Libra: One-Shot Parameter Sensitivity Estimation for Transfer Learning in Database Performance Prediction

Tatsuhiro Nakamori
Keio University
Fujisawa, Japan
tatsuhironm@keio.jp

Hideyuki Kawashima
Keio University
Fujisawa, Japan
river@sfc.keio.ac.jp

ABSTRACT

Accurate performance prediction is critical for database tuning, resource provisioning, and performance debugging. Recent work applies machine learning to predict DBMS performance, but these models often require expensive retraining when deployment contexts change. We present Libra, an end-to-end transfer learning framework that builds accurate performance models with minimal target-context sampling. Libra addresses two key challenges: (1) selecting source contexts based on performance-relevant similarity, and (2) leveraging source context data without negative transfer. We introduce a novel context retrieval method based on π-profiles, which capture parameter sensitivity. Libra uses a multilayer perceptron to infer the target π-profile in one-shot, and compares it with those of past contexts to retrieve the most similar one. Libra then selects important parameters based on percentile performance ratios and focuses sampling on high-impact parameters to efficiently train the model. Experiments across 161 contexts (combination of 7 hardware environments and 23 workloads) show that Libra outperforms state-of-the-art methods in terms of sampling efficiency (up to 32× speedup) and prediction accuracy (95.6% error reduction).

PVLDB Reference Format:
Tatsuhiro Nakamori and Hideyuki Kawashima. Libra: One-Shot Parameter Sensitivity Estimation for Transfer Learning in Database Performance Prediction. PVLDB, 19(5): 945 - 957, 2026.
doi:10.14778/3796195.3796207

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Tatzhiro/DBMSTransferLearning/tree/libra.

1 INTRODUCTION

1.1 Background

Performance prediction of database management systems (DBMSs) is a critical yet challenging task for many data-intensive applications. Recently, there has been increasing research on using machine learning (ML) models to predict system performance [11, 34, 37, 47, 51]. Accurate performance models are invaluable for a range of applications, including database configuration tuning [8, 18, 19, 42, 43, 46–50], resource provisioning in cloud environments [13, 20, 30, 31, 35], and performance debugging [36, 45].

Proceedings of the VLDB Endowment, Vol. 19, No. 5 ISSN 2150-8097.
doi:10.14778/3796195.3796207

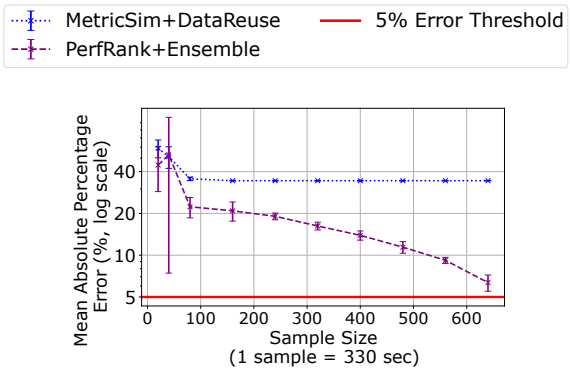


Figure 1: Required Number of Samples to Reduce Performance Model Error – MetricSim+DataReuse [42] and PerfRank+Ensemble [48] suffer from negative transfer, failing to reduce the error below the desired 5% threshold within the 640-sample budget. Each point indicates mean error ± 1 std.

Recent studies have demonstrated that performance prediction is feasible, using features such as cardinality and execution plans to predict latency, and configuration to predict throughput. [8, 10, 12, 25, 26, 33, 44, 52]. These approaches are effective but require extensive data collection and training time for each new deployment. In real-world environments where workloads, hardware, or software configurations change frequently, this becomes a serious limitation. When the deployment context changes in a way the trained model has not encountered, existing models often fail to generalize [16, 29]. As a result, performance data must be recollected and a model must be trained from scratch, incurring significant cost and delay.

To address this challenge, recent work has explored leveraging data collected in past contexts to accelerate model construction in a new target context. This approach, known as **transfer learning** [15–17, 23, 24, 27, 29, 41, 42, 48–50], has emerged as a key solution to construct *accurate* performance models with drastically *fewer* samples in the target context. Effective transfer learning for DBMS consists of (1) retrieving a context similar to the target, and (2) transferring information that reduces the cost of model construction in the target context.

Context Retrieval. The core idea of transfer learning is to apply past data that are likely to be useful in the target context. Transferring data from unrelated contexts not only fails to reduce cost, it can even degrade the accuracy of the resulting model. Therefore, the success of transfer learning largely depends on retrieving a source context that shares similarities with the target context [23, 42, 48, 50]. For example, if the target context is an 88-core machine running a

read-only workload, the source context of a 30-core machine with a read-heavy workload might be more similar than that of a 4-core machine processing a write-only workload.

Data Transfer. After similar contexts are identified, a transfer algorithm must extract useful data from them to aid learning in the target context. Existing studies leverage past data either by augmenting training set with source context data [17, 29, 41] or by extracting parameters that have significant performance impact to reduce the target sampling space [16].

1.2 Challenge

Although various approaches of these two components have been presented [16, 17, 41, 42, 48], we have observed two challenges they face that hinder the effectiveness of transfer learning.

Challenge 1. Identifying the most similar context based on performance-related criteria with few samples. Existing studies fail to achieve comparison of contexts based on criteria that are helpful for transfer learning. Previous transfer learning systems have used the proximity of system-level metrics between the target and each past context as the criterion [42, 49]. These metrics capture system state (e.g., CPU usage) but do not reflect how performance responds to parameter changes, providing no guarantee that the selected source context improves target prediction.

Rather than raw system metrics, we need to compare *performance-related characteristics* to select contexts. Ideally, we should compare the performance surfaces between contexts [29], where performance surface is the function from parameters to performance. However, existing implementations of this idea [48, 50] require a large number of target samples to accurately identify similar contexts, defeating the goal of reducing sampling cost. The challenge, then, is to infer *performance-level similarity* with *fewer* samples.

Challenge 2. Leveraging past data without negative transfer. *Negative transfer* is a phenomenon where transfer learning ends up hindering the accuracy of performance models [16, 23]. As shown in Figure 1, existing approaches suffer from negative transfer, failing to reduce prediction error below the desired threshold. For transfer learning to be effective, transfer algorithms must avoid negative transfer when leveraging past information to decrease the required target samples. Algorithms that involve data augmentation by including source context data in target model training are vulnerable to negative transfer, as they cannot account for mismatches in performance-surface trends between source and target contexts. Prior studies [29, 42, 50] show that these trends vary widely across workloads and hardware environments. Unless we can find a source context that exhibits nearly same surface trend as the target, which is not possible without large amount of target data, data augmentation is likely to cause negative transfer. Nevertheless, existing studies rely on data augmentation [17, 23, 41, 42, 48, 49].

1.3 Approach

To overcome these challenges, we propose Libra, a performance prediction framework that integrates context retrieval and transfer algorithm in a synergistic manner.

Approach 1. One-shot Parameter Sensitivity Estimation. Unlike existing approaches that rely on raw metrics or extensive sampling, our method identifies source contexts using *just one target sample* based on the performance impact of parameters. We introduce the concept of parameter importance profile, or π*-profile*, a normalized vector quantifying each parameter's relative influence on performance (e.g., throughput, latency). We calculate the distance between π-profiles to find contexts that share sensitivity to the same critical parameters, enabling comparison based on performance-relevant criteria. However, directly calculating the π-profile for the target context remains infeasible. Like the construction of performance surfaces in conventional systems, computing the π-profile also requires extensive data collection (e.g., over 72 hours in our experiments). Libra bypasses this problem by providing a **one-shot parameter sensitivity estimation** via MLP (multilayer perceptron) trained on a diverse set of previously observed workload and hardware contexts. Given a single runtime snapshot of OS and DBMS metrics, the MLP infers the target's π-profile. Existing methods only conduct one-to-one context comparisons, preventing previously acquired data to offer insights into performance-related similarity. Libra leverages all past data to identify performance-relevant similarities with just one target sample.

Approach 2. Reducing Target Sampling Space. Instead of data augmentation, which risks negative transfer, Libra only uses the source context data to select important parameters and shrink target sampling space. Important parameters are those that have significant impact on DBMS performance. Sampling all parameters results in wasted effort, as many samples vary only in unimportant parameters, offering little insight into performance behavior. To address this inefficiency, Libra identifies important parameters using source context data and conducts focused sampling on them, enabling accurate model construction with fewer samples. Libra provides important parameter selection that is less prone to false-negatives by detecting any non-linear effects and remaining robust to outliers. It measures each parameter's influence using the 99th-to-1st percentile performance ratio and retains those exceeding a predefined threshold. The selected parameters, ordered in terms of their relative impact, define a sampling distribution that shrinks the search space (97% reduction in Fig. 6b) and biases sampling toward the most influential parameters, accelerating the model convergence.

We implemented Libra as well as state-of-the-art methods and compared how fast and accurately they can build performance models. We collected MySQL data from 161 contexts to offer simulation of transfer learning between diverse contexts. While existing approaches suffer negative transfer or require many samples, Libra consistently achieves prediction error below 5% consistently with up to 32× speedup in sampling efficiency. We release our code and data publicly available [5]. The codebase offers a pluggable interface, enabling users to extend context retrieval and data transfer modules for new algorithm development and comparison.

