Facilitating Database Tuning with Hyper-Parameter Optimization: A Comprehensive Experimental Evaluation

[Experiment, Analysis & Benchmark]

Xinyi Zhang[†‡*], Zhuo Chang[†‡*], Yang Li[†], Hong Wu[‡], Jian Tan[‡], Feifei Li[‡], Bin Cui[†]

[†]EECS, Peking University [‡]Alibaba Group

[†]{zhang_xinyi, z.chang, liyang.cs, bin.cui}@pku.edu.cn [‡]{hong.wu, j.tan, lifeifei}@alibaba-inc.com

ABSTRACT

Recently, using automatic configuration tuning to improve the performance of modern database management systems (DBMSs) has attracted increasing interest from the database community. This is embodied with a number of systems featuring advanced tuning capabilities being developed. However, it remains a challenge to select the best solution for database configuration tuning, considering the large body of algorithm choices. In addition, beyond the applications on database systems, we could find more potential algorithms designed for configuration tuning. To this end, this paper provides a comprehensive evaluation of configuration tuning techniques from a broader perspective, hoping to better benefit the database community. In particular, we summarize three key modules of database configuration tuning systems and conduct extensive ablation studies using various challenging cases. Our evaluation demonstrates that the hyper-parameter optimization algorithms can be borrowed to further enhance the database configuration tuning. Moreover, we identify the best algorithm choices for different modules. Beyond the comprehensive evaluations, we offer an efficient and unified database configuration tuning benchmark via surrogates that reduces the evaluation cost to a minimum, allowing for extensive runs and analysis of new techniques.
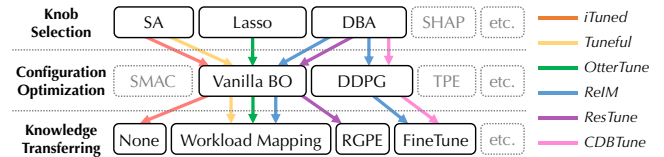
1 INTRODUCTION

Modern database management systems (DBMSs) have hundreds of configuration knobs that determine their runtime behaviors [15]. Setting the appropriate values for these configuration knobs is crucial to pursue the high throughput and low latency of a DBMS.

Figure 1: Algorithm Choices of Configuration Tuning Systems: Black boxes denote the algorithms adopted by existing database tuning systems (indicated by colored paths), and grey boxes denote the algorithms in the HPO field.

Given a target workload, configuration tuning aims to find configurations that optimize the database performance. This problem is proven to be NP-hard [76]. To find a promising configuration for the target workload, database administrators (DBAs) put significant effort into tuning the configurations. Unfortunately, manual tuning struggles to handle different workloads and hardware environments, especially in the cloud environment [65]. Therefore, automatic configuration tuning attracts intensive interests in both academia and industry [1, 16, 42, 55, 71, 72, 74, 88].

Recently, there has been an active research area on automatically tuning database configurations using Machine Learning (ML) techniques [3, 18, 24, 44, 47, 55, 91, 92]. We summarize three key modules in the existing tuning systems: *knob selection* that prunes the configuration space, *configuration optimization* that samples promising configurations over the pruned space, and *knowledge transfer* that further speeds up the tuning process via historical data. Based on the techniques used in *configuration optimization* module, these systems can be categorized into two major types: Bayesian Optimization (BO) based systems [3, 18, 24, 92] and Reinforcement Learning (RL) based [47, 91] systems. Owing to these efforts, modern database systems are well-equipped with powerful algorithms for configuration tuning. Examples includes Lasso algorithm in OtterTune [3], Sensitivity Analysis (SA) in Tuneful [24] for automatic *knob selection*; Bayesian Optimization (e.g., iTuned [18], ResTune [92]), Reinforcement Learning (e.g., CDBTune [91], Qtune [47]) for *configuration optimization*; and workload mapping in OtterTune, RGPE [26] in ResTune [92] for *knowledge transfer*. The various algorithms in each module of the configuration tuning systems enrich the solutions for database optimization and demonstrate superior performance and efficiency compared to manual tuning.

1.1 Motivation

Whilst a large body of methods have been proposed, a comprehensive evaluation is still missing. Existing evaluations [3, 18, 47, 91, 92]

compare tuning systems from a macro perspective, lacking analysis of algorithm components in various scenarios. This motivates us to conduct a comprehensive comparative analysis and experimental evaluation of database tuning approaches from a micro perspective. We now discuss the issues of the existing work.

I1: Missing comparative evaluations of intra-algorithms in different modules. Emerging database tuning systems are characterized by new intra-algorithms (i.e., algorithms in each module) such as OtterTune with Lasso-based knob selection and workload mapping, CDBTune with DDPG optimizer, ResTune with RGPE transfer framework. Figure 1 plots the three key modules and the corresponding intra-algorithms we extracted from the designs of these systems. When designing a database tuning system, we can construct many possible "paths" across the intra-algorithm choices among the three modules, even not limited to existing designs. For example, each *knob selection* algorithm determines a unique configuration space and can be "linked" to any of the *configuration optimization* algorithms. Given so many possible combinations, it remains unclear to identify the best "path" for database configuration tuning in practice. Existing researches focus on the evaluation of the entire tuning systems [3, 24, 91] or limited intra-algorithms in some of the modules [4], failing to reveal which intra-algorithm contributes to the overall success. For example, the choice of intra-algorithms in *knob selection* module is often overlooked, yet important, since different algorithms can lead to distinct configuration spaces, affecting later optimization. To this end, it is essential to conduct a thorough evaluation with system breakdown and fine-grained intra-algorithm comparisons.

I2: Absence of analysis for high-dimensional and heterogeneous scenarios. There are two challenging scenarios for configuration optimization: *high dimensionality* and *heterogeneity* of configuration space. DBMS has hundreds of configuration knobs that could be continuous (e.g., *innodb_buffer_pool_size* and *tmp_table_size*) or categorical (e.g., *innodb_stats_method* and *innodb_flush_neighbors*). We refer to the scenario with knobs of various types as *heterogeneity*. And, the categorical knobs vary fundamentally from the continuous ones in differentiability and continuity. For instance, existing BO-based methods tend to yield the best results for low-dimensional and continuous spaces. However, as for the high-dimensional space, they suffer from the over-exploration issue [70]. In addition, vanilla BO methods assume a natural ordering of input value [81], thus struggling to model the heterogeneous space. Consequently, when analyzing the optimizers, it is essential to compare their performance under the two scenarios. Although *high-dimensionality* has been mentioned in CDBTune [91], the analysis and evaluation of *heterogeneity* for database configuration tuning are still missing.

I3: Limited solution comparison without a broader view beyond database community. The database configuration tuning is formulated as a black-box optimization problem over configuration space. Thanks to the efforts of recent studies, today's database practitioners are well-equipped with many techniques of configuration tuning. However, when we look from a broader view, we can find plenty of toolkits and algorithms designed for the black-box optimization, especially hyper-parameter optimization (HPO) approaches [6, 30, 49, 60, 82, 89]. HPO aims to find the optimal hyper-parameter configurations of a machine learning algorithm as rapidly as possible to minimize the corresponding loss function. Database configuration tuning shares a similar spirit with HPO since they both optimize a black-box objective function with expensive function evaluations. We notice that recent advances in HPO field have shown promising improvement in high-dimensional and heterogeneous configuration spaces [22, 34, 40, 84, 86]. Despite their success, such opportunities to further facilitate database configuration tuning have not been investigated in the literature.

I4: Lack of cheap-to-evaluate and unified database tuning benchmarks. Evaluating new algorithms in database tuning systems can be costly, time-consuming, and hard to interpret. It requires DBMS copies, computing resources, and the infrastructure to replay workloads and the tools to collect performance metrics. It takes dozens of hours to conduct a single run of optimization with ceaseless computing resources. Moreover, obtaining a sound evaluation of more baselines takes several-fold times longer with expensive computing costs. In addition, the database performance can fluctuate across instances of the same type even though the configuration and workload are the same [4]. Therefore, we cannot run the optimizers in parallel to reduce the evaluation time, making the evaluation more troublesome. These problems pose a considerable barrier to the solid evaluation of database tuning systems. An efficient and unified tuning benchmark is needed.

1.2 Our Contributions

Driven by the aforementioned issues, we provide comprehensive analysis and experimental evaluation of database configuration tuning approaches. Our contributions are summarized as follows:
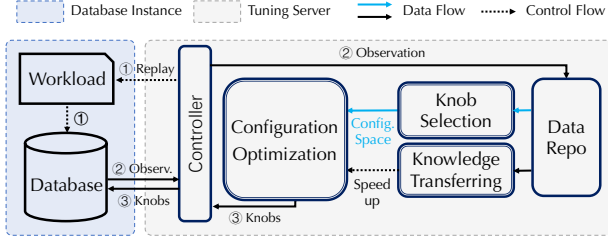
C1. We present a unified pipeline with three key modules and evaluate the fine-grained intra-algorithms. For **I1**, we break down all the database tuning systems into three key modules and evaluate the corresponding intra-algorithms. We evaluate and analyze each module's intra-algorithms to answer the motivating questions: *(1) How to determine tuning knobs? (2) Which optimizer is the winner? (3) Can we transfer knowledge to speed up the target tuning task?* With our in-depth analysis, we identify design trade-offs and potential research opportunities. We discuss the best "paths" across the three modules in various scenarios and present all-way guidance for configuration tuning in practice.

C2. We construct extensive scenarios to benchmark multiple optimizers. For **I2**, we construct specific evaluations and carry out comparative analysis for the two challenges. For *high dimensionality*, we conduct evaluations of the optimizers over configuration spaces of three sizes: small, medium, and large. We specifically analyze the different performances when optimizing over small/medium and large configuration spaces. For *heterogeneity*, we construct a comparison experiment with continuous and heterogeneous configuration spaces to validate the optimizers' support for *heterogeneity*. Such evaluation setting assists us in better understanding the pros and cons of existing database optimizers.

C3. Think out of the box: we apply and evaluate advanced HPO techniques in database tuning problems. For **I3**, we survey existing solutions for black-box optimization. We find that many recent approaches in the hyper-parameter optimization (HPO) field

Table 1: Taxonomy and Brief Description of Existing Database Configuration Tuning Systems.

