parallel \mathbf{W}^{(l)}\mathbf{e}_j^{(l)}\big]\Big)\Big). \tag{6}$$

This attention mechanism enables the model to learn anisotropic relationships among parameters, allowing it to emphasize semantically important interactions rather than relying solely on structural connectivity. In the final layer, all node embeddings are aggregated via average pooling to produce a latent vector z that represents the entire graph.

$$\mathbf{z} = \frac{1}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} \mathbf{e}_i^{(L)} \tag{7}$$

This latent vector serves as a compact representation that summarizes the overall performance patterns and relational dependencies among database parameters. It is subsequently used as the search space for the optimization phase.

2) ***Performance Aware Latent Reconstruction:*** To reconstruct the latent vector z into its original configuration form, we train an autoencoder-style decoder [35]. The decoder trains a mapping from the latent space back to the original high-dimensional configuration, denoted as $\hat{x} = \mathcal{D}(z)$, thereby encouraging the latent representation to preserve the continuity and semantic consistency across configurations. In addition, to ensure that the latent space captures not only configuration structure but also performance variations, we train a metric prediction head e_ψ in parallel to predict throughput and latency directly from the latent vector. This auxiliary objective injects performance-aware information into the latent representation, enabling it to reconstruct not only the configuration itself but also the underlying structural relationships between parameter values and performance outcomes. The decoder and the metric prediction head are jointly trained to minimize the loss function.

$$\mathcal{L} = \lambda_{recon} \|x - \hat{x}\|_2^2 + \lambda_{metric} \|y - \hat{y}\|_2^2 \tag{8}$$

Here, the reconstructed configuration and the predicted performance are denoted as $\hat{x}$ and $\hat{y} = e_\psi(z)$, respectively, while λ_{recon} and λ_{metric} are hyper-parameters that control the relative importance of the two terms. The reconstruction loss encourages the latent space to preserve the structural properties of the original configuration space, whereas the metric prediction loss aligns the latent space such that configurations with similar performance are positioned close to each other.

D. Latent Space Optimization via Hybrid-Score-Guided Bayesian Optimization

Conventional BO determines the next search region based solely on the predictions of the surrogate model within the latent space. However, this approach often requires a large number of exploration steps to achieve reliable predictions. In high-dimensional search space, such as database parameters, BO becomes inefficient and is prone to getting trapped in local minima. To overcome these limitations, we propose a Hybrid-Score-Guided Bayesian Optimization (HBO) framework that combines the surrogate model's predicted performance score

with an affinity score, which measures the proximity between a candidate latent vector and the set of previously identified high-performing latent vectors. Formally, for each latent vector z, the Hybrid Score is defined as follows:

$$\text{HybridScore}(z) = f_{\text{metric}}(z) + \gamma \cdot f_{\text{affinity}}(z) \tag{9}$$

Here, f_{metric} denotes the performance score computed from the predicted throughput and latency obtained via the metric prediction head described in Section III-C2 (Performance Aware Latent Reconstruction). Since higher throughput and lower latency indicate better performance, the score is defined as follows:

$$f_{\text{metric}}(z) = f_{\text{Throughput}}(z) - \alpha \cdot f_{\text{Latency}}(z), \tag{10}$$

where, α is a hyperparameter that controls the importance of latency in the performance score computation. In addition, $f_{\text{affinity}}(z)$ measures how close a candidate latent vector z is to the set of previously identified high performing latent vectors. Rather than forcing convergence toward a single mean point,as in conventional BO, this term encourages exploration within regions densely populated by good samples, enabling more fiverse and performance-aware search. Specifically, let Z_{good} denote the set of latent vectors whose performance exceeds a predefined threshold. The affinity score is the computed as follows:

$$f_{\text{affinity}}(z) = \frac{1}{|Z_{\text{good}}|} \sum_{z_k \in Z_{\text{good}}} \exp\left(-\frac{\|z - z_k\|^2}{2\sigma^2}\right), \tag{11}$$

where the distance between z and each $z_k \in \mathcal{Z}_{\text{good}}$ is converted into a similarity value using an RBF (Radial Basis Function) kernel, and the average is taken over all high-performing samples. Consequently, a candidate vector z located closer to high-performance regions yields a higher affinity score.

The hybrid score achieves two complementary effects. First, $f_{\text{metric}}(z)$ enables exploitative exploration within regions of the latent space that are predicted to yield high performance. Second, $f_{\text{affinity}}(z)$ encourages exploration toward regions of the latent space that are densely populated by previously high-performing configurations, thereby facilitating the discovery of new optimal configurations. (Algorithm 1, L6-L17)

Based on the defined hybrid score, BO is performed to identify the optimal latent vector within the latent space. Within each BO loop, the hybrid score serves as the objective function for training the surrogate model. A GP–based surrogate is iteratively updated to approximate the hybrid score, and the Expected Improvement (EI) acquisition function is employed to select the next exploration point with the highest potential for improvement. The EI function measured the expected gain over the current best hybrid score and is defined as follows:

$$\begin{aligned}\text{EI}(z) &= \mathbb{E}\big[\max\big(0,\, f(z) - f(z^+)\big)\big] \\ &= \big(\mu(z) - f(z^+)\big)\,\Phi\big(\gamma(z)\big) + \sigma(z)\,\phi\big(\gamma(z)\big),\end{aligned} \tag{12}$$

where $\mu(z)$ and $\sigma(z)$ denote the predictive mean and standard deviation of the GP, $f(z^+)$ represents the current best hybrid score, and $\Phi(\cdot)$ and $\phi(\cdot)$ are the cumulative and probability density functions of the standard normal distribution, respectively.