1.4 Organization

The rest of the paper is organized as follows. Section 2 explains preliminaries and the objective of this study. In Section 3, we provide the overview of Libra as well as its workflow. Section 4 describes the proposed context retrieval module while Section 5 explains the parameter selection in the data transfer module. Section 6 is the evaluation of Libra compared against state-of-the-art methods. Section 7 describes related work and Section 8 concludes this paper.

Table 1: Terminology Used in this Paper

Term	Definition
Context	A combination of factors that influence performance but are not part of the DBMS parameters, including hardware specifications (e.g., # of CPU cores, memory size), workload properties (e.g., read/write ratio), and software environments (e.g., DBMS version, OS). We denote a context as a pair of hardware and workload in Section 6 (e.g., (`machineA, workloadX`)).
Target context	The context of interest, where we want to build a performance model but no data is available.
Source context	A previously observed context that can be leveraged to improve learning in the target context. We assume that data are readily available from all source contexts. Libra leverages data from all source contexts to train the context retrieval MLP, while it selects and uses a single source context that is most similar to the target context for data transfer.
Parameters	Tunable knobs exposed by the DBMS that directly influence its runtime behavior and performance.
Configuration	A set of specific values assigned to all parameters.
Performance model	A machine learning model that predicts the performance given a specific configuration. We refer to this as the model.
Sample	A pair of DBMS configuration and its corresponding performance measurement. To obtain one sample, we set up DBMS with a specific configuration, execute a workload, and measure the resulting performance. These samples are used as training data.
Negative Transfer	A phenomenon in which transfer learning degrades the accuracy of the models. We regard prediction error greater than 5% as negative transfer.
Important Parameters	Parameters that have significant impact on performance.
Context Vector	A vector of runtime metrics measured under the default DBMS configuration in a given context. e.g., {CPU utilization: 20%, Memory usage: 80%, ... }.

2 PRELIMINARIES

This section provides the necessary background for our study. We define key terms, review relevant transfer learning systems, and formally state the problem setting. Table 1 summarizes the terminology used throughout this paper.

2.1 Transfer Learning for Configurable Systems

Transfer learning consists of (1) identifying the most similar context (*context retrieval*), and (2) transferring useful data for model building (*transfer algorithms*). Existing work has proposed various transfer algorithms [16, 17, 29, 41]. A common approach, *DataReuse* [17], augments model training with source-context data. When contexts differ only by hardware configuration, linear regression models can transform source data to fit the target context better [29, 41], an approach known as *ModelShift* [16, 24, 29, 41]. For general cases where there are workload and hardware variations between contexts, *L2S* [16] identifies parameters with strong performance impact in the source context to guide target sampling, assuming important parameters are generally similar across contexts [15].

While performance prediction related research have focused on proposing transfer algorithms, they have excluded (1) context retrieval in their studies. Transfer learning methods adopted in DBMS tuning systems OtterTune and ResTune have incorporated both the (1) context retrieval and (2) transfer algorithm to build models for recommending optimal configurations in unseen contexts [42, 48]. *MetricSim+DataReuse* in OtterTune calculates the Euclidean distance of system metrics to map target workload to the most similar past workload and constructs the target model by using DataReuse. *PerfRank+Ensemble* in ResTune computes context similarity by comparing the performance rankings of configurations between the target and each source context. For the performance prediction, it uses a weighted ensemble of the target and source models, where the weights are determined by the similarity scores. The original paper [48] demonstrates PerfRank+Ensemble's effectiveness when either hardware or workload varies.

Table 2: Libra and Existing Transfer Learning Techniques - This table shows the transfer learning components and the scope of context each technique targets. Libra implements both context retrieval and transfer algorithm to support transfer learning under simultaneous variations in workload and hardware.

	Context Scope	
Transfer Learning Components	Workload & Hardware	Workload or Hardware
Context Retrieval & Transfer Algorithm	Libra	MetricSim+DataReuse [42] PerfRank+Ensemble [48]
Context Retrieval or Transfer Algorithm	L2S [16]	ModelShift [41] DataReuse [17]

We target a transfer learning approach that can generalize to contexts where both hardware and workload vary *simultaneously*. These two factors fluctuate frequently in practical settings [29, 42], but prior work that explicitly evaluates transfer learning under simultaneous variation is limited to L2S [16]. We propose both context retrieval and transfer algorithm to enable end-to-end transfer learning for building DBMS performance models. As illustrated in Table 2, our approach is unique in that it supports transfer learning across contexts where workload and hardware vary simultaneously while also implementing both (1) and (2).

2.2 Problem Formulation

In principle, increasing the number of samples improves model accuracy, but collecting samples is costly and time-consuming. Our objective is to minimize the number of samples drawn from the target context while ensuring that the resulting model maintains high predictive accuracy. The cost of learning in a target context is measured by the number of samples collected from that context. Sampling from source contexts is not included in the cost, as we assume they are part of a historical dataset for which data has already been collected. We express this objective formally as follows:

Let $\mathcal{X} \subseteq \mathcal{P}_1 \times \mathcal{P}_2 \times \cdots \times \mathcal{P}_m$ denote the set of all possible configurations in the target context that a DBMS user is concerned with, where $\mathcal{P}_i$ is the domain of the i-th parameter, either continuous or categorical, and m is the total number of parameters. Let $\mathcal{D}_{\text{target}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n} \subset \mathcal{X} \times \mathbb{R}$ be the set of samples collected from the target context, where $\mathbf{x}_i \in \mathcal{X}$ is a configuration and $y_i \in \mathbb{R}$ is the corresponding measured performance (e.g., throughput). Let $\mathcal{M}(\mathcal{D}_{\text{target}})$ denote the model trained on $\mathcal{D}_{\text{target}}$. We define $\mathcal{L}(\mathcal{M}(\mathcal{D}_{\text{target}}), \mathcal{X})$ as the prediction error of the model $\mathcal{M}(\mathcal{D}_{\text{target}})$ evaluated over the configuration space $\mathcal{X}$, and let ϵ be a predefined error threshold. Our objective is to

$$\min |\mathcal{D}_{\text{target}}| \quad \text{subject to} \quad \mathcal{L}(\mathcal{M}(\mathcal{D}_{\text{target}}), \mathcal{X}) \leq \epsilon. \tag{1}$$

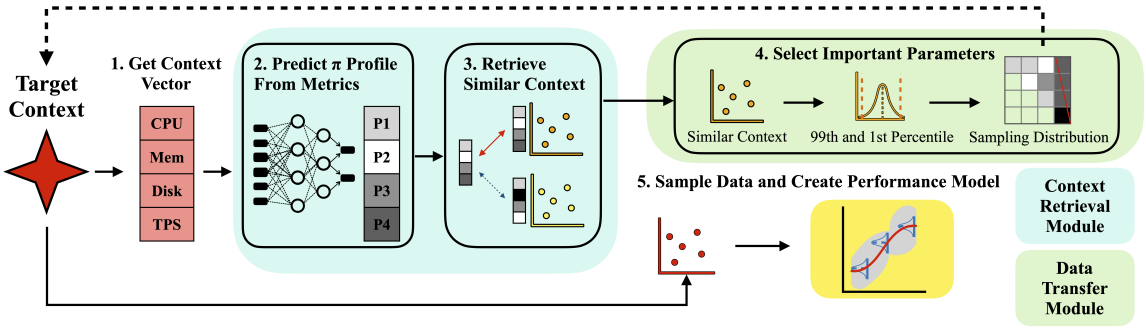


Figure 2: Overview of Libra

3 SYSTEM OVERVIEW

To achieve the objective in Equation 1, we propose *Libra*, an end-to-end transfer learning system that identifies contexts similar to a given target based on parameter sensitivity and conducts focused sampling of important parameters to construct accurate performance models. The system is composed of two primary modules: the **context retrieval module** and the **data transfer module**.

3.1 Workflow

Figure 2 illustrates the overview of Libra and the 5 steps of transfer learning. Transfer learning begins by identifying the most similar context. Libra's context retrieval takes one sample of system metrics from the target to use as its context vector (Step 1). The context vector serves as an input to predict the target's ***π*-profile** (Section 4.1), a normalized vector that quantifies the performance impact of each parameter (Step 2). To enable the one-shot π-profile prediction, Libra learns to map context vectors to π-profile based on data collected from previously observed contexts (Section 4.2). The predicted π-profile is then compared by KL-divergence against the π-profiles of past contexts to identify the one most similar to the target (Step 3). Comparing π-profiles offers a significant advantage over comparing raw metrics, since similarity in parameter sensitivity is a more reliable indicator of transferability (Section 4.1).

Once the most similar source context is selected, the data transfer module analyzes its data to identify important parameters and construct target sampling distribution (Step 4). The sampling distribution prioritizes parameters with higher performance impact, while it completely disregards the sampling of unimportant parameters to reduce the sampling space. Reducing the sampling space is critical, as it is infeasible to explore the full configuration space of a DBMS with hundreds of parameters. Using the selected subset, the module then begins sampling from the target context. The collected samples are used as training data to construct the model (Step 5).

3.2 Assumptions and Limitations

Libra requires that all parameters of interest be observed in each source context used for transfer learning. If no such context exists, we need to collect data from diverse contexts. Although it is infeasible to experiment different parameter settings on production environments that support real services, this data collection can be done on low-cost testing environments. For example, in our experiments, we collected data from inexpensive cloud instances provisioned for benchmarking. The hardware gap between these low-cost test environments and a target context with more powerful hardware does not hinder Libra's performance, as its transfer learning mechanism bridges such disparities effectively.