Category	Configuration Tuning System	Application	Design Highlight
BO-based	iTuned [18]	Performance tuning for DBMS	First adopting BO
	OtterTune [3]	Performance tuning for DBMS	Incremental knob selection, workload mapping
	Tuneful [24]	Performance tuning for analytics engines	Incremental knob selection via Sensitivity Analysis
	ResTune [92]	Resourse-oriented tuning for DBMS	Adopting RGPE to transfer historical knowledge
	ReIM [44]	Memory allocation for analytics engines	Combining white-box knowledge
RL-based	CDBTune [91]	Performance tuning for DBMS	First adopting DDPG
	QTune [47]	Performance tuning for DBMS	Supporting three tuning granularities

Figure 2: Architecture and Workflow of Configuration Tuning System for DBMS.

could be borrowed to alleviate the challenges in database tuning. Therefore, besides the configuration tuning techniques proposed in the database community, we also evaluate other advanced approaches in the HPO field under the same database tuning setting. Through evaluations, we have that the best knob selection approach achieves 38.02% average performance improvement, and the best optimization approach achieves 21.17% average performance improvement as well as significant speedup compared with existing methods. We demonstrate that database practitioners can borrow strength from the HPO approaches in an out-of-the-box manner.

C4. We define an efficient database configuration tuning benchmark via surrogates. We summarize the difficulties discussed in **I4** lie in the cost and fluctuation of performance evaluations under configurations. We propose benchmarking database configuration tuning via a regression surrogate that approximates future evaluations through cheap and stable model predictions [21]. Specifically, we train regression models on *(configuration, performance)* pairs collected in an expensive offline manner and cheaply evaluate future configurations using the model's performance predictions instead of replying workloads. The tuning benchmark offers optimization scenarios that share the same configuration spaces and feature similar response surfaces with real-world configuration tuning problems. Significantly, it reduces the evaluation cost to a minimum and achieves 150~311 × overall speedup. The benchmark is publicly available to facilitate future research. With the help of the benchmark, researchers can skip time-consuming workload replay, conduct algorithm analysis and comparison efficiently, and test new algorithms with few costs.

The rest of the paper is organized as follows: Section 2 introduces the preliminaries of this paper.Next, we present a survey on algorithms for the three modules in Section 3 and describe the general experimental setup in Section 4. Then, the experimental results and analysis are presented in Sections 5-7, followed by the description of the database tuning benchmark in Section 8. We summarize our

lessons learned and identify the research opportunities in Section 9. Finally, we conclude the paper.

2 PRELIMINARIES

We first formalize the database configuration tuning problem. Then we review the architecture and workflow of database configuration tuning systems and present the taxonomy and description about the existing systems. Finally, we introduce the background of hyperparameter optimization.

2.1 Problem Statement

Consider a database system having configuration knobs $\theta_1, \ldots, \theta_m$, we define its configuration space $\Theta = \Theta_1 \times \cdots \times \Theta_m$, where $\Theta_1, \ldots, \Theta_m$ denote their respective domains. A configuration knob θ can be continuous or categorical. We denote the database performance metric as f which can be any chosen metric to be optimized, such as *throughput*, *99%th percentile latency*, etc. Given a workload and a specific configuration θ, the corresponding performance $f(\theta)$ can be observed after evaluating it in the database. We denote the historical observations as $H_t = \{(\theta_i, y_i)\}_1^t$. Assuming that the objective is a maximization problem, database configuration tuning aims to find a configuration $\theta^* \in \Theta$, where

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

2.2 System Overview and Workflow

The database configuration tuning systems share a typical architecture as shown in Figure 2. The left part shows the target DBMS and the replayed workload. The right part represents the tuning server deployed in the backend tuning cluster. The tuning server maintains a repository of historical observations from tuning tasks. The controller monitors the states of the target DMBS and transfers the data between the DBMS side and the tuning server side. It conducts stress testing to the target DBMS through standard workload testing tools (e.g., OLTP-Bench [17]) or workload runner that executes the current user's real workload [73]. There are three main modules in the tuning system: (1) *knob selection*, (2) *configuration optimization*, (3) *knowledge transfer*. The *knob selection* module prunes the configuration space by identifying important knobs. The pruned configuration space is passed to the optimizer, which suggests a promising configuration over the pruned space at each iteration. Furthermore, the *knowledge transfer* module speeds up the current tuning task by borrowing strength from past similar tasks stored in the data repository or fine-tuning the optimizer's model.

The *configuration optimization* module functions iteratively, as shown in the black arrows: (1) The controller replay the workload

to conduct stress testing. (2) After the stress testing finishes, the performance statistics are uploaded to the data repository, and the model in the configuration optimizer is updated based on the augmented observation. Optionally, the knowledge transfer module can further speed up the optimization with historical observations. (3) The optimizer generates a promising configuration over the pruned configuration space, and the configuration is applied to the target DBMS through the controller. The above procedures repeat until the model of the optimizer converges or stop conditions are reached (e.g., budget limits).

2.3 Taxonomy

Table 1 presents a taxonomy of existing configuration tuning systems with machine learning (ML) based techniques. We classify the systems into two classes based on the algorithms of their optimizers: (1) Bayesian Optimization (BO) based systems and (2) Reinforcement Learning (RL) based systems.

2.3.1 BO-based. The existing BO-based systems adopt Vanilla BO to model the relationship between configurations and database performance. They follow Bayesian Optimization (BO) framework to search for an optimal DBMS configuration: (1) fitting probabilistic surrogate models and (2) choosing the next configuration to evaluate by maximizing acquisition function.

iTuned [18] first adopts vanilla BO models to search for a well-performing configuration. It uses a stochastic sampling technique – Latin Hypercube Sampling [61] for initialization and does not use the observations collected from previous tuning sessions to speed up the target tuning task.

OtterTune [3] selects the most impactful knobs with Lasso algorithm, and increases the size of configuration space incrementally. It proposes to speed up the target tuning by a transfer framework via workload mapping. Concretely, it re-uses the historical observations of a prior workload by mapping the target workload based on the measurements of internal metrics in DBMS.

Tuneful [24] proposes a tuning cost amortization model and demonstrates its advantage when tuning the recurrent workloads. It conducts incremental knob selection via Gini score as an importance measurement to gradually decrease the configuration space. Similar to OtterTune, it also adopts the workload mapping framework to transfer knowledge across tuning tasks.

ResTune [92] defines a resource-oriented tuning problem and adopts a constrained Bayesian optimization solver. It uses an ensemble framework (i.e., RGPE) to combine workload encoder across tuning tasks to transfer historical knowledge.

ReIM [44] is designed to configure the memory allocation for distributed analytic engines, e.g., Spark. It builds a white-box model of memory allocation to speed up Bayesian Optimization.

2.3.2 RL-based. RL-based systems adopt Deep Deterministic Policy Gradient (DDPG), which is a policy-based model-free reinforcement learning agent. It treats knob tuning as a trial-and-error procedure to trade-off between exploring unexplored space and exploiting existing knowledge.

CDBTune [91] first adopts DDPG algorithm to tune the database. Its tuning agent inputs internal metrics of the DBMS and outputs

Table 2: Importance Measurements in Knob Selection.

Measure	Category	Brief Description
Lasso [79]	Variance based	Based on coefficient of linear regression, effective for when existing irrelevant features.
Gini score [63]	Variance based	Based on the times a feature is used in tree splits, successful in high-dimensional feature selection.
fANOVA [35]	Variance based	Decomposing the variance of target function, commonly used in the HPO field.
Ablation analysis[12]	Tunability based	Comparing feature difference between configurations, straightforward and intuitive.
SHAP [53]	Tunability based	Decomposing the performance change additively, solid theoretical foundation derived from game theory.

proper configuration by modeling the tuning process as a Markov Decision Process (MDP) [8].

Qtune [47] supports three tuning granularities, including workload-level tuning, It embeds workload characteristics to predict the internal metrics for query-level tuning, and clusters the queries based on their "best" knob values for cluster-level tuning.

Scope Illustration. To make our evaluation focused yet comprehensive, we employ some necessary constraints. We focus on evaluating the configuration tuning techniques at the workload level since query-lever or cluster-lever tuning are only available on Qtune. In this paper, we concentrate on performance tuning in DBMS. As for the systems targeting the other applications (i.e., performance tuning for analytic engines, resource-oriented tuning), we implement their core techniques (e.g., RGPE transfer framework) and evaluate them in a unified setting.

2.4 Hyper-Parameter Optimization

Hyper-parameter Optimization (HPO) aims to find the hyper-parameter configurations λ^* of a given algorithm A^i which has the best performance on the validation set [29]:

$$\lambda^* = \underset{\lambda \in \Lambda_i}{\arg \min} \, L(A^i, \lambda). \tag{2}$$

HPO is known as black-box optimization since nothing is known about the loss function besides its function evaluations. Bayesian Optimization (BO) has been successfully applied to solve the HPO problem [7, 57, 85]. The main idea of BO is to use a probabilistic surrogate model to describe the relationship between a hyper-parameter configuration and its performance (e.g., validation error), and then utilize this surrogate to guide the configuration search [69]. It is common to use Gaussian process (GP) as surrogates [59, 73] and SMAC [34] and TPE [9] are another two well-established methods.

3 METHODOLOGIES

In this section, we survey the methodologies for the three components, respectively. We not only cover the techniques used in existing configuration tuning systems for DBMS but also discuss the representative approaches from the HPO field.

3.1 Knob Selection

The *knob selection* module identifies and selects important knobs to tune. Although the database system has hundreds of knobs, not all of them significantly impact database performance. Selecting

important knobs can prune the configuration space and further accelerate configuration optimization.

To conduct knob selection, we first need to collect a set of observations under different configurations. Given the observations, we adopt an algorithm to rank the knobs in terms of their importance. (We refer to the ranking algorithm as "importance measurement" to distinguish it from the other techniques.) Finally, the configuration space is determined by selecting the top-k knobs according to the importance rank. There are various choices for importance measurements, leading to distinct configuration spaces and affecting later optimization. They can be classified into two categories as shown in Table 2: *variance-based* measurements and *tunability-based* measurements [87]. *Variance-based* measurement selects the knobs that have the largest impact on the database performance. It has been adopted by existing tuning systems for DBMS. *Tunability-based* measurement [67] quantifies the tunability of a knob, measuring the performance gain that can be achieved by tuning the knob from its default value. It has been applied successfully to determine the importance of hyper-parameters of ML algorithms, especially when given a well-performing default configuration [87].

3.1.1 *Variance-based Measurements*.
We introduce three variance-based measurements due to their widely-used applications and representativeness, including Lasso [79] adopted in OtterTune, Gini score [63] adopted in Tuneful, and functional ANOVA [35], a state-of-the-art importance measurements from the HPO domain.

Lasso [79] uses the coefficient of linear regression to assess the importance of features. It penalizes the L1-norm of the coefficients and forces the coefficients of irrelevant or redundant features to be zero. The L1-norm penalization makes Lasso effective when there are many irrelevant features in the training samples [19].