Since direct performance evaluation of each optimized z is computationally expensive, the actual performance is predicted using the pretrained GAT-based metric prediction head instead of conducting costly benchmark executions. (Algorithm 1, L9-10)

Algorithm 1 Hybrid-Score-Guided Bayesian Optimization (HBO)

Require: Pre-trained GNN model $\mathcal{M}$ (with $f_{\text{enc}}, g_{\text{dec}}$, and e_ψ)
1: Initial dataset $\mathcal{D}$ (with graph G, performance P; Hyperparameters α, γ, σ; Iterations T
Ensure: Optimized configuration c^*
2: $Z_{\text{all}} \leftarrow \{f_{\text{enc}}(G_i) \mid G_i \in \mathcal{D}\}$
3: $Z_{\text{good}} \leftarrow \{z_{\text{i}} \in Z_{\text{all}} \mid \text{is\_good}(\text{performance}_i)\}$
4: Define search space $\mathcal{S}$ from the bounds of Z_{all}
5: Initialize Gaussian Process (GP) model $\mathcal{GP}$
6: **for** $t = 1$ to T **do**
7: $z_{\text{t}} \leftarrow \arg\max_{z \in \mathcal{S}} \text{EI}(z \mid \mathcal{GP})$
8: Calculate Hybrid Score for the proposed point z_t
9: $f_{\text{TPS}}(z_{\text{t}}), f_{\text{Latency}}(z_{\text{t}}) \leftarrow e_\psi(z_{\text{t}})$
10: $f_{\text{metric}}(z_{\text{t}}) \leftarrow f_{\text{TPS}}(z_{\text{t}}) - \alpha \cdot f_{\text{Latency}}(z_{\text{t}})$
11: $f_{\text{affinity}}(z_{\text{t}}) \leftarrow \frac{1}{|Z_{\text{good}}|} \sum_{z' \in Z_{\text{good}}} \exp\left(-\frac{\|z_t - z'\|_2^2}{2\sigma^2}\right)$
12: $\text{HybridScore}(z_{\text{t}}) \leftarrow f_{\text{metric}}(z_{\text{t}}) + \gamma \cdot f_{\text{affinity}}(z_{\text{t}})$
13: Update $\mathcal{GP}$ with the observation $(z_{\text{t}}, \text{HybridScore}(z_{\text{t}}))$
14: **end for**
15: $z^* \leftarrow$ the latent vector with the best observed HybridScore
16: $c^* \leftarrow g_{\text{dec}}(z^*)$
17: **return** c^*

IV. Experiment

A. Experiment Setup

Hardware. All MySQL and PostgreSQL performance measurements were performed on a server running the Ubuntu 20.04.6 operating system with an Intel® Core™ i7-11700 @ 2.52GHZ, 32 GB of RAM, and a 256 GB disk.

Tuning Setting. We conducted experiments using *MySQL v5.7.37* [23] and *PostgreSQL v14.19* [24]. For all experiments, the models were trained on a dataset consisting of 5,000 configuration and performance samples. Each workload experiment was repeated five times with different random seeds, and the reported results represent the average of these runs. For baselines that optimize only the top-ranked parameters, the number of selected parameters k was set to 5, 10, and 15, respectively, and the best-performing result among them was used for comparison.

Baselines. We compared our method with the following representative tuning frameworks: **OtterTune** [6], **RGPE** [10], and **CSAT** [11], which employ BO over a subset of top

TABLE III: MySQL Workload Information

MySQL							
Workload Index	Scale Factor	Data Size	Read	Insert	Scan	Update	Read Modify Write
A	12000	15GB	**50%**	-	-	**50%**	-
B			**95%**	-	-	**5%**	-
E			-	**5%**	**95%**	-	-
F			**50%**	-	-	-	**50%**

parameters; **GPTuner** [12], which leverages large language models to extract and convert domain knowledge for BO-based tuning; and **CDBTune** [8], which performs RL-based tuning. All baselines were executed under identical hardware and workload conditions for a fair comparison.