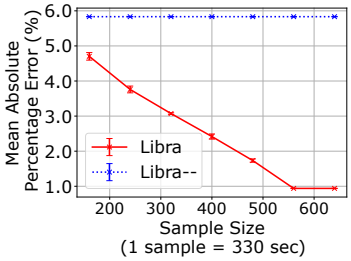


Figure 3: Performance Prediction Accuracy of Libra for TPC-C Workload – The baseline is able to select from all Sysbench [1] workloads listed in Table 5, while the blue dotted line represents a case where only the read-only workload contexts are available.

When collecting data, we need to define the appropriate parameter ranges and granularities. This process cannot be fully automated, as suitable ranges and levels of granularity are dependent on application-specific requirements. Therefore, Libra requires application developers or database administrators (DBAs) to be involved in making such decisions.

Finally, as with all transfer learning methods, Libra assumes that there exists one previously observed context that is sufficiently similar to the target to enable effective transfer learning. When that assumption fails, negative transfer can occur. Libra−− in Figure 3 shows the result of transfer learning for TPC-C workload target when we only have source contexts of read-only workloads. Because none of the read-only workload contexts are similar to the target, transfer learning results in a negative transfer. This problem can be mitigated by collecting data on a diverse set of simple benchmark workloads. Libra in Figure 3 uses all Sysbench workload variants listed in Table 5. Even for a more complex workload like TPC-C, such simple-benchmark data are sufficient to avoid negative transfer and enable effective transfer.

Practical Use Case. For example, Libra is suited for organizations that have cloud infrastructure and DBAs to support both production

services and internal testing. These organizations can use low-cost virtual machines to benchmark various hardware–workload combinations without impacting production. With enough data from diverse contexts, they can apply Libra to build accurate performance models for new target contexts with less samples.

4 CONTEXT RETRIEVAL MODULE

4.1 Parameter Importance Profile

If hardware were the sole factor defining a context, one might expect that similarity in system specifications directly translates to similarity between contexts. However, as additional factors, such as workload, are introduced, it becomes increasingly difficult to quantify context similarity using such a simple measure. It is unclear how much each factor (hardware environments versus workload characteristics) contributes to overall context similarity.

In the context of transfer learning, parameter sensitivity offers a meaningful indicator of context similarity. Intuitively, transfer learning is effective between contexts that share the same important parameters. Suppose we have a target context X and 2 candidate source contexts A and B. The performance in context X is sensitive to parameters $\{o, p, q\}$, whereas A is sensitive to $\{p, q\}$, and B to $\{r\}$. In such a case, we can learn about the parameters p and q by leveraging data from A and transferring the knowledge to X, but B does not have relevant knowledge to offer for X. Previous studies [16, 23, 42, 49] have not integrated this intuition that contexts with shared important parameters are similar. Some systems measure the similarity by the Euclidean distance between system metrics [42, 49], and some select a source context by testing out how generalizable each context is within the past contexts [21–23].

These similarity definitions do not necessarily reflect how effective transfer learning will be. Euclidean distance assumes all system metrics contribute equally to context similarity, which undermines performance-relevant features. Selecting source contexts solely based on past generalizability offers no guarantee of effectiveness on a new, unseen target context. Existing approaches ignore whether the source and target contexts share the same important parameters. There needs to be a metric that can measure the similarity in terms of parameter sensitivity.

To address this problem, we define parameter importance profile, or *π-profile*, a metric that captures the information about a context's sensitivity to individual parameters in relation to others.

$$\boldsymbol{\pi} = \frac{\boldsymbol{p}}{\|\boldsymbol{p}\|}, \quad \text{where} \quad \boldsymbol{p} = \left[p_1, p_2, \ldots, p_m\right]^\intercal \tag{2}$$

$\boldsymbol{\pi}$ in Equation 2 denotes a π-profile, represented as a normalized vector in R^m, where m is the total number of parameters being considered for transfer learning (we selected parameters listed in Table 3 based on expert DBA recommendations). Each element p_i indicates the contribution of a parameter to database throughput. To compute p_i, we measure the range of throughput values observed when varying that parameter, while other parameters are held to their default values (shown in Table 3).

Since we assume that past context data are already collected, we can easily obtain the π-profile for source contexts by simply calculating the range for all parameters. Even if we were to construct a context's π-profile from scratch, it is still feasible as the cost only increases linearly with respect to the number of parameters: π-profile only takes the independent effect of one parameter as its element. For 12 parameters shown in Table 3, π-profile for one context can be prepared in as fast as 3 hours.

The distance between π-profiles reflects the similarity of performance characteristics. Consequently, the selected context is more likely to provide insights about performance in the target context, enabling more effective transfer learning.

4.2 One-Shot Prediction

Unlike source contexts, which have sufficient data to compute their π-profiles, the target context lacks enough samples at the start of transfer learning. While the π-profile is useful for identifying similar contexts, obtaining it requires substantial data. This contradicts the goal of transfer learning to minimize the number of samples needed from the target. To overcome this challenge, we train an MLP that can predict the π-profile of a target context in one-shot. MLPs are well-suited for modeling π-profile (normalized vector) because they can jointly model interdependencies between output dimensions and enforce normalization constraints via softmax.

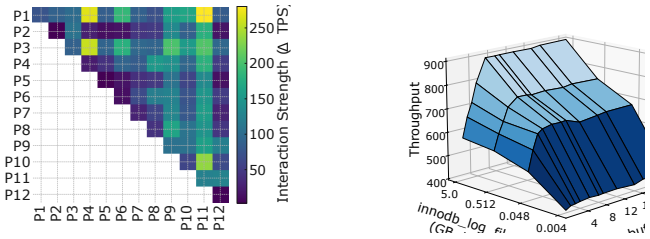
Specifically, we use a standard multilayer perceptron (MLP), which takes as input a context vector (a vector of runtime metrics measured under default DBMS configuration in a context), and outputs an estimation of the true π-profile. The MLP learns to map raw system metrics, represented by a context vector, to π-profile, a meaningful representation that captures a context's parameter sensitivity. The metrics in the context vector are collected under a fixed database parameter configuration. In our experiments, we obtain OS and DBMS metrics using Prometheus [4] while MySQL is configured to our default parameter values as shown in Table 3. Table 6 shows the 9 metrics used as the elements of context vectors.

Formally, let $\boldsymbol{c} \in \mathbb{R}^d$ denote the input context vector, composed of runtime metrics, and let $\boldsymbol{f_\theta} : \mathbb{R}^d \to \mathbb{R}^m$ denote the MLP with parameters $\boldsymbol{\theta}$ ($d = 9, m = 12$ in our experiments). The MLP outputs an unnormalized vector $\hat{\boldsymbol{p}} = \boldsymbol{f_\theta}(\boldsymbol{c}) \in \mathbb{R}^m$, which is then normalized to form the π-profile via the softmax function [6]. This normalization ensures that $\hat{\boldsymbol{\pi}}$ represents a π-profile, where each value reflects the relative parameter sensitivity, and all elements sum to one.

To train the MLP, we prepare a dataset comprising diverse contexts, formed by combining various hardware platforms and workload types. Each row of the dataset consists of a context vector paired with its corresponding π-profile. As explained in Section 4.1, π-profiles of historical contexts can be readily obtained. We pair π-profile with multiple instances of context vectors to construct training samples for one context. Our training dataset consists of 1,650 samples collected in 161 distinct contexts (about 11 samples per context). During training, the MLP learns to associate context vectors with corresponding π-profiles by minimizing the Kullback-Leibler (KL) divergence between the predicted π-profile $\hat{\boldsymbol{\pi}}$ and the ground-truth π-profile $\boldsymbol{\pi}$:

$$\mathcal{L}_{\mathrm{KL}}(\hat{\boldsymbol{\pi}} \| \boldsymbol{\pi}) = \sum_{i=1}^{m} \hat{p}_i \log\left(\frac{\hat{p}_i}{p_i}\right). \tag{3}$$

The KL divergence loss $\mathcal{L}_{\mathrm{KL}}$ in Equation 3 encourages the MLP to capture the relative parameter contributions rather than exact magnitudes, aligning with the normalized nature of π-profiles.



(a) Heatmap of parameter interaction

(b) Interaction of log file size and buffer pool size

Figure 4: Interaction of Parameters

4.3 Identifying the Most Similar Context

Once the MLP is trained, we use it to predict the π-profile of the target context. For each target context, we obtain a context vector constructed from a single snapshot of the runtime metrics. Because we train the MLP with a single input of a context vector, we can predict the π-profile of the target in one-shot. Finally, we compute the KL divergence between $\hat{\boldsymbol{\pi}}$ and $\boldsymbol{\pi}$ of the candidate contexts. The context that outputs the lowest KL divergence is selected as the most similar context. The inference of a π-profile takes just few seconds. The computation of the KL divergence is also low cost, taking less than a second. The computational complexity of retrieving the most similar context is $O(N)$, where N is the number of past contexts (161 contexts in our experiments).

5 DATA TRANSFER MODULE

5.1 Selecting Important Parameters to Sample

Libra's data transfer module conducts transfer learning by sampling only from the important parameters. Source context data are used to identify the important parameters. As data-augmenting transfer algorithms are prone to negative transfer [16, 29], the data transfer module excludes source context data from training.

Although reducing the target sampling space can accelerate building models, it can also lead to negative transfer when we have false negatives. To avoid this, we need a selection method that accurately identifies all impactful parameters.