Gini Score [63] is derived from tree-based models, like random forest model [77]. The Gini score of each feature is defined as the number of times the given feature is used in a tree split across all the forest trees since important features discriminate the larger number of samples and are used more frequently in decision tree splits. Gini score has been successfully applied to high-dimensional feature selection [62].

fANOVA [35] (i.e., Functional analysis of variance) measures the importance of features by analyzing how much each feature contributes to the variance of target function across the configuration space. Based on a regression model (e.g., random forest), functional ANOVA decomposes the target function into additive components that only depend on subsets of its inputs. The importance of each feature is quantified by the fraction of variance it explains. Functional ANOVA is commonly used in the HPO domian due to its solid theoretical foundation [33, 54, 80].

3.1.2 *Tunability-based Measurements*.
While *Variance-based* measurements are interested in the global effect of a feature, *tunability-based* measurements focus on "good" regions of the space which are better than the default configuration. It can be used directly to determine the necessity of tuning a knob from the given default value. We introduce two typical local algorithms that can measure the tunability – ablation analysis [12] and SHAP [53].

Ablation Analysis [23] selects the feature whose change contributes the most to improve the performance of configurations.

Table 3: Algorithms for Optimizers. ✓means the optimizer has specific design for the case in the column and – means the optimizer does not have such design.

Algorithm	High-dimensionality	Heterogeneity
Vanilla BO	–	–
Mixed-Kernel BO [41]	–	✓
SMAC [34]	✓	✓
TPE [9]	–	✓
TurBO [22]	✓	–
DDPG [52]	✓	–
GA [46]	–	✓

It identifies the difference between configurations by modifying each feature iteratively from its default value to the value of well-performing configurations and evaluating the performance change [23]. For speedup, the evaluations are replaced by cheap predictions obtained from surrogates, e.g., random forest. Given a set of observations, we fit a surrogate and conduct ablation analysis between the default configuration and the better ones.

SHAP [53] (i.e., SHapley Additive exPlanations) is a unified framework to interpret the performance change derived from classic Shapley value estimation [75] in cooperative game theory. The performance change is decomposed additively between features. SHAP computes each feature's contribution (i.e., SHAP value) for pushing the default performance to the target one. Given a set of observations, each feature's tunability is calculated by averaging its positive SHAP value (assuming a maximization problem).

3.2 Configuration Optimization

In *configuration optimization* module, an optimizer suggests promising configurations and updates its model based on the evaluation results iteratively. We introduce six state-of-the-art optimizers used by the database tuning systems or from the HPO community. We also summarize their designs in Table 3.

Vanilla BO denotes the BO-based optimizer that adopts vanilla Gaussian process (GP) as its surrogate model. Vanilla GP is widely used for objective function modeling in database configuration systems [3, 18, 24], due to its expressiveness, well-calibrated uncertainty estimates and closed-form computability of the predictive distribution [36]. A GP $\mathcal{N}(\mu(\boldsymbol{\theta}), \sigma^2(\boldsymbol{\theta}))$ is fully specified by mean $\mu(\boldsymbol{\theta})$ and a variance function $\sigma^2(\boldsymbol{\theta})$, which is expressed as:

$$
\begin{aligned}
\mu(\boldsymbol{\theta}) &= k^T \left(K + \sigma^2 I\right)^{-1} y, \\
\sigma^2(\boldsymbol{\theta}) &= k(\boldsymbol{\theta}, \boldsymbol{\theta}) - k^T \left(K + \sigma^2 I\right)^{-1} k,
\end{aligned}
\tag{3}
$$

where $k = [k(\boldsymbol{\theta_1}, \boldsymbol{\theta}), ..., k(\boldsymbol{\theta_n}, \boldsymbol{\theta})]^T$ and K is the covariance matrix whose (i, j)-th entry is $K_{i,j} = k(\boldsymbol{\theta_i}, \boldsymbol{\theta_j})$. The kernel $k(\boldsymbol{\theta}, \boldsymbol{\theta}')$ models the overall smoothness of the target function. For Vanilla BO, the kernel function (e.g., RBF kernel in OtterTune) is calculated based on the Euclidean distance between two configurations, assuming the natural ordering property and continuity of configuration space.

Mixed-kernel BO [41] denotes the BO-based optimizer that adopts a mixed-kernel GP as its surrogate model. The mixed-kernel GP uses Matérn kernel for continuous features, Hamming kernel for categorical features, and finally calculates the product of these two kernels. Mátern kernel is a continuous kernel that generalizes the RBF with a smoothing parameter. Hamming kernel is based

on hamming distance, which is suitable to measure the distance between categorical variables. Since the quality of the GP depends solely on the kernel function [36], the continuous kernels adopted by vanilla BO has poor performance on handling categorical features. The mixed-kernel BO has shown promising performance over the heterogeneous configuration space empirically [41].

SMAC [34] (i.e., Sequential Model-based Algorithm Configuration) adopts a random forest based surrogate, which is known to perform well for high-dimensional and categorical input [13]. SMAC assumes a Gaussian model $N(y|\hat{\mu}, \hat{\sigma}^2)$, where the $\hat{\mu}$ and $\hat{\sigma}^2$ are the mean and variance of the random forest. SMAC supports all types of variables, including continuous, discrete, and categorical features. It has been the best-performing optimizer in Auto-WEKA [78].

TPE [9] (i.e., Tree-structured Parzen estimator) is a non-standard Bayesian optimization algorithm. While GP and SMAC modeling the probability $p(y|\theta, H)$ directly, TPE models $p(\theta|y, H)$ by tree-structured Parzen density estimators. TPE describes the configuration space by a generative process and supports categorical features. TPE has been used successfully in several papers [10, 11, 78].

TuRBO [22] (i.e., Trust-Region BO) proposes a local strategy for global optimization using independent surrogate models. These surrogates allow for local modeling of the objective function and do not suffer from over-exploration. To optimize globally, TuRBO leverages a multi-armed bandit strategy to select a promising suggestion from local models. TuRBO exhibits promising performances when solving high dimensional optimization problems [22].

DDPG [52] denotes the Deep Deterministic Policy Gradient (DDPG) algorithm that is adopted to learn the configuration tuning policy for DBMS. It has been successfully adopted by CDBTune and Qtune. While other reinforcement learning algorithms, such as Deep-Q learning [56], are limited to setting a knob from a finite set of predefined values, DDPG can work over a continuous action space, setting a knob to any value within a range. DDPG consists of two neural networks: the actor that chooses an action (i.e., configuration) based on the input states, and the critic that evaluates the selected action based on the reward. In other words, the actor decides how to suggest a configuration, and the critic provides feedback on the suggestion to guide the actor.

GA [46] (i.e., Genetic Algorithm) is a meta-heuristic inspired by the process of natural selection. In a genetic algorithm, a population of candidate solutions to an optimization problem is evolved toward better solutions, iteratively. In each iteration, the fitness of each solution, which is usually the value of the objective function, is evaluated. The candidates with higher fitness (i.g. the value of objective function) will have more chance to be selected, and their features (i.g. encoded configuration) will be recombined and mutated to form new candidates for the next iteration. Genetic algorithms are simple yet effective and naturally support categorical features. They have been applied to various research problems, including hyper-parameter optimizations [68, 90]

3.3 Knowledge Transfer

The *knowledge transfer* module is designed to accelerate the target tuning task by leveraging the experience from historical tuning tasks. We introduce three knowledge transfer frameworks – workload mapping, RGPE, and fine-tuning.

Table 4: Profile information for workloads.

Workload	Class	Size	Table	Read-Only Txns
JOB	Analytical	9.3G	21	100.0%
SYSBENCH	Transactional	24.8G	150	43.0%
TPC-C	Transactional	17.8G	9	8.0%
SEATS	Transactional	12.7G	10	45.0%
Smallbank	Transactional	2.4G	3	15.0%
TATP	Transactional	6.3G	4	40.0%
Voter	Transactional	0.06M	3	0.0%
Twitter	Web-Oriented	7.9G	5	0.9%
SIBench	Feature Testing	0.5M	1	50%

Table 5: Hardware configurations for database instances.

Instance	A	B	C	D
CPU	4 cores	8 cores	16 cores	32 cores
RAM	8GB	16GB	32GB	64GB

Workload Mapping is proposed by OtterTune. It matches the target workload to the most similar historical one based on the absolute distances of database metrics and reuses the historical observations from the similar workload. This strategy can be adopted by any BO-based optimizers assuming a Gaussian model.

RGPE [26] (i.e., ranking-weighted Gaussian process ensemble) is an ensemble model for BO-based optimizers. RGPE combines similar base GP models of historical tasks via distinguishable weights. The weights are assigned using relative ranking loss to generalize across different workloads and various hardware environments. The ensemble manner avoids the poor scaling that comes with fitting a single Gaussian process model with all the observations from similar tasks. RGPE is adopted in ResTune to accelerate the tuning process of the target tasks.

Fine-tune is used in the RL-based optimizers. For example, CDB-Tune and Qtune pre-train a basic DDPG model and fine-tune the model's weight when workloads changes [47, 91]. Such a process is claimed to help the optimizer adapt to different workloads with fewer observations instead of training a model from scratch.

4 GENERAL SETUP OF EVALUATION

Our study conducts experimental evaluation for the three key modules of configuration tuning systems: *knob selection, configuration optimization, knowledge transfer*. The evaluation is driven by the practical questions we encountered when tuning the databases:
Q1: How to determine the tuning knobs? (i.e., Which importance measurements to use and how many knobs to tune?)
Q2: Which optimizer is the winner given different scenarios?
Q3: Can we transfer knowledge to speed up the target tuning task? We describe the general setup of the evaluation in this section and leave the specific setting (e.g., procedure, metrics) when answering the corresponding questions.

4.1 Hardware, Workloads and Tuning Setting

Hardware. We conduct our experiments on cloud servers. Each experiment consists of two instances. The first instance is used for the tuning sever, deployed with 4 vCPUs and 8 GB RAM. The second instance is used for the target DBMS deployment, with four kinds of hardware configurations as shown in Table 5. We adopt version 5.7 of RDS MySQL. And we use the MySQL default configuration as the default in our experiments except that we set a larger and

reasonable buffer pool size (60% of the instance's memory). Without specification, we deploy the target DBMS on instance B.

Workloads. We use nine workloads listed in Table 4 with various characteristics. They can be categorized into four types: analytical, transactional, web-oriented, and feature testing [17]. The analytical workload JOB contains 113 multi-joint queries with realistic and complex joins [45]. The transactional workloads (i.e., SYSBENCH, TPC-C, Seats, Smallbank, TATP, and Voter) are traditional OLTP benchmarks featured with write-heavy transactions with complex relations. They cover various size and read-write ratios, as shown in Table 4. Twitter is a web-oriented benchmark that is designed to mimic real-world applications using publicly available traces and data dumps [14]. SIBench is a lightweight benchmark designed to test individual features of DBMS [38]. Except JOB and SYSBENCH, the other workloads are extracted from OLTP-Benchmark [2, 17]. When answering **Q1** and **Q2**, we mainly use the analytical workload JOB and the transactional workload SYSBENCH. To answer **Q3**, we use the eight OLTP workloads, including SYSBENCH, TPC-C, Twitter, Smallbank, SIbench, Voter, Seats, and TATP.