Workloads. We evaluated the proposed method on four workloads for MySQL and two workloads for PostgreSQL. For MySQL, we adopted multiple combinations of **YCSB (Yahoo! Cloud Serving Benchmark)** [36] workloads, which define diverse ratios of database operations such as *load*, *insert*, *read*, *update*, and *scan*. The workload configurations are summarized in Table III. For PostgreSQL, we used two representative workloads: the OLTP workload **TPC-C** [37] and the OLAP workload **TPC-H** [38]. All workloads were executed using **BenchBase** [39], [40], which reports throughput as *transactions per second (TPS)* by treating each execution unit, including individual queries in TPC-H, as a single transaction. To ensure consistency, we uniformly report throughput in TPS units across all workloads, together with corresponding latency results.

Implementation Details. Our model projected to 32 dimensions for MySQL and PostgreSQL. We set the same 300 iterations for all optimization processes in our experiments. In the performance function f_{metric}, we set $\alpha = 0.5$ to reflect the trade-off between throughput and latency, ensuring a balanced optimization objective between the two metrics. In the Hybrid Score, the parameter $\gamma = 1.0$ was chosen to perform balanced exploration by equally weighting the metric prediction and the affinity score. These hyperparameters are user-adjustable, allowing flexible trade-offs between throughput, latency, and exploration intensity depending on the workload characteristics.

B. Performance Comparison

Figure 3 and 4 present the throughput and latency results across different workloads for both MySQL and PostgreSQL. For MySQL, RelTune consistently outperforms all baseline methods across four YCSB workloads (YCSB A, B, E, and F). In particular, under YCSB E, RelTune achieves the largest performance gain, improving throughput by 56.5% compared to the default configuration. YCSB B is a read-heavy workload with a very low proportion of write or update operations. Due to this characteristic, the influence of parameters related to internal buffering or log writing is relatively limited, resulting in a small

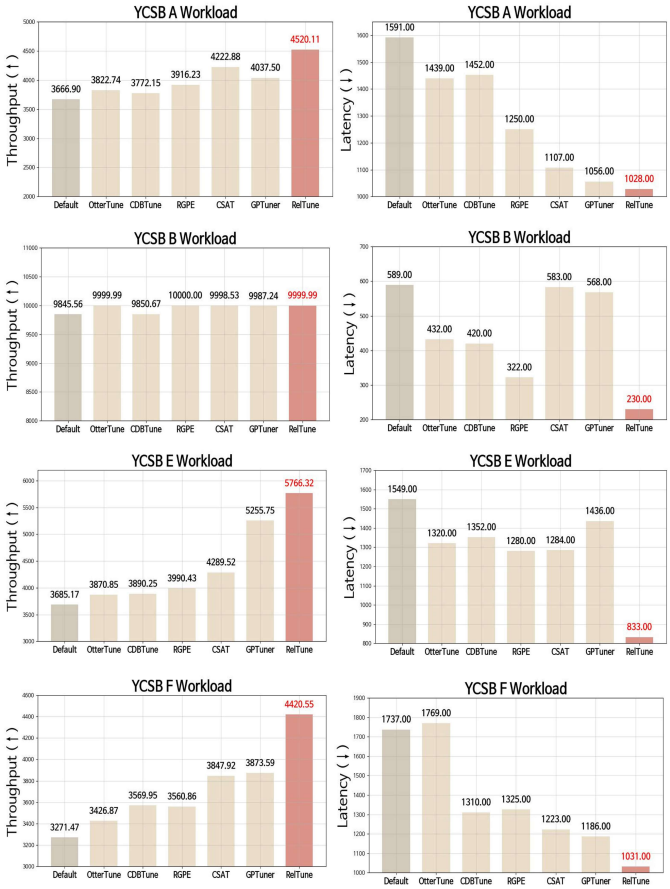


Fig. 3: Performance improvement for MySQL YCSB A,B,E and F.

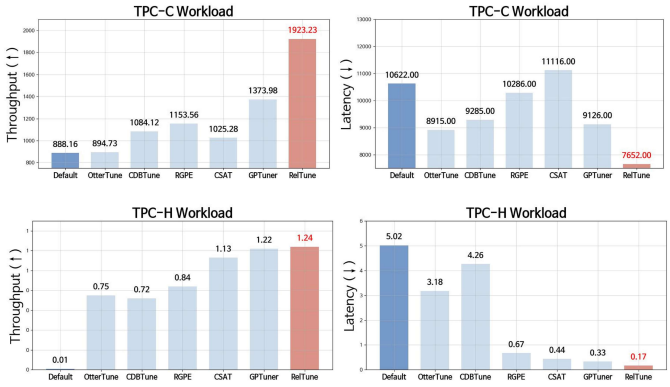


Fig. 4: Performance improvement for PostgreSQL TPC-C and TPC-H.

improvement in throughput and other baselines. Nevertheless, RelTune achieves a substantial reduction in latency, with a 60.9% improvement compared to the default performance.

Across both the OLTP workload (TPC-C) and the OLAP workload (TPC-H), RelTune shows superior performance in terms of both throughput and latency. To maintain a consistent evaluation metric between the MySQL and PostgreSQL

experiments, we adopted a unified throughput-based evaluation framework for both DBMSs. This consistency was necessary because evaluating TPC-H using only latency or QPS would make it difficult to directly compare the relative performance improvements with other workloads, such as YCSB.