Existing transfer learning systems are susceptible to false negatives, as they use linear regression based methods to select important parameters [15, 16, 29, 36, 42]. Linear regression based approaches assume that each parameter contributes to system performance through a fixed, linear relationship, implying that changes in a parameter always result in proportionate changes in the output. However, this assumption rarely holds in practice. Many database parameters exhibit nonlinear behaviors such as saturation effects, thresholds, or non-monotonic trends. For example, increasing log file size improves throughput from 4MB to 48MB, but further increases have little to no performance impact, as shown in Fig. 9a. When such nonlinear behavior is present, linear models may assign low or zero importance to parameters that actually have substantial impact (i.e., false negative), because the effect does not follow a straight line. Furthermore, outliers can significantly affect the result of linear regression based methods [6].

Algorithm 1 Selecting Important Parameters via Interaction-Aware Profiling

```
function SelectImportantParameters
    X ← ∅,    M ← ProfileSensitivity
    for each parameter p ∈ {1, ..., m} do
        if all entries in row p and column p of M are zero then
            continue
        X ← X ∪ {p}
    return X

function ProfileSensitivity
    Initialize matrix M ∈ ℝ^{m×m} with zeros
    for each parameter p ∈ {1, ..., m} do
        for each parameter q ∈ {p, ..., m} do
            f ← PerformanceSurface(p, q)
            (f_low, f_high) ← Get99thMinMax(f)
            if f_high / f_low ≤ τ then continue
            if not IsConditionallyImportant(p, q) or
            not IsConditionallyImportant(q, p) then
                continue
            M[p][q] ← f_high / f_low
    Normalize matrix M
    return M

function IsConditionallyImportant(p, q)
    for each value v in UniqueValues(p) do
        f ← PerformanceSurface(q | p = v)
        (f_low, f_high) ← Get99thMinMax(f)
        if f_high / f_low > τ then return True
    return False
```

To capture non-linear effects and remain robust to outliers, we quantify the influence of parameters as the ratio of the 99th and 1st percentile performance. Similar to how we calculate π-profile, we extract the data where the parameter of interest is varied while other parameters are set to a certain value. We compute their ratio and compare it with a predefined threshold to decide whether a parameter is important or not. This ratio reflects the factor by which performance varies as the parameter changes, capturing the effects of the parameters regardless of whether the relationship is linear or non-linear. Taking percentile performance instead of absolute max and min also makes this method robust to outliers and noise.

We cannot use π-profile to select parameters because it does not measure the importance of each parameter individually. A parameter with substantial impact may appear relatively unimportant in the π-profile if other parameters have an even larger influence on performance. Furthermore, unlike the π-profile which uses the absolute range, we calculate the ratio to assess importance. Setting a standard predefined threshold for absolute ranges is difficult because meaningful performance differences varies across contexts. By using ratio of the 99th and 1st percentile performance, we can define uniform criteria of important parameters for all contexts.

In order to construct accurate performance models, we also need to consider interaction of parameters because joint effects of parameters sometimes lead to significant performance shifts. Figure 4a shows that the interaction of P1 (buffer pool size) and P11 (log file size) has the largest performance impact among all parameters.

Specifically, Figure 4b illustrates that the performance impact of the buffer pool size changes depending on the log file size. When the log file size is set to 4MB (0.004GB), the buffer pool size has only a minor effect, with throughput varying by just 28 units (from 405 to 432). In contrast, when the log file size is 5GB, increasing the buffer pool size leads to a much larger impact, with throughput ranging by 327 units (from 575 to 902). This example demonstrates that a parameter may appear unimportant in isolation but can have a substantial impact through interactions with other parameters. Such conditionally important parameters must be included in the target sampling space to ensure that the resulting model can accurately predict their joint effects and avoid negative transfer. Our parameter selection method addresses this by analyzing not only their marginal effects but also their interaction effects, ensuring that conditionally important parameters are not mistakenly excluded.

The process for selecting important parameters is described in Algorithm 1. For each pair of DBMS parameters, the algorithm extracts a slice of the performance surface from the similar context, focusing on configurations where only those two parameters were varied (lines 10–12). The performance surface represents the function from parameter values to system performance. From this surface, the 99th and 1st percentiles of the performance are computed (line 13). If the ratio between these percentiles exceeds a predefined threshold (line 14), the parameter or parameter pair is considered potentially impactful.

For single-parameter sensitivity, this ratio alone is sufficient to determine importance. However, for parameter interactions, the algorithm includes an additional check using the IsConditionallyImportant function (line 21). Specifically, it verifies whether each parameter is conditionally important with respect to the other (lines 21–26). This check removes unimportant parameters that appear to have meaningful interaction only because the other parameter has a large effect on performance (line 17). If both parameters are conditionally important, it means that the parameter pair amplifies each other's impact, indicating a meaningful interaction. The performance ratio of their interaction is then recorded in the sensitivity matrix (line 18). Otherwise, the matrix entry remains zero, indicating limited or redundant interaction effects.

After processing all parameter pairs, the matrix is normalized to produce a relative importance scale (line 19) that is used to define a sampling distribution over parameters for the target context. This ensures that parameters or interactions with higher influence are sampled more frequently. Unimportant parameters are disregarded from the sampling distribution to reduce unnecessary exploration.

5.2 Performance Model

The collected samples are used as training data for the performance model. We adopt Gaussian Process (GP) regression [32] in Libra, as it offers several advantages for modeling DBMS performance. First, GP regression can model arbitrary smooth functions, making it well-suited for capturing the complex, non-linear relationships between DBMS parameters and performance. Second, it is widely used in the literature for performance prediction and in self-tuning systems, aligning our approach with established practices [16, 17, 23, 29, 36, 42, 48, 49]. Third, GP regression naturally provides a confidence interval for its predictions, which is useful for guiding exploration in settings with limited data.

Most importantly, GP regression is the de facto model used in Bayesian optimization [39], a proven approach for automatic DBMS tuning [8, 42, 49]. By using a GP model, we ensure that the resulting model is directly compatible with existing tuning methods.

Table 3: MySQL Parameters Selected based on Expert DBA Recommendations – Range: the ranges of values used in the experiments. Default: the default values used for the experiments.

ID	Parameter	Range	Default
1	innodb_buffer_pool_size	1GB-24GB	1GB
2	innodb_read_io_threads	1-24	2
3	innodb_write_io_threads	1-24	2
4	innodb_flush_log_at_trx_commit	0-2	1
5	innodb_adaptive_hash_index	ON, OFF	ON
6	sync_binlog	0, 1	1
7	innodb_lru_scan_depth	100-10000	1024
8	innodb_buffer_pool_instances	1-8	1
9	innodb_change_buffer_max_size	0-50	25
10	innodb_io_capacity	100-20000	1
11	innodb_log_file_size	4MB-5GB	48MB
12	table_open_cache	1-4000	4000

Table 4: Hardware Environments – ID: name of the environment, #Cores: number of CPU cores, RAM Size: size of main memory, Type: cloud instance (C) or physical machine (PM).

ID	#Cores	RAM Size	CPU	Disk	Type
small	4	6 GB	AMD EPYC 7551P	VirtIO SSD	C
medium	8	12 GB	AMD EPYC 7551P	VirtIO SSD	C
large	12	16 GB	AMD EPYC 7551P	VirtIO SSD	C
xlarge	16	24 GB	AMD EPYC 7551P	VirtIO SSD	C
xxlarge	24	32 GB	AMD EPYC 7551P	VirtIO SSD	C
xxxlarge	32	64 GB	AMD EPYC 7551P	VirtIO SSD	C
prod	88	190 GB	Xeon Gold 6238L	SSD	PM

6 EVALUATION

6.1 Experimental Setup

We conducted all the experiments using MySQL version 8.0.32, while employing different hardware environments and workloads.
MySQL [2] is a relational database management system widely used for its diverse capabilities. MySQL has approximately 190 parameters to precisely control all these layers and offer adaptability [47]. Not all of them are important in industrial settings, however. Based on the experience of experts from the industry, we selected 12 parameters to use in the experiments, shown in Table 3.
Hardware Environments. Table 4 shows the list of hardware environments used in our experiments. 6 of the hardware environments were prepared as cloud instances. Using these as target contexts simulates a scenario in which users initialize a new cloud instance that hosts a DBMS. We also included a physical machine that largely exceeds the hardware capabilities of the cloud instances. This machine was used to test the capability of Libra to generalize to a new context vastly different from the collected contexts. Use of this machine also replicates a scenario in which developers are

Table 5: Workloads in the Experiment – Sysbench: 7 read-write ratios × 3 skew levels = 21 workloads. TPC-C: 2 warehouse settings.

Workload Type	Read-Write Ratio	Skew
Read-only	100:0	0.2, 0.6, 1.0
Read-write	95:5	0.2, 0.6, 1.0
Read-write	80:20	0.2, 0.6, 1.0
Read-write	50:50	0.2, 0.6, 1.0
Read-write	20:80	0.2, 0.6, 1.0
Read-write	5:95	0.2, 0.6, 1.0
Write-only	0:100	0.2, 0.6, 1.0
TPC-C	–	10, 100 warehouses

interested in the performance of DBMS for newly developed application in a large production environment. All of the environments used CentOS 7 as their operating systems.