Tuning Setting. we run three tuning sessions per optimizer and report the median and quartiles of the best performances. Each tuning session is comprised of 200 iterations without specification. We restart the DBMS each iteration since the change of many knobs requires the restart. And then, we conduct a three-minute stress testing, replaying the given workload. For OLTP workload, we use throughput as maximization objective, and for OLAP workload, we use the 95% quantile latency as minimization objective. For a failed configuration (i.e., the one causing DBMS crash or unable to start), we set the result for that iteration to be the worst performance ever seen in order to avoid the scaling problem [4]. Following iTuned and OtterTune, we initialize each tuning session for BO-based optimizers by executing 10 configurations generated by Latin Hypercube Sampling (LHS) [61] in the first 10 iterations. When reporting the results, we calculate the performance improvement against the default configuration.

4.2 Implementation

Importance measurements. We compare the five importance measurements listed in Table 2. For Lasso [79], we adopt the implementation in OtterTune which includes second-degree polynomial features to improve the accuracy of its regression models. For ablation analysis, we use random forest implemented by scikit-learn [66] as the surrogate. For functional ANOVA [35], we use fANOVA library to calculate its importance measurement. As for SHAP [53], we adopt the implementation released by the authors with minor modification where we use the given default as the base configuration when calculating performance gain.

Optimizers. We compare the optimizers listed in Table 3. For Vanilla BO, we use a similar design with OtterTune [3]: Gaussian process as its surrogate and Expected Improvement as its acquisition function. We implement it using Botorch library [5]. DDPG is implemented using PyTorch library [64] with its neural network architecture borrowed from CDBTune [91]. We adopt the OpenBox [50]'s implementation for mixed-kernel BO and GA. For SMAC [34] and TurBO [22], we adopt the implementation released by the authors.

Table 6: Overall performance ranking (bold values are best).

Measurement	Gini	Lasso	fANOVA	Ablation Analysis	SHAP
Overall Ranking	3.13	3.50	3.00	4.30	**1.13**

Knowledge transfer frameworks. For workload mapping, we implement the methodology in OtterTune. For RGPE [26], we adopt the implementation in OpenBox [50]. We fine-tune DDPG's pre-trained models' weight when obtaining the target observations.

5 HOW TO DETERMINE TUNING KNOBS?

In order to conduct configuration optimization, we first need to choose tuning knobs that decide the configuration space. Specifically, it connotes the following two questions:

Q5.1: Which importance measurement to use to evaluate the importance of configuration knobs?

Q5.2: Given the evaluation result, how many knobs to choose for tuning?

To measure the effectiveness of importance measurements, we compare the tuning performances on each knob set selected by different importance measurements. To further understand the results from the comparison, we analyze the sensitivity of the compared measurements on training samples [39]. Lastly, we analyze the effect of knobs number by varying the size of configuration space.

5.1 Setup

There are 197 configuration knobs in MYSQL 5.7, except the knobs that do not make sense to tune (e.g., path names of where the DBMS stores files). We collect 6250 samples (i.e., observations) respectively for the two workloads SYSBENCH and JOB via Latin Hypercube Sampling (LHS) [61]. LHS attempts to distribute sample points evenly across all possible values over the 197-dimensional space. Then, we apply different importance measurements to generate knob importance rankings, and we respectively choose the most important top 5 and top 20 knob set based on the rankings. On each knob set, we perform configuration optimization, using a common BO-based method – vanilla BO and an RL-based method – DDPG, and finally collate the performance improvement after tuning. In addition, we conduct sensitivity analysis varying the number of training samples to compare their important ranks, and the Coefficient of Determination [31] (i.e. R^2) of modeling. Finally, we vary the number of tuning knobs to analyze the effect of configuration space with different sizes.

5.2 Which importance measurement to use for ranking knobs' importance?

Figure 3 presents the tuning improvement over the knob set generated by different importance measurements, and Table 6 summarizes their overall performance ranking. We observe that the tunability-based method SHAP achieves the best average ranking. This is due to that SHAP recommends the knobs worthy tuning. When changing a knob's value from the default only leads to the downgrade of database performance, SHAP will recommend not to tune the knob, while variance-based measurements will consider this knob to have a large impact on the performance and need tuning. The default values of database knobs are designed to be robust
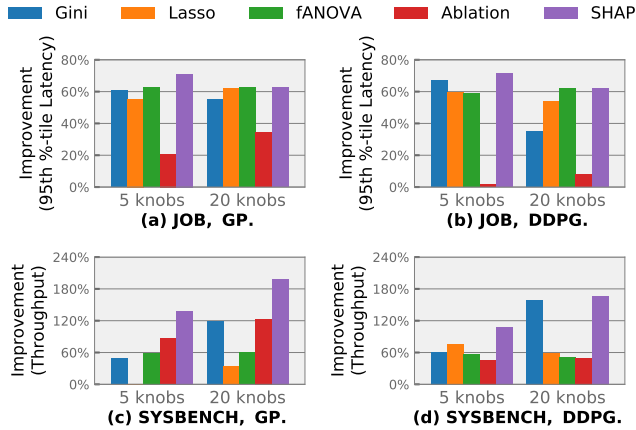
Figure 3: Performance improvement when tuning top 5/20 knobs ranked by different importance measurements.
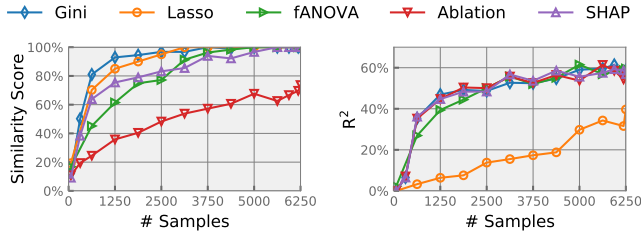

Figure 4: Sensitivity analysis for importance measurements.

and can be good enough in most cases. Therefore, the variance-based measurements will be less effective. Ablation analysis tends to yield the worst performance since it largely depends on the high-quality training samples better than the default. Among the variance-based measurements, Gini score performs the best overall, while Lasso tends to yield the worst improvement. The poor performance of Lasso may be due to its assumption of a linear or quadratic configuration space, while in reality there exist complex dependencies from configurations to database performance.

To further understand the different performance of importance measurements, we conduct sensitivity analysis on the number of training samples for SYSBENCH workload as shown in Figure 4. The samples are randomly chosen from the 6250 samples, and the final result is the average of 10 executions for each importance measurement. The y-axis in the left figure is the similarity score (intersection-over-union index [32]) of the top-5 important knobs ranked using the subset of training samples against that of the baseline (6250 samples). A larger similarity score indicates that the importance measurement is more stable since the measurement can find the important knobs with fewer observations. The right figure plots the Coefficient of Determination [31] (i.e. R^2) on the validation set. A larger R^2 indicates that the surrogate used by the measurement can better model the relationship between configurations and database performance. We have that Lasso fails to model this relationship, while it is very stable. Ablation analysis has the lowest stability, and its calculation highly depends on the high-quality samples. Gini score has the highest similarity score, indicating its sample efficiency. In the meantime, SHAP has a similarity score comparable to the variance-based measurements. Considering that SHAP has the best overall performance and comparable stability,
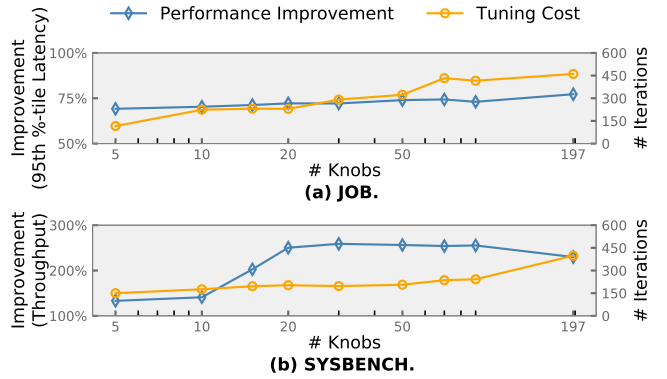

Figure 5: Performance improvement and tuning cost when increasing the number of tuning knobs.

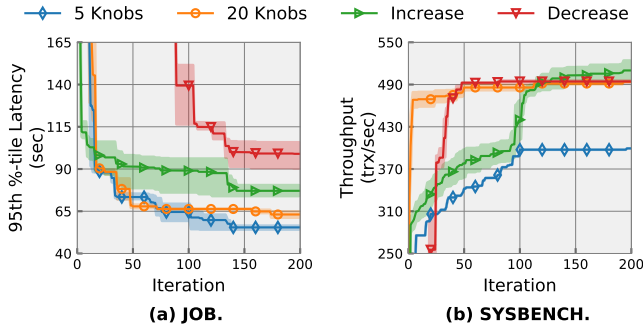we use SHAP as the default measurement for the remainder of our study.

5.3 How many knobs to choose for tuning?

We further conduct experiments to explore the effect of different numbers of tuning knobs. To measure the effect, we observe the tuning performance of vanilla BO over knob sets of different sizes ranked by SHAP. For each set, we conduct tuning for 600 iterations on SYSBENCH and JOB. We report the maximum performance improvement and the corresponding tuning costs (i.e., the iterations needed to find the configurations with the maximum improvement).

As shown in Figure 5, for JOB, the improvement is relatively stable, but the tuning cost increases when increasing the number of knobs. For SYSBENCH, the improvement is negligible at the beginning because a small number of tuning knobs have little impact on the performance. As the number of knobs grows from 5 to 20, the improvement increases from 133.10% to 250.33% since a larger configuration space leads to more tuning opportunities. Afterward, the improvement decreases as the tuning complexity increases. We conclude that it is better to tune the top-5 knobs for JOB and to tune the top-20 knobs for SYSBENCH. Generally, there is a trade-off between performance improvement and tuning cost. Given more knobs to be tuned, the improvement of the best configuration found is better, and more tuning iterations are required to find it. With a limited tuning budget, it is vital to set the number of tuning knobs to an appropriate value since a small set of knobs only has a minor impact on the database performance, and a large set would introduce excessive tuning cost.