As a result, RelTune exhibits higher performance stability and superior optimization capability compared to both traditional BO-based or RL-based tuning approaches and recent baseline models that optimize only a subset of parameters. Moreover, RelTune consistently delivers performance improvements across both OLTP and OLAP environments, demonstrating its robustness and general applicability.

C. Graph Quality Analysis

We constructed the parameter relationship graph by connecting edges only between parameter nodes whose embedding cosine similarity exceeded 0.75. To validate the effectiveness of the threshold, we conducted a comprehensive evaluation of the structural and semantic quality of the resulting graph from multiple perspectives.

First, we evaluated the structural cohesiveness of the graph by computing its modularity (Q). Modularity measures how strongly the nodes are grouped into communities by comparing the proportion of intra-community edges in the actual graph to that expected in a random graph with the same number of nodes and edges. A higher Q value indicates that the graph exhibits a well-defined community structure, implying that edges are not randomly connected but instead form functionally cohesive groups. The modularity Q is defined as follows:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j), \tag{13}$$

Here, A_{ij} denoted the element of the adjacency matrix, k_i and k_j represent the degrees of nodes i and j, respectively, m is the total number of edges, and $\delta(c_i, c_j)$ is an indicator function that equals 1 if nodes i and j belong to the same community, and 0 otherwise.

Additionally, we we evaluated the semantic consistency of the graph by applying the Louvain community detection algorithm. To measure the alignment between the communities identified by the Louvain algorithm and the predefined parameter subsystems, we employed two clustering evaluation metrics: Normalized Mutual Information (NMI) and Adjusted Rand Index (ARI). Here, each subsystem was defined by grouping parameters according to their prefixes (e.g., innodb_..., optimizer_...). NMI measures the overall consistency between the detected community structure and the predefined subsystems, while ARI quantifies the pairwise labeling agreement adjusted for random chance. Higher values of both metrics indicate stronger semantic alignment between the graph's community structure and the actual subsystem organization.

However, since the subsystem prefixes of parameters do not always correspond to their functional relationships, we jointly considered modularity, representing structural cohesiveness, and NMI, ARI, reflecting semantic consistency, to comprehensively validate the quality of the constructed graph.

TABLE IV: Graph quality metrics of the MySQL parameter graph.

Threshold	Modularity	NMI	ARI
0.65	0.45	0.45	0.21
0.70	0.61	0.45	0.11
☺ **0.75**	**0.77**	**0.46**	**0.12**
0.80	0.87	0.46	0.03
0.85	0.92	0.45	0.01

TABLE V: Graph quality metrics of the PostgreSQL parameter graph.

Threshold	Modularity	NMI	ARI
0.65	0.72	0.58	0.32
0.70	0.79	0.60	0.25
☺ **0.75**	**0.89**	**0.60**	**0.17**
0.80	0.91	0.56	0.02
0.85	0.81	0.55	0.01

Table IV and V summarize the evaluation results for the parameter relationship graphs of MySQL and PostgreSQL, where the cosine similarity threshold was varied from 0.65 to 0.85. A comprehensive analysis of the three evaluation metrics (Modularity, NMI and ARI) reveals a common trend across MySQL and PostgreSQL: as the threshold increases, Modularity improves initially, but ARI begins to drop sharply beyond a certain point.

For MySQL, the modularity reached its peak value (0.92) when the threshold was set to 0.85. However, at this point, the ARI sharply dropped to 0.01, indicating severe over-segmentation of communities. This suggests that the graph became excessively sparse, leading to the loss of essential inter-subsystem connections.

In contrast, PostgreSQL exhibited more moderate variations in the evaluation metrics as the threshold changed, which can be attributed to its more homogeneous correlation structure among parameters compared to MySQL. Specifically, when the threshold was set to 0.80, the modularity increased to 0.91, but the ARI sharply declined to 0.02, indicating a degradation in the semantic alignment of the graph.

In summary, Taken together, these results confirm that a threshold of 0.75 provides the optimal balance between structural cohesiveness (Modularity) and semantic consistency (ARI) for both DBMSs. For MySQL, the modularity remained sufficiently high at 0.77, while the ARI stayed stable, and for PostgreSQL, both the modularity (0.89) and ARI (0.17) maintained steady levels, preserving subsystem-level semantics without excessive fragmentation. Accordingly, this study adopts 0.75 as the common threshold, as it best captures the trade-off between semantic relatedness and structural stability in constructing parameter relationship graphs for both DBMSs.