Workload. We employed Sysbench [1] to simulate a variety of workloads. Sysbench is a multi-threaded benchmark tool commonly used for database benchmarks [47, 48, 50]. We ran Sysbench for varying read-write ratio and skewness, ranging from 0% to 100% reads and 0.2 to 1.0, respectively. Read-write workloads are denoted as `RW_readratio-skew` (e.g., `RW_50-0.2`). For read-only and write-only workloads, we use the formats `RO-skew` and `WO-skew`, respectively. We also ran TPC-C like workloads developed by Percona Labs [3] with warehouse 10 and 100. This workload differs by not using random text fields and allowing multiple table sets, but otherwise follows the TPC-C standard benchmark, which simulates an order processing application[7, 42]. We will refer to this workload as TPC-C in the evaluation, using the notation `TPCC-numWarehouse` (e.g., `TPCC-10`). In total, we prepared 23 distinct workloads. Table 5 summarizes the workloads used in the experiments.

Evaluation Criteria. We first evaluate the accuracy of predicting π-profiles in Sectiton 6.4. We use KL divergence to measure the similarity between the predicted and ground-truth π-profiles, as well as between the source and target π-profiles.

Second part of our evaluation tests the effect of (1) selecting source contexts and (2) the choice of transfer algorithms on the number of samples required to build the model with a certain accuracy. Finally, we evaluate the performance of end-to-end transfer learning. We use mean absolute percentage error (MAPE) to evaluate the accuracy of the final model. MAPE is a scale-independent metric for evaluating the accuracy of performance models[14, 16, 41]. We run each experiment five times and calculate the average MAPE, with error bars in figures indicating ± 1 standard deviation.

6.2 Data Collection

To simulate transfer learning, we collected data from contexts that are the combinations of 7 hardware environments and 23 workloads, totaling 161 distinct contexts. We collected data by running the 23 workloads one by one on each of the 7 hardware environments. Before running the Sysbench workloads, we initialized a database with 64 tables of 1,000,000 entries. Subsequent workload runs used the prepared database repeatedly. We initialized a separate database for the TPC-C workloads with 4 table sets.

Before each run, we set up the DBMS with different configuration. Because it is infeasible to collect data from the entire parameter space, we focused up to the interaction of two parameters in our experiments. Specifically, we only changed two parameters at one time, while the rest were set to their default values. Previous study involving SQLite found that majority of pairwise interactions have negligible effect on performance, which suggests that spending a lot of time on further interactions is not worth the cost [15]. After setting the parameters, we conducted 30 seconds of warm up run to minimize the effect of cold cache. Finally, we ran the benchmark for 90 seconds for Sysbench workloads, and 300 seconds for TPC-C like workloads to collect one sample of throughput and other 124 system metrics (e.g. disk IOPS, number of read queries).

For each context, we collected around 1,000 samples. The total duration of the data collection adds up to 7,000 machine hours.

Table 6: List of 12 Input Features Used in the Context Retrieval MLP – Source: origin of the metric (DBMS or OS).

Source	Input Features
DBMS	Transactions per second; InnoDB buffer pool cache hit rate; Dirty buffer pages count; Number of queries (per 60s); Rows deleted/inserted/read/updated (per 60s)
OS	Average memory usage (%); Max CPU usage (%); Average read/write disk IOPS (60s interval)

6.3 MLP Training

To demonstrate that the context retrieval module generalizes to unseen workloads and hardware environments, we trained a separate MLP for each target context. Unless otherwise specified, the training data excluded any samples that shared the same workload or hardware as the target context. If the target context was `(prod, TPCC-10)`, then the training data excludes any contexts involving the `prod` hardware or the `TPCC-10` workload.

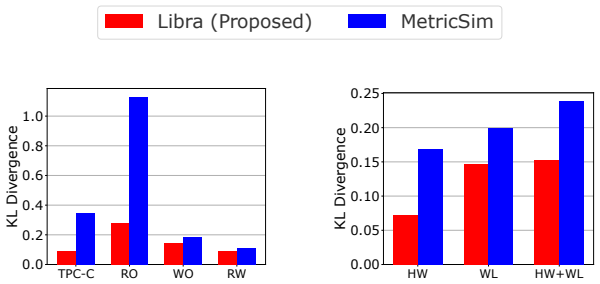
The training dataset comprises 1,650 samples, each representing data collected under varying workload and hardware contexts. For each context, we included about 11 instances of context vectors under default DBMS configuration. Compared to standard benchmark datasets commonly used in regression tasks, such as the California Housing dataset which contains 20,640 instances, our dataset is considerably smaller in scale. Despite the limited data volume, our MLP demonstrates generalization to unseen contexts, highlighting the effectiveness of the context retrieval module (Section 6.4).

To improve generalization, we conducted experiments to identify metrics that were irrelevant to predicting π-profiles. Out of the 125 metrics collected, we selected 12 metrics to use as the features in the MLP (Table 6). Furthermore, we incorporated an L2 regularization term to the KL divergence loss function to prevent overfitting:

$$\mathcal{L}_{\text{KL+L2}}(\hat{\boldsymbol{\pi}} \| \boldsymbol{\pi}; \boldsymbol{\theta}) = \sum_{i=1}^{m} \hat{p}_i \log\left(\frac{\hat{p}_i}{p_i}\right) + \lambda \|\boldsymbol{\theta}\|_2^2 \tag{4}$$

where $\|\boldsymbol{\theta}\|_2^2$ represents the squared L2 norm of all MLP weights, and λ is a hyperparameter that balances the trade-off between minimizing the KL divergence and constraining model complexity. Regularization is a widely used technique in machine learning to improve model generalization by discouraging overly complex models that may overfit to the training data [6, 28].

6.4 Context Retrieval Module Evaluation



(a) Context Retrieval for `prod` grouped by TPC-C and Sysbench: Read-only (RO), Write-only (WO), Read-write (RW)

(b) Context Retrieval for Unseen Hardware (HW), Workload (WL), and `prod`+Workload (HW+WL)

Figure 5: Context Retrieval by Libra and MetricSim – Each bar shows the average KL divergences between the target context π-profiles and those of the contexts selected by each method.

In this section, we evaluate the performance of Libra's context retrieval module. Specifically, we compare Libra's ability to select similar contexts against OtterTune's method. We compare the similarity of π-profiles using KL divergence.

OtterTune is a learning-based configuration tuning system for DBMSs that leverages transfer learning to recommend optimal configurations. OtterTune utilizes factor analysis and k-means clustering to identify system metrics that best represent workloads. It then selects the most similar source by computing the Euclidean distance between the characteristic metrics of the target workload and those of past workloads. We refer to this context retrieval method as **MetricSim**. Comparison of system metrics to identify similar contexts is used in other state-of-the-art tuning system as well [49]. We calculate the KL divergence between the target context's ground-truth π-profile and that of the most similar context selected by each method to compare Libra and MetricSim.

Figure 5a is the results of context retrieval for `prod` context grouped by workload type, showing the average KL divergence between the selected source and target contexts. TPC-C workloads are treated as a single group, while Sysbench workloads are divided into 3 groups based on their read-write ratios. Libra outperforms MetricSim at identifying source contexts that have similar π-profiles as target contexts. This result shows that it is possible to identify the most similar context based on π-profile in one-shot, solving Challenge 1 (Section 1.2). Compared to Libra, MetricSim exhibits greater variance in selecting the most similar context, as measured by KL divergence between π-profiles. This indicates that the system metrics MetricSim uses do not consistently reflect similarity in terms of π-profiles. MetricSim's dependency on probabilistic algorithms such as K-means clustering and factor analysis to select characteristic system metrics [42] also contributes to the inconsistency.

Next, we simulate target contexts that differ from the training data by hardware (HW), workload (WL), or both (HW+WL). For HW, we fix the workload to `TPCC-100` and vary the hardware. For WL, we fix the hardware to `xxxlarge` and vary the workload. For HW+WL,

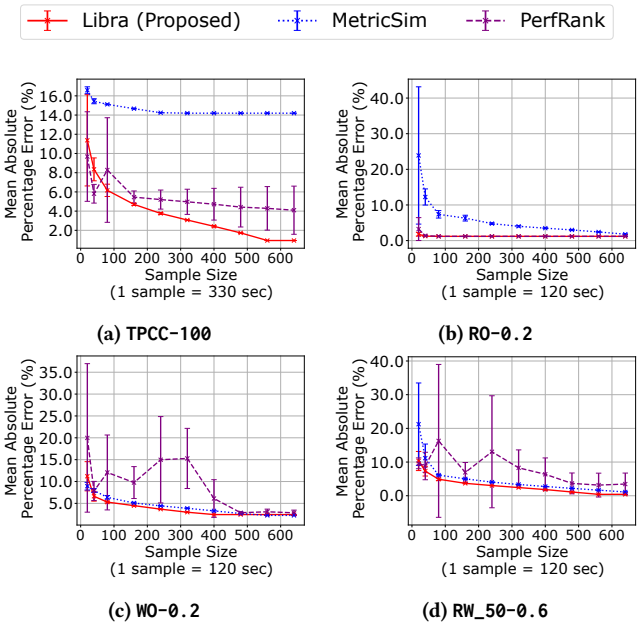


Figure 6: Effect of Selecting Similar Contexts

we set the target hardware to `prod` and compute the average KL divergence across all workloads. The data of the fixed workload or hardware is assumed to be present in the training data. Figure 5b shows that whether the target context differs by hardware only, workload only, or both, Libra selects source contexts that are more similar to target contexts in terms of π-profiles. In both Libra and MetricSim, average KL divergence increases in the order of HW, WL, to HW+WL. This trend indicates that it is easier to identify similar contexts when the difference is limited to hardware, compared to when workloads differ. Selecting similar contexts becomes most challenging when both hardware and workload vary.