Incremental Knob Selection. Previously, we have enumerated the number of tuning knobs and conducted extensive tuning experiments. Using such a procedure to determine the configuration space is not practical in production due to the high costs. As discussed in Section 2.3, there are two incremental tuning heuristics to determine the number of knobs: (1) increasing the number of knobs, proposed in OtterTune [3], and (2) decreasing the number of knobs, proposed in Tuneful [24]. We implement the two methods, and the details can be found in the Appendix. Figure 6 presents the experimental result. For JOB, neither increasing nor decreasing the number of knobs surpass tuning the fixed 5 knobs. In contrast, for SYSBENCH, increasing the number of knobs has better performance, but decreasing the number limits the potential performance

Figure 6: Best performance of incremental knob selection methods over iteration (for JOB, bottom left is better; for SYSBENCH, top left is better). We use top-5 knobs tuning and the top-20 knobs tuning as baselines.

gain. As previously discussed, the number of important knobs of JOB is relatively small, thus the incremental knob selection on a larger space will not bring extra benefits. Instead, on SYSBENCH, the increasing method allows optimizers to explore a smaller configuration space of the most impacting knobs before expanding to a larger space.

5.4 Main Findings

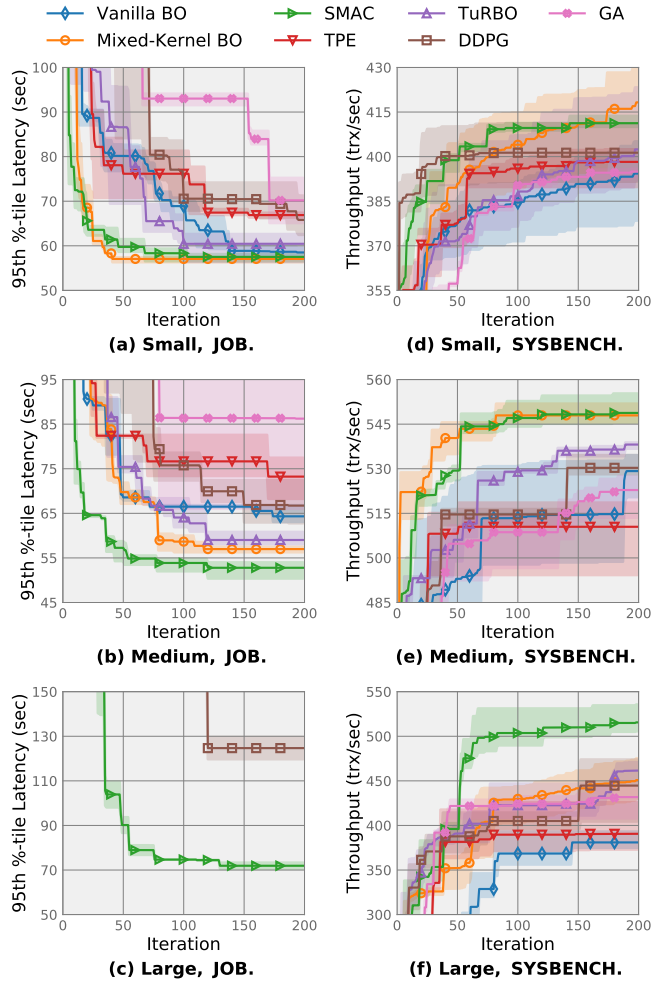Our main findings of this section are summarized as follows:

- Given a limited tuning budget, tuning over the configuration space with all the knobs is inefficient. It is recommended to pre-select important knobs to prune the configuration space.
- Configuration spaces determined by different importance measurements will impact tuning performance significantly.
- SHAP is the best importance measurements based on our evaluation. Compared with traditional measurements (i.e., Lasso and Gini score), it achieves 38.02% average performance improvement. When training samples are limited, Gini score is also effective.
- When determining the number of tuning knobs, there is a trade-off between the performance improvement and tuning cost. Increasing/decreasing the number of tuning knobs has good performances in some cases. However, how to determine the number theoretically is still an open problem with research opportunities.

6 WHICH OPTIMIZER IS THE WINNER?

In this section, we aim to find the best optimizer regarding the different sizes of configuration spaces. Furthermore, we construct two scenarios - continuous configuration space and heterogeneous configuration space to validate the optimizers' support for heterogeneity. In addition, the algorithm overheads are also studied.

6.1 Setup

According to the proprieties of black-box optimization [20], we choose three configuration spaces with different sizes: small, medium, and large, where we tune the most important 5, 20, and 197 (all) knobs ranked by SHAP, respectively. We first evaluate the six optimizers' performance over different spaces on workloads – JOB and SYSBENCH. To further validate the solutions to configuration space with heterogeneity, we focus on the well-performing optimizers and construct a comparison experiment on JOB where we

Figure 7: Best performance of optimizers over iteration (for JOB, bottom left is better; for SYSBENCH, top left is better).

use the configuration space of top-20 continuous knobs as a control group (continuous space) and the space of top-5 categorical knobs and top-15 integer knobs as a test group (heterogeneous space). In addition, we measure the execution time of recommending a configuration (i.e., algorithm overhead) of different optimizers.

6.2 Performance Comparison

6.2.1 Performance over configuration spaces of different sizes. Figure 7 presents the results, where each solid line denotes the mean of best performance across three runs, and shadows of the same color denote the quartile bar. We summarize the average ranking of optimizers in terms of the best performance they found in Table S4.

We have that SMAC achieves the best overall performance, and TPE performs worst. SMAC adopts random forest surrogate, which scales better to high dimensions and categorical input than other algorithms. TPE fails to find the optimal configuration, and the main reason could be the lack of modeling the interactions between knobs [11]. It is almost certain that the optimal values of some knobs depend on the settings of others. For instance, the tuning knobs

Table 7: Average ranking of optimizers in terms of the best performance (the bold values are the best).

Optimizer	Vanilla BO	Mixed-Kernel BO	SMAC	TPE	TuRBO	DDPG	GA
Small	4.50	**2.00**	2.83	5.17	3.33	4.33	5.83
Medium	4.33	2.17	**1.33**	6.33	3.33	4.67	5.83
Large	6.83	5.00	**1.00**	6.33	5.00	3.00	5.83
Overall	5.22	3.06	**1.72**	5.94	3.89	4.00	5.83

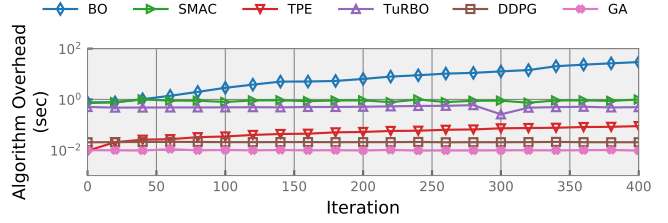**(a) Continuous Space.**   **(b) Heterogeneous Space.**

Figure 8: Comparison experiment for knobs heterogeneity (bottom left is better).

tmp_table_size and *innodb_thread_concurreny* define the maximum size of in-memory temporary tables and the maximum number of threads permitted. Intuitively, the larger number of threads running, the more in-memory temporary tables created. The relationship can be modeled by the considered optimizers, while TPE does not. In addition, the meta-heuristic method – GA also performs poorly.

Over small and medium configuration spaces, SMAC and mixed-kernel BO exhibit outstanding performance. While both adopting the Gaussian Process, mixed-kernel BO and vanilla BO have distinct performance due to their different design with the categorical knobs, which we will discuss with a comparative experiment in Section 6.2.2. In addition, DDPG has relatively bad performance on the small and medium configuration space. It learns a mapping from internal metrics (state) to configuration (action).However, given a target workload, the optimal configuration is the same for any internal metrics, leading to the fact that action and state are not necessarily related [92]. We also notice that the tuning cost of DDPG is constantly high due to the requirement of learning a great number of neural network parameters.

Over the large configuration space of JOB, only SMAC and DDPG have found the configurations better than the default latency (about 200s) within 200 iterations. The success of DDPG can be attributed to the good representation ability of the neural network to learn the high dimensional configuration space. Over the large configuration of SYSBENCH, all methods have found improved configurations, among which SMAC still performs the best. TuRBO also performs well finding the second-best configuration on SYSBENCH since its local modeling strategy avoids the over-exploration in boundaries, especially in the high dimensional space. The effectiveness of global GP methods (vanilla BO and mixed-kernel BO) further decreases when the number of tuning knobs increases.

6.2.2 Comparison Experiment for Knobs Heterogeneity. Figure 8 demonstrates the performance of SMAC, mixed-kernel BO, DDPG, and vanilla BO on continuous space and heterogeneous space, respectively. While vanilla BO and mixed-kernel BO perform similarly on continuous space, they reach quite different performances on

Figure 9: Algorithm overhead for different optimizers.

heterogeneous space. Mixed-kernel BO could find better configurations than vanilla BO and with a faster convergence speed. The reasons are as follows: vanilla BO with the RBF kernel cannot handle categorical knobs well, as it assumes a partial order between the different values of a categorical knob [81]. While mixed-kernel BO uses Hamming kernel which helps accurately understand configuration spaces. In addition, SMAC performs well over the two spaces due to its random forest modeling, and DDPG has high tuning costs for finding good configurations, as we previously discussed.

6.3 Algorithm Overhead Comparison

Algorithm overhead is the execution time taken by an optimizer to generate the next configuration to evaluate per iteration, and does not include the evaluation time. Precisely, it consists of the time for statistics collection, model fitting, and model probe [44]. Figure 9 shows the algorithm overhead of different optimizers when tuning workload - JOB over medium configuration space. GA has the lowest algorithm overhead.DDPG, SMAC, and TPE also have negligible overhead (< 1 second) in 400 iterations. However, due to the cubic scaling behavior of GP, the overhead of vanilla BO and mixed-kernel BO become extraordinarily expensive as the number of iterations increases. In particular, it takes > 10 seconds to select the next configuration after 200 iterations and > 1 minute after 400 iterations. TuRBO's overhead is comparable to SMAC. TuRBO uses a collection of simultaneous local optimizers with independent GP models instead of a global GP and terminates unpromising local optimizers, which avoids suffering from many observations.

6.4 Main findings

Our main findings of this section are summarized as follows:
- SMAC has the best overall performance and could simultaneously handle the high-dimensionality and heterogeneity of configuration space. Compared with traditional optimizer (i.e., vanilla BO, DDPG), it achieves 21.17% average performance improvement.
- TPE is worse than other optimizers in most cases since it struggles to model the interaction between knobs.
- DDPG has considerable tuning costs (i.e., more iterations) in small and medium configuration spaces due to its redundant MDP modeling and complexity of the neural network. Meanwhile, it has a relatively good performance in a large configuration space.
- On small and medium configuration spaces, SMAC and mixed-kernel BO rank the top two, while on the large configuration space, SMAC, DDPG, and TuRBO all have good performance rankings. The effectiveness of global GP methods decreases as the number of tuning knobs increases.
- Mixed-kernel BO outperforms vanilla BO in heterogeneous configuration space since the Hamming kernel design can help deal with the categorical knobs.