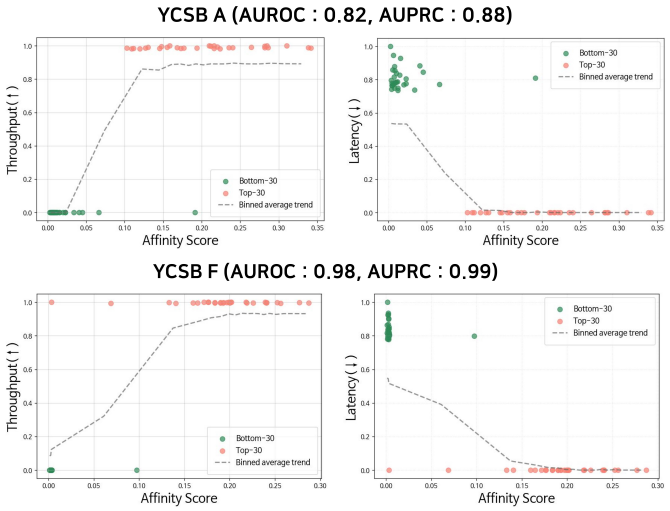


Fig. 5: Correlation between Affinity Score and Performance Metrics for MySQL.

D. Validation of the Affinity Score

To verify whether the proposed affinity score reliably reflects the relationship between the latent representations and the actual performance, we analyzed and evaluated the correlation between the affinity score and the normalized throughput and latency. Specifically, we selected the top-30 and bottom-30 configurations based on performance (throughput and latency) and converted them into their corresponding latent vectors. Using these two sets, we computed the affinity score of each sample by measuring its distance to the latent vectors of previously well-performing configurations. This evaluation allows us to verify whether higher affinity scores are indeed associated with better throughput and lower latency in practice.

For quantitative evaluation, we employed AUROC (Area Under the ROC Curve) and AUPRC (Area Under the Precision–Recall Curve), and additionally performed visual analysis to support the quantitative findings. AUROC represents the probability that the model correctly distinguishes between high-performing and low-performing latent vectors, while AUPRC measures the balance between precision and recall for identifying high-performing latent vectors. Both metrics range from 0 to 1, with higher values indicating a stronger ability to discriminate well-performing configurations. In this study, these metrics were used to evaluate whether the proposed Affinity Score exhibits a strong correlation with actual performance. The dashed line in the figure represents the binned average trend, which shows the average performance variation within each affinity interval.

Figure 5 presents the results for MySQL on the YCSB A and YCSB F workloads. The upper plot corresponds to the YCSB A workload, while the lower plot shows the result for YCSB F.

For the YCSB A workload, the binned trend clearly demonstrates that as the affinity score increases, throughput gradually rises while latency decreases, revealing a consistent overall trend. The affinity score exhibited a clear correlation with throughput and latency, showing AUROC = 0.82 and AUPRC = 0.88 in quantitative evaluation. Latent vectors with higher affinity values tended to achieve higher throughput and lower latency, indicating that the proposed affinity score effectively captures the performance-relevant structure within the latent space. Although a few latent vectors in the mid-range affinity intervals exhibit relatively lower performance, the overall results indicate that the Affinity Score reliably identifies high-performing regions within the latent space.

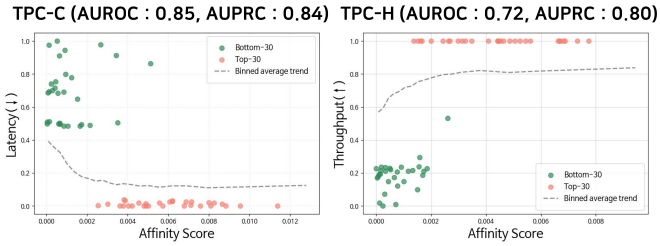


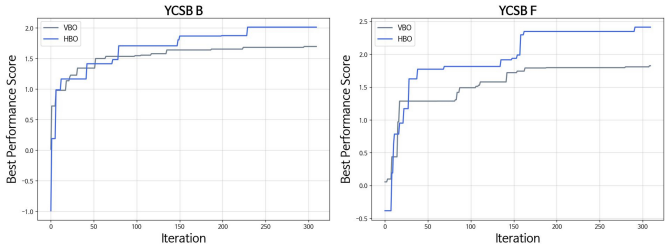
Fig. 6: Correlation between Affinity Score and Performance Metrics for PostgreSQL.

For the YCSB F workload, the correlation between the affinity score and performance was even more pronounced, with AUROC = 0.98 and AUPRC = 0.99, indicating a very strong relationship. The relation between affinity and throughput exhibited an almost linear increasing pattern, where latent vectors with higher affinity values consistently achieved higher throughput and lower latency.

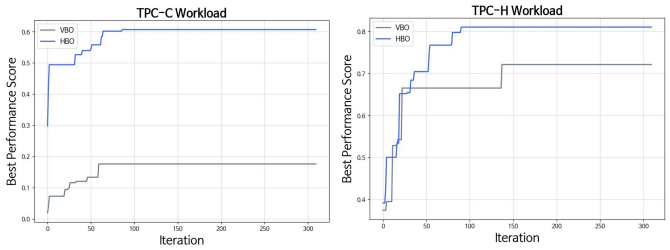
Figure 6 shows the results for PostgreSQL on the TPC-C and TPC-H workloads. For the TPC-C workload, the relationship between latency and the affinity score exhibited a highly consistent monotonic decreasing pattern. Quantitatively, AUROC and AUPRC reached 0.85 and 0.84, respectively, demonstrating a strong negative correlation. As the affinity value increased, latency sharply decreased, forming a clear downward binned-average trend. This observation indicates that the affinity score effectively captures the sensitive variations in latency performance, serving as a reliable discriminative indicator within the latent space even for transaction-oriented workloads such as TPC-C.