6.5 Effect of Selecting Similar Contexts

To understand how the π-profile-similarity can affect the accuracy of the resulting performance model, we conduct end-to-end transfer learning. We interchange the context retrieval component, using either the proposed method or existing methods to identify the most similar context. Once the contexts are identified, we proceed with Libra's data transfer module and compare the resulting model accuracy to isolate the effect of the context retrieval method.

In addition to MetricSim, we implement **PerfRank**, a context retrieval method used in ResTune [48] and OpAdviser [50]. Unlike MetricSim, which relies on system metrics, PerfRank compares contexts based on the performance rankings of configurations. For each pair of samples from the target context, PerfRank checks whether the source model predicts the same relative performance ordering as observed in the target. If the model agrees on which configuration performs better, the pair is counted as a *concordant ranking pair*. The similarity score is defined as the number of concordant ranking pairs. While Libra and MetricSim identify the most similar

context in one shot, PerfRank dynamically selects a new source context each time a new target sample is collected.

Figure 6 shows the results of transfer learning for target contexts involving prod hardware and Table 7 lists the selected parameters. Because Libra selects source contexts with π-profiles similar to the target in one shot, it consistently minimizes the error more quickly than MetricSim and PerfRank.

6.5.1 Comparison Against MetricSim. MetricSim performs relatively well on write-only and read-write workloads, but its performance drops for TPC-C and read-only workloads. This pattern aligns with the quality of context retrieval for the prod context in Figure 5a. These results show that the similarity of the selected source context directly influences transfer learning effectiveness.

The result of TPC-C in Figure 6a and read-only in Figure 6b exemplify 2 problems caused by the comparison of system metrics to identify similar contexts. The first problem is the negative transfer. For the target context (prod, TPCC-100), MetricSim selects (xxlarge, WO-1.0) as the source context. While the buffer pool size has the largest performance impact in the target context, it has a minimal impact in the source context. Consequently, the data transfer module regards this parameter as unimportant and cannot cover all the important parameters, resulting in a model with 14% error. Libra's context retrieval module, in contrast, selects (xxlarge, RW_50-0.6) as the source context, sufficiently covering the majority of the target important parameters.

The second problem arises from covering excess parameters. The target context in Figure 6b has only 2 parameters that have significant impact. While Libra selects a context with exactly the same important parameters, reducing the sampling space by 97%, MetricSim selects a context with 9. This results in 7 false positives, causing the data transfer to require 32× more samples to build an accurate model as it wastes time on sampling irrelevant parameters.

Libra avoids both of these issues by selecting source contexts according to π-profile similarity, increasing the change that the selected context shares the same important parameters as the target. As a result, Libra minimizes the negative transfer and improves the transfer learning efficiency, achieving up to 32× faster convergence.

6.5.2 Comparison Against PerfRank. Although PerfRank is less susceptible to negative transfer compared to MetricSim, it requires a relatively large number of target samples to reliably produce accurate models. In the read-only case, where the performance surface is primarily influenced by just two parameters, PerfRank can quickly identify the most similar context, leading to fast model convergence. However, in other contexts where performance depends on a larger number of parameters, PerfRank needs substantially more samples to accurately distinguish the most relevant context and improve the prediction accuracy. In the TPC-C workload, while Libra achieves an error below 5% with just 160 samples, PerfRank needs 320 samples, twice as many, to reach the same error threshold. The instability of the model accuracy is also an issue. As Figures 6c and 6d show, with limited number of samples, PerfRank exhibits high variance in model error. Its error fluctuates as it repeatedly updates its choice of context, leading to unstable and prolonged convergence. Libra's context retrieval identifies a relevant source context in one shot, enabling faster convergence.

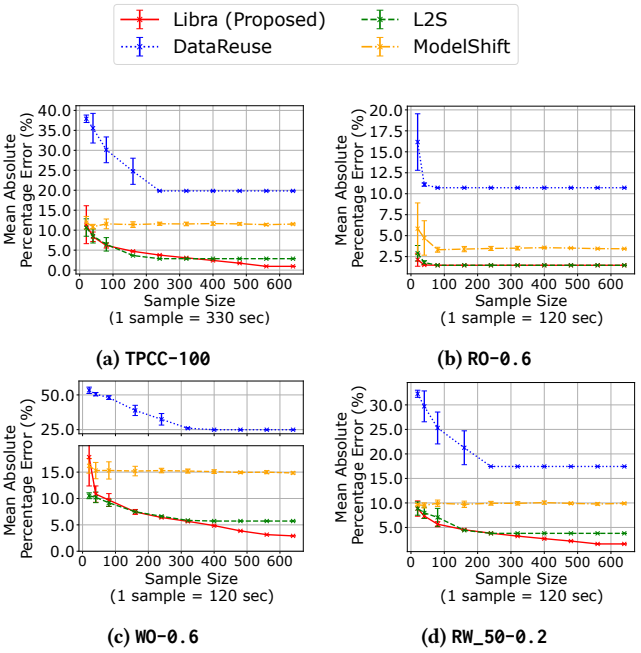


Figure 7: Comparing Transfer Algorithms

6.6 Comparing Transfer Algorithms

We compare Libra's data transfer module with ModelShift, DataReuse, and L2S. For all methods, the source context is selected using Libra's context retrieval to ensure that any differences in performance arise solely from the transfer strategy. **ModelShift** [41] linearly transforms the prediction from a source context model to estimate target performance. **DataReuse** [17] uses the source context data and data samples from the target context to build a performance model. OtterTune adops DataReuse as its transfer algorithm. To match the OtterTune implementation, our version of DataReuse performs sampling space reduction using Lasso-based parameter selection [42]. Both ModelShift and DataReuse are **data augmentation methods** that incorporate source context data directly into the target model training process. **L2S** [15, 16] uses stepwise regression to identify important parameters likely to be shared between the source and target contexts and samples the data of those parameters to build the target model. To ensure a fair comparison, we use Gaussian Process regression as the performance predictor for all the methods.

Figure 7 shows the result of transfer learning conducted for the prod contexts. Libra consistently outperforms other state-of-the-art transfer algorithms, minimizing the model's prediction error to low values in much less samples from the target context. Existing methods experience negative transfer, as shown in Figure 8.

6.6.1 Comparison Against Data-Augmentation Methods. DataReuse produces a model with the worse accuracy for all cases, as shown in Figure 7. The fundamental problem with DataReuse is that it completely disregards the absolute difference between contexts. Since absolute performance values depend on many factors, it is rare for two contexts to exhibit performance trends on the exact same scale.

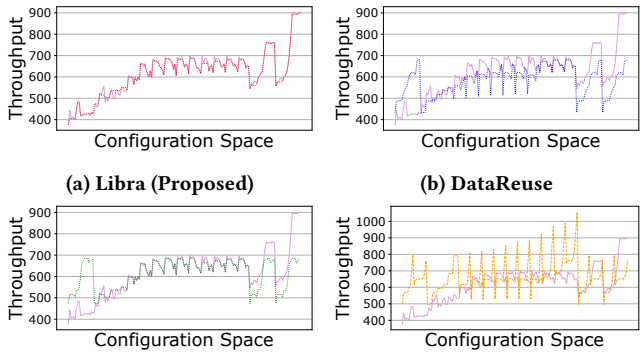


Figure 8: Actual vs Predicted Throughput – Write-only skew 0.6, sample size = 400. The actual value is depicted by the pink solid line. Each point in the configuration space represents one of 172 unique DBMS configurations defined by four parameters (1, 3, 7, and 11). DataReuse, L2S, and ModelShift predictions deviate, showing negative transfer, while Libra matches actual throughput.

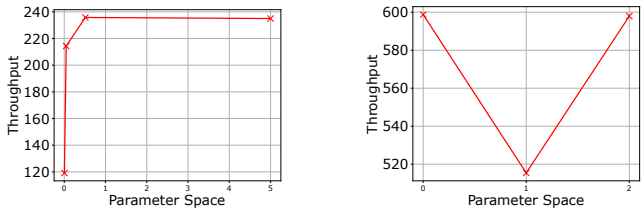


Figure 9: Problem of L2S – Non-linear Parameter Behaviors – Parameter space denotes the range of parameter values.

For this reason, DataReuse experiences negative transfer even if the performances in two contexts trend similarly.

ModelShift creates more accurate models, achieving peak accuracy with fewer than 100 samples across all cases. However, its accuracy displays no improvement even with additional samples. Like DataReuse, ModelShift augments the training data with source context data to predict target performance. Unless the source context closely resembles the performance surface of the target context, negative transfer remains likely, even after applying a linear transformation to adjust for differences in scale.

6.6.2 Comparison Against Conventional Parameter Selection. Relying on sampling space reduction instead of data augmentation enables Libra and L2S to consistently obtain accurate models.

In Figure 7b, Libra and L2S have comparable result as they both reduce the error to below 3% in less than 20 samples. In other cases, Libra performs better by more reliably identifying important parameters and avoiding the false negatives that hinder L2S.

L2S incurs higher prediction error because it relies on stepwise regression for parameter selection, an approach known to produce unreliable results [9, 38]. Stepwise regression struggles to capture non-linear relationships, often excluding impactful parameters and compromising model accuracy. In Figures 7a, 7c, and 7d, it mislabels certain parameters (Figure 9) as unimportant and excludes them from sampling, despite their significant non-linear influence on performance. Whether linear or non-linear, Libra captures important parameters by taking the ratio of 99th and 1st percentile performance, enabling it to minimize negative transfer and improve convergence efficiency. The selected parameters for each experiment is listed in Table 7.