Table 8: Evaluation results for different transfer frameworks (the bold values are the best). We report speedup, performance enhancement (i.e., PE) against the base optimizers and the absolute performance ranking (i.e., APR).

Transfer Framework	RGPE						Workload Mapping						Fine-Tune		
Base Optimizer	Mixed-Kernel BO			SMAC			Mixed-Kernel BO			SMAC			DDPG		
Metric	Speedup	PE	APR	Speedup	PE	APR	Speedup	PE	APR	Speedup	PE	APR	Speedup	PE	APR
TPCC	**98.28**	**10.44%**	**1**	8.03	2.18%	2	×	-2.48%	4	0.35	2.43%	3	1.71	3.75%	4
SYSBENCH	**4.76**	0.53%	4	0.78	**13.32%**	**1**	×	-0.51%	5	3.08	2.23%	2	0.93	4.59%	3
Twitter	**28.42**	1.56%	**1**	28.42	0.02%	2	×	-1.70%	5	×	-0.12%	4	0.83	**3.12%**	3
Avg.	**51.52**	4.18%	2	12.41	**5.17%**	**1.67**	×	-1.56%	4.67	1.14	1.51%	3.00	1.32	3.82%	3.33

Table 9: Regression performance (bold values are the best).

Model	RF		GB		SVR		NuSVR		KNN		RR	
Metric	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2
SYSBENCH	**26.5**	**93.0%**	27.2	92.6%	97.4	5.6%	97.4	5.6%	54.6	70.2%	64.1	59.1%
JOB	11.8	97.4%	**11.1**	**97.7%**	41.7	67.9%	41.7	67.9%	27.5	86.0%	52.3	49.5%

- Global GP-based optimizers (i.e., vanilla BO and mixed-kernel BO) have cubic algorithm overhead.

7 CAN WE TRANSFER KNOWLEDGE TO SPEED UP THE TARGET TUNING TASK?

In the previous sections, different optimizers are compared from scratch without any knowledge transfer. In this section, we test whether we can utilize the historical data to speed up the target tuning task and figure out the applicability of different transfer frameworks for database configuration tuning.

7.1 Setup.

Baselines. The transfer learning framework is used to speed up base optimizers. To narrow down the candidate baselines, we evaluate workload mapping and RGPE accelerating the best-performing BO-based optimizers – SMAC and mixed-kernel BO. Then we have five baselines: Mapping (SMAC), Mapping (Mixed-Kernel BO), RGPE (SMAC), RGPE (Mixed-Kernel BO), and Fine-tune (DDPG).

Metrics. We use three metrics to evaluate the performance of transfer frameworks: performance enhancement, speedup, and absolute performance. The performance enhancement and speedup focus on whether the transfer framework can speed up the tuning process compared with non-transfer. We denote the best performance within 200 iterations for the base optimizer without transfer as $f(x^*_{base})$ and the best performance with transfer as $f(x^*_{tran})$. The performance enhancement (PE) is defined as
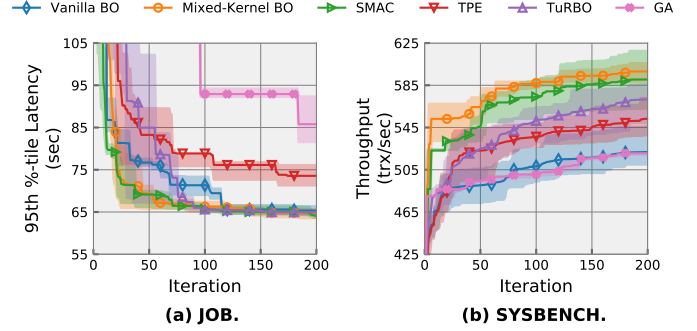
$$PE = \frac{f(x^*_{tran}) - f(x^*_{base})}{f(x^*_{base})}; \qquad (4)$$

and the speedup (η) is calculated as:

$$\eta = \frac{steps\ to\ find\ x^*_{base}\ without\ transfer}{steps\ to\ find\ config.better\ than\ x^*_{base}\ with\ transfer}. \qquad (5)$$

The absolute performance focuses on the performance of the combination of base learner and transfer framework (e.g., the best performance Mapping (SMAC) achieved within 200 iterations).

Main Procedure. We conduct experiments on three target workloads – SYSBENCH, TPC-C, and Twitter. As for the knobs we tune, we use SHAP to select top-20 impacting knobs across OLTP workloads and hardware instances listed in Table 5 and more details can be found in the Appendix. To gather historical tuning data, we collect observations from five source workloads – four transactional workloads (SEATS, Voter, TATP, and Smallbank) and one feature testing workload (SIBench). Since fine-tune (DDPG) relays on a

(a) JOB. **(b) SYSBENCH.**

Figure 10: Tuning Performance over surrogate benchmark

pre-trained model, we pre-train DDPG's network 300 iterations on the five source workloads in turn. We use DDPG's training observations as the historical data for workload mapping and RGPE frameworks. Such a setting follows the evaluation of CDBTune for data fairness. With the pre-trained models and source observations, we compare the five baselines and obtain absolute performances. To calculate performance enhancement and speedup, we also run base optimizers on the target workloads without knowledge transfer.

7.2 Performance Comparison

Table 8 shows the experimental results. If a baseline fails to find configurations better than x^*_{base}, we will put "×" in the speedup column. We observe that fine-tune and workload mapping may hinder the optimization (i.e., "negative transfer") in some cases where the speedup is smaller than one or the improvement is negative. Workload mapping always maps a similar workload and combines its observations in the surrogate model together with the target observations, which may be problematic since the source workloads may not be entirely identical to the target one. RGPE solves this problem by assigning adaptive weights to source surrogates and utilizing them discriminately. As for the fine-tuning, the performance is not stable since the neural network may over-fit the source workloads, and fine-tuning the over-fitted network may be less efficient than training from scratch. Overall, the RGPE framework has the best performance. When accelerating the base optimizer – mixed-kernel BO, RGPE has shown impressive speedup, which may come from the inferior performance of mixed-kernel BO compared with SMAC. In terms of absolute performance, RGPE (SMAC) achieves the best overall performance.

8 EFFICIENT DATABASE TUNING BENCHMARK VIA SURROGATES

As discussed previously, to ease the burden of evaluating tuning optimizers, we propose to benchmark the database tuning via surrogate models that approximate expensive evaluation through cheap and stable model predictions. A user can easily test optimizers by

interacting with the surrogate models (i.e., input the configuration suggested by the optimizer and output the corresponding database performance). In this section, we present the construction of the tuning benchmarks. More details can be found in the Appendix.

To collect the training data, we run existing database optimizers to densely sample high-performance regions of the configuration space and sample poorly-performing regions [21] via LHS. Since there is significant performance fluctuation among different instances of the same type [4], we collect the data within one instance in order to obtain a consistent performance measurement for configurations. For each configuration space, we collect 6250 samples. It takes about 13 days to gather the data for one configuration space, and we focus on the small configuration space of JOB and medium space of SYSBENCH. We consider a broad range of commonly used regression models as candidates for the surrogates, including Random Forest (RF), Gradient Boosting (GB), Support Vector Regression (SVR), NuSVR, k-nearest-neighbours (KNN), Ridge Regression (RR). We evaluate their performance via 10-fold cross-validation, and Table 9 presents the resulted mean squared error (RMSE) and R^2. We have that the two tree-based models, RF and GB, perform the best. Since RFs are widely used with simplicity, we adopt RF as the surrogate for the tuning benchmark.

Figure 10 depicts the best performance found by different optimizers using the tuning benchmark based on RFs. We report means and quartiles across ten runs of each optimizer. We observe that our tuning benchmark yields evaluation results closer to the result in Figure 10 – SMAC and mixed-kernel BO have the best overall performance. In addition, the experiments on our tuning benchmarks are much faster. For example, as previously mentioned, a single function evaluation on SYSBENCH workload requires at least 3 minutes, while a surrogate evaluation needs 0.08 seconds on average. When considering the algorithm overhead of optimizers, the previous 200-iteration experiment takes at least 10 hours, while the same experiment on the tuning benchmark takes about 2~4 minutes. The tuning benchmark brings 150~311 × speedups. To further benchmark RL-based optimizers, we have to train a surrogate to learn the state transaction (i.e., internal metrics of DBMS). We believe that the idea of using surrogates to replace the expensive evaluations can be adopted extensively in the other modules if we can collect diverse and adequate training data. For example, we can fit a surrogate with all the knobs as input to benchmark the *knob selection* modules. We leave such ideas as a goal for future work.

9 DISCUSSION

We summarize our answers to the motivating questions and discuss the research opportunities we draw from the evaluations.

9.1 Answers to The Motivating Questions

A1: We recommend the tunability-based method – SHAP as an importance measurement since it indicates the necessity of tuning a knob when the DBMS has relatively reasonable default knob values. We can determine the number of knobs to tune by exhaustive and expensive enumeration. How to determine the number theoretically with fewer trials is still an open problem. In practice, with a limited tuning budget, we can resort to incremental tuning.

A2: When comparing different optimizers, we need to consider the size and composition of configuration space. SMAC and DDPG are recommended for high-dimensional configuration spaces, although it is better to first conduct knob selection to prone the configuration space. SMAC is the winner that can handle the high high-dimensionality and heterogeneity of configuration space.

A3: For RL-based optimizers, we can fine-tune its pre-trained model to adapt to the target workload. However, we find that it might suffer from the negative transfer issue empirically. For BO-based optimizers, RGPE exhibits excellent speedup and improvement since it avoids negative transfer via adaptive weight assignment.

In summary, we have that using SHAP measurement to prone the unimportant knobs and adopting SMAC optimizer in the RGPE transfer framework could reach the best end-to-end performance.

9.2 Research Opportunities

An End-to-End Optimization for Designing Database Tuning Systems. The *end-to-end optimization* can be viewed as optimizing over a joint search space, including the selection of importance measurements, knobs, optimizers, and transfer frameworks. Because of the joint nature, the search space of the *end-to-end optimization* is complex and huge. In our evaluation, we decompose the joint space reasonably to narrow the search space. For example, when comparing the importance measurements, we fix the optimizers to the two representative ones, vanilla BO and DDPG. Meanwhile, a class of methods in the HPO field treats the selection of algorithms as a new hyper-parameter to optimize. They optimize over the joint search space with probabilistic models [25, 28, 43, 48, 51], which could be another research direction.

Tuning Budget Allocation. How to allocate a limited tuning budget between different modules (e.g., knob selection and knobs optimization) is a problem that needs exploring [27, 83]. For example, an accurate ranking of knobs can facilitate later optimization but comes with the cost of collecting extensive training samples. There is a trade-off between the budgets for sample collection and the latter optimization. In addition, as discussed, a larger configuration space gives us more tuning opportunities but with higher tuning costs. There still remains space when determining the number of tuning knobs wisely, given a limited tuning budget.