For the TPC-H workload, despite its relatively low structural variability as an analytical query workload, the affinity score showed a meaningful positive correlation with throughput. AUROC and AUPRC were 0.72 and 0.80, respectively, revealing that latent vectors with higher affinity values tended to cluster in regions with higher throughput. The binned trend exhibited a gradually increasing shape, suggesting that the affinity score maintains a consistent level of predictive capability even under workloads with limited structural diversity.

Overall, the proposed affinity score exhibited a strong positive correlation with throughput and a strong negative correlation with latency, demonstrating consistent performance patterns across diverse workloads. These results confirm that



(a) Convergence comparison between VBO and HBO for MySQL.



(b) Convergence comparison between VBO and HBO for PostgreSQL.

Fig. 7: Convergence comparisons on two DBMSs.

the affinity score serves as a reliable indicator of performance proximity within the latent space.

E. Comparison between Vanilla BO and Hybrid-Score-Guided BO

To evaluate the search efficiency of the proposed Hybrid-Score-Guided Bayesian Optimization (HBO), we compared its performance against the conventional Vanilla Bayesian Optimization (VBO). The comparison experiments were conducted on two MySQL workloads (YCSB B and YCSB F) and two PostgreSQL workloads (TPC-C and TPC-H). Each experiment was performed over 300 tuning iterations, and the progression of the best performance score was plotted to visualize the optimization trajectory.

As shown in Figure 7a, for the YCSB B and YCSB F workloads, HBO converges to high performance regions more rapidly than VBO within the first 100 iterations. In particular,for YCSB F, HBO achieves over a 30% improvement in performance around 150 iterations and continues to maintain a consistently higher score subsequently. This demonstrates that by jointly considering both the surrogate model's prediction and the Affinity Score, HBO more effectively guides the search toward already verified high performing regions within the latent space.

Figure 7b shows that TPC-C and TPC-H, HBO consistently outperforms VBO for PostgreSQL. In the case of TPC-C, HBO exhibited a sharp increase in score from the early iterations and converged near the optimum within 100 iterations, whereas VBO showed unstable fluctuations during the initial exploration phase and eventually stagnated at a lower performance level. This result suggests that in transaction oriented workloads, HBO effectively captures performance sensitive regions early and mitigates unnecessary local exploitation. For the TPC-H workload, although the overall score variation was limited due to the analytical nature of the workload, HBO reached high-performing regions faster than VBO and achieved a higher final performance level.

Overall, HBO consistently outperformed VBO in both search efficiency and convergence speed. This improvement can be attributed to the Affinity-Score-guided search, which enables more effective identification of high-performing regions within the latent space. These results demonstrate that HBO goes beyond conventional surrogate-model-based exploration by performing directional optimization that reflects the performance-relevant geometry of the latent space.

TABLE VI: Time cost (minutes) to complete tuning across workloads. Lower is better.

Method	YCSB B	YCSB F	TPC-C	TPC-H
OtterTune	**50.16**	49.70	264.17	345.80
CDBTune	370.15	421.85	462.25	424.61
RGPE	68.46	**34.24**	197.57	224.70
CSAT	242.45	267.53	428.35	467.20
GPTuner	276.20	284.24	305.40	357.50
RelTune	183.13	164.29	**192.61**	**167.16**

F. Analysis of Tuning Overhead and Efficiency

Table VI compares the time cost required by each baseline to complete tuning across different workloads. For the MySQL YCSB workloads, RelTune exhibited a slightly longer tuning time due to the additional GAT training phase on the parameter relationship graph. This initial training step, which encodes parameter dependencies into a latent representation, introduces extra overhead compared to purely search-based approaches. Unlike traditional methods, RelTune requires an initial model training stage to encode these dependencies into a latent representation space. Nevertheless, RelTune achieved the highest performance on both YCSB B and YCSB F workloads, indicating that the initial learning cost is offset by improved exploration efficiency during the tuning process.

In contrast, for the PostgreSQL workloads (TPC-C and TPC-H), RelTune achieved the shortest tuning time among all baselines. This is because performance evaluation through actual benchmark execution is particularly time-consuming in PostgreSQL. Most baseline methods perform real DB restarts and benchmark executions for each BO iteration to measure TPS and latency, resulting in significant overhead throughout the optimization process. RelTune, on the other hand, as shown in line 10 of Algorithm 1, does not execute the actual benchmark in every iteration. Instead, its surrogate model's metric prediction head e_ψ evaluates performance based on the predicted latency in the latent space. By estimating performance through surrogate prediction rather than expensive real executions, RelTune substantially reduces the overall tuning time.