Table 7: Important Parameters Selected in the Experiments of Figure 6 and Figure 7. Each number refers to the parameter ID defined in Table 3. (init) denotes the parameters selected at initialization, while (fix) denotes those selected after the source context has been fixed through sufficient sampling.

	Workload			
Figure 6	**TPCC-100**	**RO-0.2**	**WO-0.2**	**RW_50-0.6**
Libra	1,3,4,7–12	1,12	1,3,4,6,8–11	1,3,4,7–12
Metric-Sim	3,4,6,8–11	1–4, 7–12	1,3,4,7–12	1–4,7–12
PerfRank (init)	4,6,11,12	1,3,8, 9,11,12	1,3,8,9,11,12	4,6,11,12
PerfRank (fix)	1,2,3,8–11	1,12	1,3,4,7–12	1,3,4,6–12
Figure 7	**TPCC-100**	**RO-0.6**	**WO-0.6**	**RW_50-0.2**
Libra	1,3,4,7–12	1,12	1,3,4,6,8–11	1,3,4,7–12
L2S	1,3,8–11	1,12	1,3,6,8–10,12	1,3,9,10,12

6.7 End-To-End Transfer Learning

Finally, we compare the complete transfer learning pipeline of Libra against two existing methods, **MetricSim+DataReuse** and **PerfRank+Ensemble**, which are integrated into state-of-the-art DBMS tuning systems OtterTune and ResTune, respectively. Many automatic tuning systems rely on accurate performance prediction to recommend optimal configurations, and both OtterTune and ResTune adopt transfer learning strategies to build accurate models with minimal sampling. MetricSim+DataReuse is the strategy used in OtterTune [42]. It uses MetricSim for context retrieval and DataReuse as its transfer algorithm. PerfRank+Ensemble, used within ResTune [48], employs PerfRank to calculate context similarities. Its transfer algorithm is a data augmentation method directly tied to PerfRank. It constructs a weighted ensemble of the target model and all previously acquired source models (Ensemble), with weights determined by PerfRank's similarity scores.

Libra outperforms both methods across all evaluation settings. Compared to MetricSim+DataReuse, Libra reduces prediction error by an average of 95.6%. MetricSim+DataReuse suffers from severe negative transfer because MetricSim often fails to retrieve truly similar source contexts, causing DataReuse to incorporate misaligned source data into model training. Compared to **PerfRank+Ensemble**, Libra achieves the same level of accuracy with 2 to 32× fewer samples. PerfRank+Ensemble depends on a large number of samples to assess context similarity. When sample size is limited, it tends to include less relevant source models in the ensemble, reducing accuracy and stability.

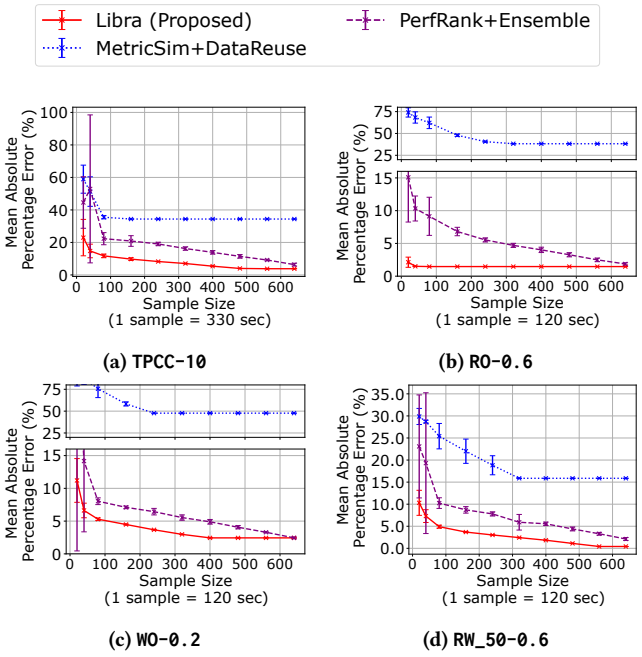


Figure 10: Performance Prediction Accuracy of Libra Compared to MetricSim+DataReuse and PerfRank+Ensemble

7 RELATED WORK

Scope of Transfer Learning: Previous studies have examined transfer learning in video encoders, data compressors, stream processing systems, machine learning models, compilers, operating systems, etc. [15–17, 23, 24, 27, 29, 41]. We focus on the widely used database system MySQL.

Prior work has examined context variation across hardware [29, 41, 48], workloads [42, 48], software versions [27], and related systems [24]. In practice, multiple contextual factors often vary simultaneously. Jamshidi et al. analyzed the interaction of hardware, workload, and software versions on system performance [15]. We focus on the combined influence of hardware and workload, as both frequently vary in industrial settings.

Context Retrieval: BEETLE [23] selects a source context by testing how generalizable each context is within the past contexts. The selected context is used for transfer learning regardless of similarity to the target. Our experiments show that the optimal source context varies by target, highlighting the need to retrieve the context most similar to the target. OtterTune and OnlineTune [42, 49] use runtime-statistic to calculate context similarity, but these metrics do not consistently reflect similarity in performance trends. ResTune and OpAdviser [48, 50] instead compare performance ranking of configurations, which improves transfer quality but demands many target samples. OPPerTune [40] uses decision tree to classify contexts based on performance, but its training requires many target samples and cannot be prepared pre-deployment.

Transfer Algorithm: DataReuse [17] is a standard approach used in DBMS tuning systems [42, 49]. DataReuse is effective when the source is highly similar to the target. ModelShift [41] is an approach that adjusts the performance difference caused by hardware difference. Both are data augmentation methods which augment the training data with samples from other contexts. These methods are vulnerable to negative transfer when hardware or workload differences introduce performance-trend discrepancies [16, 29]. L2S [16] and LlamaTune [19] aim to shrink target sampling space to reduce data collection cost. L2S assumes impactful parameters are the same in any contexts [15, 16], but our experiments show parameter sensitivity varies widely, motivating Libra's parameter-sensitivity–based context retrieval. Llamatune [19] reduces sampling space without prior context data, but Libra can reduce the space further by leveraging data from a similar context.

Table 8: Comparison of Libra with Existing Transfer Learning Methods – N.A. indicates that the method does not perform context retrieval. Libra achieves performance-relevant context retrieval in one-shot, avoids data augmentation to minimize negative transfer, and reduces sampling space to accelerate model convergence.

System	Perf. Relevant Retrieval	Small Sample Retrieval	Avoid Data Augmentation	Reduces Sampling Space
Proposed	Yes	Yes	Yes	Yes
ModelShift[41]	N.A.	N.A.	No	No
DataReuse[17]	N.A.	N.A.	No	No
L2S[16]	N.A.	N.A.	Yes	Yes
ChimeraTL[29]	N.A.	N.A.	No	Yes
OtterTune[42]	No	Yes	No	Yes
ResTune[48]	Yes	No	No	No
BEETLE[23]	No	Yes	No	No

8 CONCLUSION

We presented Libra, an end-to-end transfer learning framework for cost-efficient performance modeling. Libra addresses two challenges of transfer learning: identifying source contexts likely to transfer well and sampling from the target context efficiently to construct an accurate performance model. We introduced a novel context retrieval method based on π-*profiles*, selecting similar contexts based on a performance-relevant comparison. We incorporated an MLP that predicts the π-profile from a single runtime snapshot of the target context. Once the similar context are identified, Libra selects important parameters by comparing 99th and 1st percentile performance ratio, and reduces the configuration space for exploration. Sampling process prioritizes parameters with higher performance impact to efficiently build a performance model. Libra outperforms existing approaches in terms of convergence speed (up to 32× faster) and predictive accuracy (95.6% error reduction).

ACKNOWLEDGMENTS

We would like to thank Shohei Matsuura and Takashi Miyazaki for their valuable industrial insights. This paper is based on results obtained from Research and Development Project of the Enhanced Infrastructures for Post-5G Information and Communication Systems (JPNP20017), JPNP16007 commissioned by the New Energy and Industrial Technology Development Organization (NEDO), JSPS KAKENHI Grant Number 25H00446, JST CREST Grant Number JPMJCR24R4, SECOM Science and Technology Foundation, JST COI-NEXT SQAI (JPMJPF2221), and JST BOOST (JPMJBS2409).

REFERENCES

[1] 2025. akopytov/sysbench. https://github.com/akopytov/sysbench.

[2] 2025. MySQL. https://www.mysql.com/.

[3] 2025. Percona-Lab/sysbench-tpcc. https://github.com/Percona-Lab/sysbench-tpcc.

[4] 2025. Prometheus: Monitoring System and Time Series Database. https://prometheus.io/.

[5] 2025. Tatzhiro/DBMSTransferLearning. https://github.com/Tatzhiro/DBMSTransferLearning

[6] Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag.

[7] Transaction Processing Performance Council. 2010. TPC-C Benchmark (Revision 5.11.0). https://www.tpc.org/TPC_Documents_Current_Versions/pdf/tpc-c_v5.11.0.pdf

[8] Songyun Duan, Vamsidhar Thummala, and Shivnath Babu. 2009. Tuning Database Configuration Parameters with ITuned. *Proc. VLDB Endow.* 2, 1 (2009), 1246–1257.