10 CONCLUSION

Given emerging new designs and algorithms for configuration tuning systems of DBMS, we are curious about the best solution in different scenarios. In this paper, we decompose existing systems into three modules and comprehensively analyze and evaluate the corresponding intra-algorithms to construct optimal design "paths" in various scenarios. Meanwhile, we identify the design trade-offs to suggest insightful principles and promising research directions to optimize the tuning systems. In addition, we propose an efficient database tuning benchmark that reduces the evaluation overhead to a minimum, facilitating the evaluation and analysis for new algorithms with fewer costs. It is noted that we do not restrict our evaluation within the database community and extensively evaluate promising approaches from the HPO field. Our evaluation demonstrates that such an out-of-the-box manner can further enhance the performance of database configuration tuning systems.

REFERENCES

[1] Sanjay Agrawal, Surajit Chaudhuri, Lubor Kollár, Arunprasad P. Marathe, Vivek R. Narasayya, and Manoj Syamala. 2004. Database Tuning Advisor for Microsoft SQL Server 2005. In *VLDB*. Morgan Kaufmann, 1110–1121.

[2] Dana Van Aken, Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudré-Mauroux. 2015. BenchPress: Dynamic Workload Control in the OLTP-Bench Testbed. In *SIGMOD Conference*. ACM, 1069–1073.

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

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

[5] Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. 2020. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *NeurIPS*.

[6] Juan Cruz Barsce, Jorge Palombarini, and Ernesto C. Martínez. 2017. Towards autonomous reinforcement learning: Automatic setting of hyper-parameters using Bayesian optimization. In *CLEI*. IEEE, 1–9.

[7] Syrine Belakaria, Aryan Deshwal, and Janardhan Rao Doppa. 2019. Max-value Entropy Search for Multi-Objective Bayesian Optimization. In *NeurIPS*. 7823–7833.

[8] R. E. Bellman. 1957. A Markov decision process. *Journal of Mathematical Fluid Mechanics* 6 (1957).

[9] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. In *NIPS*. 2546–2554.

[10] James Bergstra and David D. Cox. 2013. Hyperparameter Optimization and Boosting for Classifying Facial Expressions: How good can a "Null" Model be? *CoRR* abs/1306.3476 (2013).

[11] James Bergstra, Daniel Yamins, and David D. Cox. 2013. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. In *ICML (1) (JMLR Workshop and Conference Proceedings)*, Vol. 28. JMLR.org, 115–123.

[12] Andre Biedenkapp, Marius Lindauer, Katharina Eggensperger, Frank Hutter, Chris Fawcett, and Holger H. Hoos. 2017. Efficient Parameter Importance Analysis via Ablation with Surrogates. In *AAAI*. AAAI Press, 773–779.

[13] Leo Breiman. 2001. Random Forests. *Mach. Learn.* 45, 1 (2001), 5–32.

[14] Meeyoung Cha, Hamed Haddadi, Fabrício Benevenuto, and P. Krishna Gummadi. 2010. Measuring User Influence in Twitter: The Million Follower Fallacy. In *ICWSM*. The AAAI Press.

[15] Surajit Chaudhuri and Vivek R. Narasayya. 2007. Self-Tuning Database Systems: A Decade of Progress. In *VLDB*. ACM, 3–14.

[16] Surajit Chaudhuri and Gerhard Weikum. 2006. Foundations of Automated Database Tuning. In *VLDB*. ACM, 1265.

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

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

[19] Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. 2004. Least angle regression. *The Annals of statistics* 32, 2 (2004), 407–499.

[20] Katharina Eggensperger. 2013. Towards an Empirical Foundation for Assessing Bayesian Optimization of Hyperparameters.

[21] Katharina Eggensperger, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2015. Efficient Benchmarking of Hyperparameter Optimizers via Surrogates. In *AAAI*. AAAI Press, 1114–1120.

[22] David Eriksson, Michael Pearce, Jacob R. Gardner, Ryan Turner, and Matthias Poloczek. 2019. Scalable Global Optimization via Local Bayesian Optimization. In *NeurIPS*. 5497–5508.

[23] Chris Fawcett and Holger H. Hoos. 2016. Analysing differences between algorithm configurations through ablation. *J. Heuristics* 22, 4 (2016), 431–458.

[24] Ayat Fekry, Lucian Carata, Thomas F. J.-M. Pasquier, Andrew Rice, and Andy Hopper. 2020. To Tune or Not to Tune?: In Search of Optimal Configurations for Data Analytics. In *KDD*. ACM, 2494–2504.

[25] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and Robust Automated Machine Learning. In *NIPS*. 2962–2970.

[26] Matthias Feurer, Benjamin Letham, and Eytan Bakshy. 2018. Scalable meta-learning for bayesian optimization using ranking-weighted gaussian process ensembles. In *AutoML Workshop at ICML*, Vol. 7.

[27] Yaping Fu, Hui Xiao, Loo Hay Lee, and Min Huang. 2021. Stochastic optimization using grey wolf optimization with optimal computing budget allocation. *Appl. Soft Comput.* 103 (2021), 107154.

[28] Chen Gao, Quanming Yao, Depeng Jin, and Yong Li. 2021. Efficient Data-specific Model Search for Collaborative Filtering. In *KDD*. ACM, 415–425.

[29] Marius Geitle and Roland Olsson. 2019. A New Baseline for Automated Hyper-Parameter Optimization. In *LOD (Lecture Notes in Computer Science)*, Vol. 11943.

[30] David Gonzalez-Cuautle, Uriel Yair Corral-Salinas, Gabriel Sanchez-Perez, Héctor Pérez-Meana, Karina Toscano-Medina, and Aldo Hernandez-Suarez. 2019. An Efficient Botnet Detection Methodology using Hyper-parameter Optimization Trough Grid-Search Techniques. In *IWBF*. IEEE, 1–6.

[31] Richard F. Gunst. 1999. Applied Regression Analysis. *Technometrics* 41, 3 (1999), 265–266.

[32] M. H. Heine. 1973. Distance between sets as an objective measure of retrieval effectiveness. *Inf. Storage Retr.* 9, 3 (1973), 181–198.

[33] Tobias Hinz, Nicolás Navarro-Guerrero, Sven Magg, and Stefan Wermter. 2018. Speeding up the Hyperparameter Optimization of Deep Convolutional Neural Networks. *Int. J. Comput. Intell. Appl.* 17, 2 (2018), 1850008:1–1850008:15.

[34] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential Model-Based Optimization for General Algorithm Configuration. In *LION (Lecture Notes in Computer Science)*, Vol. 6683. Springer, 507–523.

[35] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2014. An Efficient Approach for Assessing Hyperparameter Importance. In *ICML (JMLR Workshop and Conference Proceedings)*, Vol. 32. JMLR.org, 754–762.

[36] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren (Eds.). 2019. *Automatic Machine Learning: Methods, Systems, Challenges*. Springer.

[37] Donald R. Jones, Matthias Schonlau, and William J. Welch. 1998. Efficient Global Optimization of Expensive Black-Box Functions. *J. Glob. Optim.* 13, 4 (1998), 455–492.

[38] Hyungsoo Jung, Hyuck Han, Alan D. Fekete, and Uwe Röhm. 2011. Serializable Snapshot Isolation for Replicated Databases in High-Update Scenarios. *Proc. VLDB Endow.* 4, 11 (2011), 783–794.

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

[40] Beomjoon Kim, Kyungjae Lee, Sungbin Lim, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. 2020. Monte Carlo Tree Search in Continuous Spaces Using Voronoi Optimistic Optimization with Regret Bounds. In *AAAI*. AAAI Press, 9916–9924.

[41] Aaron Klein. 2017. RoBO : A Flexible and Robust Bayesian Optimization Framework in Python.

[42] Jan Kossmann and Rainer Schlosser. 2020. Self-driving database systems: a conceptual approach. *Distributed Parallel Databases* 38, 4 (2020), 795–817.

[43] Lars Kotthoff, Chris Thornton, Holger H. Hoos, Frank Hutter, and Kevin Leyton-Brown. 2017. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *J. Mach. Learn. Res.* 18 (2017), 25:1–25:5.

[44] Mayuresh Kunjir and Shivnath Babu. 2020. Black or White? How to Develop an AutoTuner for Memory-based Analytics. In *SIGMOD Conference*. ACM, 1667–1683.

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

[46] Stefan Lessmann, Robert Stahlbock, and Sven F Crone. 2005. Optimizing hyperparameters of support vector machines by genetic algorithms.. In *IC-AI*. 74–82.

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

[48] Yang Li, Jiawei Jiang, Jinyang Gao, Yingxia Shao, Ce Zhang, and Bin Cui. 2020. Efficient Automatic CASH via Rising Bandits. In *AAAI*. AAAI Press, 4763–4771.

[49] Yang Li, Yu Shen, Jiawei Jiang, Jinyang Gao, Ce Zhang, and Bin Cui. 2021. MFES-HB: Efficient Hyperband with Multi-Fidelity Quality Measurements. *Proceedings of the AAAI Conference on Artificial Intelligence* 35, 10 (May 2021), 8491–8500.

[50] Yang Li, Yu Shen, Wentao Zhang, Yuanwei Chen, Huaijun Jiang, Mingchao Liu, Jiawei Jiang, Jinyang Gao, Wentao Wu, Zhi Yang, Ce Zhang, and Bin Cui. 2021. OpenBox: A Generalized Black-box Optimization Service. In *KDD*. ACM, 3209–3219.

[51] Yang Li, Yu Shen, Wentao Zhang, Jiawei Jiang, Bolin Ding, Yaliang Li, Jingren Zhou, Zhi Yang, Wentao Wu, Ce Zhang, and Bin Cui. 2021. VolcanoML: Speeding up End-to-End AutoML via Scalable Search Space Decomposition. *Proc. VLDB Endow.* 14 (2021), 2167–2176.

[52] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning. In *ICLR (Poster)*.

[53] Scott M. Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *NIPS*. 4765–4774.

[54] Gang Luo. 2016. A review of automatic selection methods for machine learning algorithms and hyper-parameter values. *Netw. Model. Anal. Health Informatics Bioinform.* 5, 1 (2016), 18.

[55] Lin Ma, Dana Van Aken, Ahmed Hefny, Gustavo Mezerhane, Andrew Pavlo, and Geoffrey J. Gordon. 2018. Query-based Workload Forecasting for Self-Driving Database Management Systems. In *SIGMOD Conference*. ACM, 631–645.