Furthermore, the benchmark execution in PostgreSQL is inherently slower than in MySQL, mainly due to its heav-

TABLE VII: Ablation study results on MySQL workloads.

Module		YCSB A		YCSB B	
RGE	HBO	TPS	Latency	TPS	Latency
✗	✗	3684.76	1426	9968.24	395
✓	✗	4071.56	1146	9967.39	353
✗	✓	3741.25	1169	9972.83	320
✓	✓	**4520.11**	**1028**	**9999.99**	**230**

Module		YCSB E		YCSB F	
RGE	HBO	TPS	Latency	TPS	Latency
✗	✗	3862.43	1421	3357.27	1652
✓	✗	4774.22	1330	4071.56	1146
✗	✓	5319.82	1115	3400.37	1270
✓	✓	**5766.32**	**833**	**4520.11**	**1028**

ier transaction commit and Write-Ahead Logging (WAL) mechanisms, as well as the cluster re-initialization and data reloading steps that benchmark performs at each iteration. Even under such system-level constraints, RelTune effectively minimizes the number of actual executions through surrogate-based evaluation, achieving the highest time efficiency among all methods in the PostgreSQL environment.

G. Ablation Study

Table VII presents the ablation study results evaluating the contributions of the two core components of the proposed RelTune framework: the *Relational Graph Encoder (RGE)* and the *Hybrid-Score-Guided Bayesian Optimization (HBO)*.

In the experiment without the RGE module, the relational graph in the GNN was removed, and a simple MLP encoder was used that takes only configuration–metric pairs as input. Under this setting, each parameter was treated as an independent feature when constructing the latent representation, thereby excluding dependency information among parameters. As a result, the expressiveness of the latent space was degraded, leading to a noticeable decline in both TPS and latency performance. These findings support that relational-aware encoding plays a crucial role in capturing inter-parameter dependencies within database configurations.

In the experiment without the HBO module, the affinity term in the hybrid score was removed, and a vanilla Bayesian Optimization (VBO) was used for optimization. Although the search process remained functional, the absence of affinity-guided exploration resulted in lower performance compared to HBO under the same number of iterations. This outcome supports that the hybrid score enables more efficient exploration by guiding the search toward latent vectors located near high-performing regions, allowing HBO to discover better-performing configurations faster than VBO.

As a result, when both modules were used together, RelTune achieved the highest TPS and the lowest latency across all workloads. These results demonstrate that generating latent vectors that incorporate inter-parameter relationships and optimizing those vectors through HBO work in a complementary manner to enhance overall performance.

V. Conclusion

This study addresses three key limitations in existing database configuration tuning methods: (1) the neglect of inter-parameter relationships, (2) inefficient partial tuning to mitigate high-dimensional search space complexity, and (3) the limited optimization efficiency of conventional search-based approaches. To overcome these issues, we propose RelTune, a framework that models parameter dependencies through a Relational Graph Representation and learns them via a Graph Neural Network (GNN) encoder–decoder architecture. This design produces latent representations that capture semantic and structural interactions among parameters, enabling more consistent and stable tuning behavior. We further introduce the Affinity Score, a quantitative measure of proximity to high-performing regions in the latent space, which shows strong correlation with throughput and latency metrics across workloads. By integrating this metric into Bayesian Optimization, our Hybrid-Score-Guided BO (HBO) achieves faster convergence and superior final performance compared to vanilla BO.

In summary, RelTune unifies relational structure modeling with efficient optimization, achieving high tuning efficiency across diverse DBMSs and workloads, and lays the foundation for future extensions toward real-time adaptive tuning.

Acknowledgment

This research was supported by the National Research Foundation (NRF) funded by the Korean government (MSIT) (No. RS-2023-00229822).

References

[1] X. Zhou, C. Chai, G. Li, and J. Sun, "Database meets artificial intelligence: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 3, pp. 1096–1116, 2020.

[2] ——, "Database meets artificial intelligence: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 3, pp. 1096–1116, 2020.

[3] X. Zhao, X. Zhou, and G. Li, "Automatic database knob tuning: a survey," *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[4] S. Duan, V. Thummala, and S. Babu, "Tuning database configuration parameters with ituned," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 1246–1257, 2009.

[5] A. Fekry, L. Carata, T. Pasquier, A. Rice, and A. Hopper, "Tuneful: An online significance-aware configuration tuner for big data analytics," *arXiv preprint arXiv:2001.08002*, 2020.

[6] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang, "Automatic database management system tuning through large-scale machine learning," in *Proceedings of the 2017 ACM international conference on management of data*, 2017, pp. 1009–1024.

[7] X. Zhang, H. Wu, Z. Chang, S. Jin, J. Tan, F. Li, T. Zhang, and B. Cui, "Restune: Resource oriented tuning boosted by meta-learning for cloud databases," in *Proceedings of the 2021 international conference on management of data*, 2021, pp. 2102–2114.