[9] Lynda Flom and David Cassell. 2007. Stopping Stepwise: Why Stepwise and Similar Selection Methods Are Bad, and What You Should Use. *Proc. NESUG* (2007).

[10] Jesus Flores-Contreras, Carlos Ruiz, Pablo Salazar, Hector A. Duran-Limon, Efren Mezura-Montes, Nicandro Cruz-Ramirez, and Héctor-Gabriel Acosta-Mesa. 2015. A Performance Prediction Model for Database Environments: A Preliminary Analysis. In *Proc. HPCC/CSS/ICESS*. IEEE, 1707–1712.

[11] Jianmei Guo, Krzysztof Czarnecki, Sven Apel, Norbert Siegmund, and Andrzej Wąsowski. 2013. Variability-Aware Performance Prediction: A Statistical Learning Approach. In *Proc. ASE*. IEEE, 301–311.

[12] Benjamin Hilprecht and Carsten Binnig. 2022. Zero-Shot Cost Models for Out-of-the-Box Learned Cost Prediction. *Proc. VLDB Endow.* 15, 11 (2022), 2361–2374.

[13] Darong Huang, Luis Costero, Ali Pahlevan, Marina Zapater, and David Atienza. 2024. CloudProphet: A Machine Learning-Based Performance Prediction for Public Clouds. *IEEE Trans. Sustain. Comput.* 9, 4 (2024), 661–676.

[14] Rob J. Hyndman and Anne B. Koehler. 2006. Another Look at Measures of Forecast Accuracy. *Int. J. Forecast.* 22, 4 (2006), 679–688.

[15] Pooyan Jamshidi, Norbert Siegmund, Miguel Velez, Christian Kästner, Akshay Patel, and Yuvraj Agarwal. 2017. Transfer Learning for Performance Modeling of Configurable Systems: An Exploratory Analysis. In *Proc. ASE*. IEEE, 497–508.

[16] Pooyan Jamshidi, Miguel Velez, Christian Kästner, and Norbert Siegmund. 2018. Learning to Sample: Exploiting Similarities across Environments to Learn Performance Models for Configurable Systems. In *Proc. ESEC/FSE*. ACM, 71–82.

[17] Pooyan Jamshidi, Miguel Velez, Christian Kästner, Norbert Siegmund, and Prasad Kawthekar. 2017. Transfer Learning for Improving Model Predictions in Highly Configurable Software. In *Proc. SEAMS*. IEEE, 31–41.

[18] Konstantinos Kanellis, Ramnatthan Alagappan, and Shivaram Venkataraman. 2020. Too Many Knobs to Tune? Towards Faster Database Tuning by Pre-Selecting Important Knobs. In *Proc. HotStorage*. USENIX Association.

[19] 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.

[20] Shin-Gyu Kim, Hyeonsang Eom, and Heon Y. Yeom. 2013. Virtual Machine Consolidation Based on Interference Modeling. *J. Supercomput.* 66, 3 (2013), 1489–1506.

[21] Rahul Krishna and Tim Menzies. 2019. Bellwethers: A Baseline Method for Transfer Learning. *IEEE Trans. Softw. Eng.* 45, 11 (2019), 1081–1105.

[22] Rahul Krishna, Tim Menzies, and Wei Fu. 2016. Too Much Automation? The Bellwether Effect and Its Implications for Transfer Learning. In *Proc. ASE*. ACM, 122–131.

[23] Rahul Krishna, Vivek Nair, Pooyan Jamshidi, and Tim Menzies. 2021. Whence to Learn? Transferring Knowledge in Configurable Systems Using BEETLE. *IEEE Trans. Softw. Eng.* 47, 12 (2021), 2956–2972.

[24] Luc Lesoil, Hugo Martin, Mathieu Acher, Arnaud Blouin, and Jean-Marc Jezequel. 2022. Transferring Performance between Distinct Configurable Systems: A Case Study. In *Proc. VaMoS*. ACM.

[25] Zibo Liang, Xu Chen, Yuyang Xia, Runfan Ye, Haitian Chen, Jiandong Xie, and Kai Zheng. 2024. DACE: A Database-Agnostic Cost Estimator. In *Proc. ICDE*. IEEE, 4925–4937.

[26] Ryan Marcus and Olga Papaemmanouil. 2019. Plan-Structured Deep Neural Network Models for Query Performance Prediction. *Proc. VLDB Endow.* 12, 11 (2019), 1733–1746.

[27] Hugo Martin, Mathieu Acher, Juliana Alves Pereira, Luc Lesoil, Jean-Marc Jézéquel, and Djamel Eddine Khelladi. 2022. Transfer Learning Across Variants and Versions: The Case of Linux Kernel Size. *IEEE Trans. Softw. Eng.* 48, 11 (2022), 4274–4290.

[28] Kevin P. Murphy. 2012. *Machine Learning: A Probabilistic Perspective*. The MIT Press.

[29] Tatsuhiro Nakamori, Shohei Matsuura, Takashi Miyazaki, Sho Nakazono, Taiki Sato, Takashi Hoshino, and Hideyuki Kawashima. 2024. ChimeraTL: Transfer Learning in DBMS with Fewer Samples. In *Proc. ICDEW*. IEEE, 310–316.

[30] Piotr Nawrocki and Mateusz Smendowski. 2023. Long-Term Prediction of Cloud Resource Usage in High-Performance Computing. In *Proc. ICCS*. Springer-Verlag, 532–546.

[31] Thanh-Phuong Pham, Juan J. Durillo, and Thomas Fahringer. 2020. Predicting Workflow Task Execution Time in the Cloud Using a Two-Stage Machine Learning Approach. *IEEE Trans. Cloud Comput.* 8, 1 (2020), 256–268.

[32] Carl Edward Rasmussen and Christopher K. I. Williams. 2005. *Gaussian Processes for Machine Learning*. The MIT Press.

[33] Maximilian Rieger and Thomas Neumann. 2025. T3: Accurate and Fast Performance Prediction for Relational Database Systems with Compiled Decision Trees. *Proc. ACM Manag. Data* 3, 3 (2025), 1–27.

[34] Atrisha Sarkar, Jianmei Guo, Norbert Siegmund, Sven Apel, and Krzysztof Czarnecki. 2015. Cost-Efficient Sampling for Performance Prediction of Configurable Systems. In *Proc. ASE*. IEEE, 342–352.

[35] Shashank Shekhar, Hamzah Abdel-Aziz, Anirban Bhattacharjee, Aniruddha Gokhale, and Xenofon Koutsoukos. 2018. Performance Interference-Aware Vertical Elasticity for Cloud-Hosted Latency-Sensitive Applications. In *Proc. IEEE CLOUD*. IEEE, 82–89.

[36] Norbert Siegmund, Alexander Grebhahn, Sven Apel, and Christian Kästner. 2015. Performance-Influence Models for Highly Configurable Systems. In *Proc. ESEC/FSE*. ACM, 284–294.

[37] Norbert Siegmund, Sergiy S. Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller, and Gunter Saake. 2012. Predicting Performance via Automated Feature-Interaction Detection. In *Proc. ICSE*. IEEE, 167–177.

[38] Gary Smith. 2018. Step Away from Stepwise. *J. Big Data* 5, 32 (2018).

[39] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *Proc. NIPS*. 2951–2959.

[40] Gagan Somashekar, Karan Tandon, Anush Kini, Chieh-Chun Chang, Petr Husak, Ranjita Bhagwan, Mayukh Das, Anshul Gandhi, and Nagarajan Natarajan. 2024. OPPerTune: Post-Deployment Configuration Tuning of Services Made Easy. In *Proc. NSDI*. USENIX Association.

[41] Pavel Valov, Jean-Christophe Petkovich, Jianmei Guo, Sebastian Fischmeister, and Krzysztof Czarnecki. 2017. Transferring Performance Prediction Models Across Different Hardware Platforms. In *Proc. ICPE*. ACM, 39–50.

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

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

[44] Ziniu Wu, Ryan Marcus, Zhengchun Liu, Parimarjan Negi, Vikram Nathan, Pascal Pfeil, Gaurav Saxena, Mohammad Rahman, Balakrishnan Narayanaswamy, and Tim Kraska. 2024. Stage: Query Execution Time Prediction in Amazon Redshift. In *Proc. SIGMOD*. ACM, 280–294.

[45] Dong Young Yoon, Barzan Mozafari, and Douglas P. Brown. 2015. DBSeer: Pain-Free Database Administration Through Workload Intelligence. *Proc. VLDB Endow.* 8, 12 (2015), 2036–2039.

[46] 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 *Proc. SIGMOD*. ACM, 415–432.

[47] 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.

[48] 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 *Proc. SIGMOD*. ACM, 2102–2114.

[49] Xinyi Zhang, Hong Wu, Yang Li, Jian Tan, Feifei Li, and Bin Cui. 2022. Towards Dynamic and Safe Configuration Tuning for Cloud Databases. In *Proc. SIGMOD*. ACM, 631–645.

[50] 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.

[51] Yi Zhang, Jianmei Guo, Eric Blais, and Krzysztof Czarnecki. 2015. Performance Prediction of Configurable Software Systems by Fourier Learning. In *Proc. ASE*. IEEE, 365–373.

[52] Yue Zhao, Gao Cong, Jiachen Shi, and Chunyan Miao. 2022. QueryFormer: A Tree Transformer Model for Query Plan Representation. *Proc. VLDB Endow.* 15, 8 (2022), 1658–1670.