[8] J. Zhang, Y. Liu, K. Zhou, G. Li, Z. Xiao, B. Cheng, J. Xing, Y. Wang, T. Cheng, L. Liu *et al.*, "An end-to-end automatic cloud database tuning system using deep reinforcement learning," in *Proceedings of the 2019 international conference on management of data*, 2019, pp. 415–432.

[9] G. Li, X. Zhou, S. Li, and B. Gao, "Qtune: A query-aware database tuning system with deep reinforcement learning," *Proceedings of the VLDB Endowment*, vol. 12, no. 12, pp. 2118–2130, 2019.

[10] X. Zhang, Z. Chang, Y. Li, H. Wu, J. Tan, F. Li, and B. Cui, "Facilitating database tuning with hyper-parameter optimization: a comprehensive experimental evaluation," *arXiv preprint arXiv:2110.12654*, 2021.

[11] Y. Li, L. Bao, K. Huang, and C. Wu, "Csat: Configuration structure-aware tuning for highly configurable software systems," *Journal of Systems and Software*, vol. 222, p. 112316, 2025.

[12] J. Lao, Y. Wang, Y. Li, J. Wang, Y. Zhang, Z. Cheng, W. Chen, M. Tang, and J. Wang, "Gptuner: An llm-based database tuning system," *ACM SIGMOD Record*, vol. 54, no. 1, pp. 101–110, 2025.

[13] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, 2012.

[14] M. A. Wiering and M. Van Otterlo, "Reinforcement learning," *Adaptation, learning, and optimization*, vol. 12, no. 3, p. 729, 2012.

[15] *MySQL 5.7 Reference Manual*, MySQL, 2025. [Online]. Available: https://dev.mysql.com/doc/refman/5.7/en/

[16] M. Malu, G. Dasarathy, and A. Spanias, "Bayesian optimization in high-dimensional spaces: A brief survey," in *2021 12th International Conference on Information, Intelligence, Systems & Applications (IISA)*. IEEE, 2021, pp. 1–8.

[17] X. Wang, Y. Jin, S. Schmitt, and M. Olhofer, "Recent advances in bayesian optimization," *ACM Computing Surveys*, vol. 55, no. 13s, pp. 1–36, 2023.

[18] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society Series B: Statistical Methodology*, vol. 58, no. 1, pp. 267–288, 1996.

[19] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *Advances in neural information processing systems*, vol. 30, 2017.

[20] H. Hao, X. Zhang, and A. Zhou, "Model uncertainty in evolutionary optimization and bayesian optimization: A comparative analysis," in *2024 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2024, pp. 1–9.

[21] Y. Li, T. Rudner, and A. Wilson, "A study of bayesian neural network surrogates for bayesian optimization, 2024."

[22] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[23] MySQL, "MySQL v5.7 Documentation," https://dev.mysql.com/doc/refman/5.7/en/, 2024.

[24] PostgreSQL, "PostgreSQL v14.19 Documentation," https://www.postgresql.org/docs/release/14.19/, 2025.

[25] Y. Zhu, J. Liu, M. Guo, Y. Bao, W. Ma, Z. Liu, K. Song, and Y. Yang, "Bestconfig: tapping the performance potential of systems via automatic configuration tuning," in *Proceedings of the 2017 symposium on cloud computing*, 2017, pp. 338–350.

[26] C. Williams and C. Rasmussen, "Gaussian processes for regression," *Advances in neural information processing systems*, vol. 8, 1995.

[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[28] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *Advances in neural information processing systems*, vol. 12, 1999.

[29] GPT4, "GPT Documentation," https://openai.com/ko-KR/index/gpt-4/, 2024.

[30] C. Qin, D. Klabjan, and D. Russo, "Improving the expected improvement algorithm," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[31] P. Auer, "Using confidence bounds for exploitation-exploration trade-offs," *Journal of machine learning research*, vol. 3, no. Nov, pp. 397–422, 2002.

[32] minilm, "minilm," https://huggingface.co/sentence-transformers/paraphrase-MiniLM-L6-v2, 2019.

[33] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019.

[34] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[35] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML workshop on unsupervised and transfer learning*. JMLR Workshop and Conference Proceedings, 2012, pp. 37–49.

[36] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM symposium on Cloud computing*, 2010, pp. 143–154.

[37] tpcc, "tpcc Documentation," https://www.tpc.org/tpcc/, 1992.

[38] tpch, "tpch Documentation," https://www.tpc.org/tpch/, 1992.

[39] benchbase, "benchbase github," https://github.com/cmu-db/benchbase, 2013.

[40] D. E. Difallah, A. Pavlo, C. Curino, and P. Cudre-Mauroux, "Oltp-bench: An extensible testbed for benchmarking relational databases," *Proceedings of the VLDB Endowment*, vol. 7, no. 4, pp. 277–288, 2013.