[56] Vasilis Maglogiannis, Dries Naudts, Adnan Shahid, and Ingrid Moerman. 2018. A Q-Learning Scheme for Fair Coexistence Between LTE and Wi-Fi in Unlicensed Spectrum. *IEEE Access* 6 (2018), 27278–27293.

[57] Mohit Malu, Gautam Dasarathy, and Andreas Spanias. 2021. Bayesian Optimization in High-Dimensional Spaces: A Brief Survey. In *IISA*. IEEE, 1–8.

[58] Kaspar Märtens, Michalis K. Titsias, and Christopher Yau. 2019. Augmented Ensemble MCMC sampling in Factorial Hidden Markov Models. In *AISTATS (Proceedings of Machine Learning Research)*, Vol. 89. PMLR, 2359–2367.

[59] Ruben Martinez-Cantin. 2014. BayesOpt: a Bayesian optimization library for nonlinear optimization, experimental design and bandits. *J. Mach. Learn. Res.* 15, 1 (2014), 3735–3739.

[60] Camille Maurice, Francisco Madrigal, and Frédéric Lerasle. 2017. Hyper-Optimization tools comparison for parameter tuning applications. In *AVSS*. IEEE Computer Society, 1–6.

[61] Michael D. McKay. 1992. Latin Hypercube Sampling as a Tool in Uncertainty Analysis of Computer Models. In *WSC*. ACM Press, 557–564.

[62] Bjoern H. Menze, B. Michael Kelm, Ralf Masuch, Uwe Himmelreich, Peter Bachert, Wolfgang Petrich, and Fred A. Hamprecht. 2009. A comparison of random forest and its Gini importance with standard chemometric methods for the feature selection and classification of spectral data. *BMC Bioinform.* 10 (2009).

[63] Stefano Nembrini, Inke R. König, and Marvin N. Wright. 2018. The revival of the Gini importance? *Bioinform.* 34, 21 (2018), 3711–3718.

[64] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS*. 8024–8035.

[65] Andy Pavlo. 2021. Make Your Database System Dream of Electric Sheep: Towards Self-Driving Operation. *Proc. VLDB Endow.* 14, 12 (2021), 3211–3221.

[66] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* 12 (2011), 2825–2830.

[67] Philipp Probst, Anne-Laure Boulesteix, and Bernd Bischl. 2019. Tunability: Importance of Hyperparameters of Machine Learning Algorithms. *J. Mach. Learn. Res.* 20 (2019), 53:1–53:32.

[68] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. 2017. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*. PMLR, 2902–2911.

[69] Matthias W. Seeger. 2004. Gaussian Processes For Machine Learning. *Int. J. Neural Syst.* 14, 2 (2004), 69–106.

[70] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. 2016. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proc. IEEE* 104, 1 (2016), 148–175.

[71] Dennis E. Shasha and Philippe Bonnet. 2002. Database Tuning: Principles, Experiments, and Troubleshooting Techniques. In *VLDB*. Morgan Kaufmann.

[72] Dennis E. Shasha and Steve Rozen. 1992. Database Tuning. In *VLDB*. Morgan Kaufmann, 313.

[73] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *NIPS*. 2960–2968.

[74] Adam J. Storm, Christian Garcia-Arellano, Sam Lightstone, Yixin Diao, and Maheswaran Surendra. 2006. Adaptive Self-tuning Memory in DB2. In *VLDB*. ACM, 1081–1092.

[75] Erik Strumbelj and Igor Kononenko. 2014. Explaining prediction models and individual predictions with feature contributions. *Knowl. Inf. Syst.* 41, 3 (2014), 647–665.

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

[77] Vladimir Svetnik, Andy Liaw, Christopher Tong, J. Christopher Culberson, Robert P. Sheridan, and Bradley P. Feuston. 2003. Random Forest: A Classification and Regression Tool for Compound Classification and QSAR Modeling. *J. Chem. Inf. Comput. Sci.* 43, 6 (2003), 1947–1958.

[78] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In *KDD*. ACM, 847–855.

[79] Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)* 58, 1 (1996), 267–288.

[80] Jan N. van Rijn and Frank Hutter. 2018. Hyperparameter Importance Across Datasets. In *KDD*. ACM, 2367–2376.

[81] Xingchen Wan, Vu Nguyen, Huong Ha, Bin Xin Ru, Cong Lu, and Michael A. Osborne. 2021. Think Global and Act Local: Bayesian Optimisation over High-Dimensional Categorical and Mixed Search Spaces. In *ICML (Proceedings of Machine Learning Research)*, Vol. 139. PMLR, 10663–10674.

[82] Hao Wang, Yitan Lou, and Thomas Bäck. 2019. Hyper-Parameter Optimization for Improving the Performance of Grammatical Evolution. In *CEC*. IEEE, 2649–2656.

[83] Tianxiang Wang, Jie Xu, and Jian-Qiang Hu. 2021. A Study on Efficient Computing Budget Allocation for a Two-Stage Problem. *Asia Pac. J. Oper. Res.* 38, 2 (2021), 2050044:1–2050044:20.

[84] Zi Wang, Clement Gehring, Pushmeet Kohli, and Stefanie Jegelka. 2018. Batched Large-scale Bayesian Optimization in High-dimensional Spaces. In *AISTATS (Proceedings of Machine Learning Research)*, Vol. 84. PMLR, 745–754.

[85] Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando de Freitas. 2016. Bayesian Optimization in a Billion Dimensions via Random Embeddings. *J. Artif. Intell. Res.* 55 (2016), 361–387.

[86] Ziyu Wang, Babak Shakibi, Lin Jin, and Nando de Freitas. 2014. Bayesian Multi-Scale Optimistic Optimization. In *AISTATS (JMLR Workshop and Conference Proceedings)*, Vol. 33. JMLR.org, 1005–1014.

[87] Hilde J. P. Weerts, Andreas C. Mueller, and Joaquin Vanschoren. 2020. Importance of Tuning Hyperparameters of Machine Learning Algorithms. *CoRR* abs/2007.07588 (2020).

[88] Gerhard Weikum, Axel Mönkeberg, Christof Hasse, and Peter Zabback. 2002. Self-tuning Database Technology and Information Services: from Wishful Thinking to Viable Engineering. In *VLDB*. Morgan Kaufmann, 20–31.

[89] Li Yang and Abdallah Shami. 2020. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing* 415 (2020), 295–316.

[90] Steven R Young, Derek C Rose, Thomas P Karnowski, Seung-Hwan Lim, and Robert M Patton. 2015. Optimizing deep learning hyper-parameters through an evolutionary algorithm. In *Proceedings of the workshop on machine learning in high-performance computing environments*. 1–5.

[91] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, Minwei Ran, and Zekang Li. 2019. An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning. In *SIGMOD Conference*. ACM, 415–432.

[92] Xinyi Zhang, Hong Wu, Zhuo Chang, Shuowei Jin, Jian Tan, Feifei Li, Tieying Zhang, and Bin Cui. 2021. ResTune: Resource Oriented Tuning Boosted by Meta-Learning for Cloud Databases. In *SIGMOD Conference*. ACM, 2102–2114.

Supplemental Material

S1 MORE BACKGROUND FOR EVALUATING CONFIGURATION TUNING SYSTEMS

The previous evaluation for database configuration tuning is limited to a subset of existing systems where the analysis and evaluation of intra-algorithm components are ignored. Instead, we identify three key modules of configuration tuning systems and conduct a thorough analysis and experimental evaluation from a micro perspective (i.e., evaluating every fine-grained algorithms). Figure S1 presents the fine-grained algorithms adopted by existing database tuning systems or from the HPO field. When conducting database configuration tuning in practice, we have to chose a solution "path" across the three modules: (1) *knob selection*, (2) *configuration optimization*, and (3) *knowledge transfer*, as shown in the figure. Each *knob selection* algorithm could determine an unique configuration space and can be "linked" to any of the *configuration optimization* algorithms (i.e., optimizers). And among the optimizers, all the BO-based optimizers assuming a Gaussian model (SMAC, vanilla BO, mixed-kernel BO, TurBO) can be "linked" to workload mapping or RGPE transfer frameworks. And the DDPG algorithm is "linked" to fine-tune framework. We have noted that existing systems only cover a part of the possible solutions and it remains unclear to identify the best "path" for database configuration tuning. We evaluate all the fine-grained algorithms listed in Figure S1 and carefully decompose the search space to identify the best "path" in various scenarios.

S2 DETAILS ABOUT INTRA-ALGORITHMS

In this section, we present details about the intra-algorithms which we describe on a high level in the paper due to space limitations.

S2.1 Ablation Analysis

Ablation analysis [12] selects the features whose changes contribute the most to improve the performance of configurations We now describe how ablation analysis quantifies the performance change due to a certain feature's change. Given a default configuration $\theta_{default}$ and a target configuration θ_{target} (usually a better one), ablation analysis first computes the feature differences $\Delta(\theta_{default}, \theta_{target})$ between the default and target configurations. Next, an ablation path $\theta_{default}, \theta_1, \theta_2, \ldots, \theta_{target}$ is iteratively constructed. In each iteration i with previous ablation path configuration θ_{i-1}, we consider all remaining feature changes $\delta \in \Delta(\theta_{default}, \theta_{target})$ and apply the change to the previous ablation path configuration and obtain the candidate $\theta_{i-1}[\delta]$. Each parameter change δ is a modification of one feature from its value in θ_{i-1} to its value in θ_{target}, along with any other feature modifications that may be necessary due to conditionally constraints in Θ. The next configuration on the ablation path θ_i is the candidate $\theta_{i-1}[\delta]$ with the best objective performance f. The performance evaluation is approximated via surrogate for efficiency reasons. The order that the feature is changed is the importance rank from the ablation analysis. Given a set of observations, we conduct ablation analysis between the default configuration and the configurations with better performance than the default in the observation set (i.e., target configurations). For each feature, we use the average rank from each ablation path as the final ranking of importance.

S2.2 SHAP

SHAP [53] (SHapley Additive exPlanation) uses Shapley values of a conditional expectation function of the original model. SHAP values attribute to each feature the change in the expected model prediction when conditioning on that feature. They explain how to get from the base value that would be predicted if we did not know any features to the current output. When the model is non-linear or the input features are not independent, the order in which features are added to the expectation matters, and the SHAP values arise from averaging the contributing values across all possible orderings [53]. The exact computation of SHAP values is challenging, which can be estimated by Shapley sampling values method [75] or Kernel SHAP method [53].

S2.3 SMAC

SMAC [34] constructs a random forest as a set of regression trees, each of which is built on n data points randomly sampled with repetitions from the entire training data set. It computes the random forest's predictive mean $\hat{\mu}(\theta)$ and variance $\hat{\sigma}^2(\theta)$ for a new configuration θ as the empirical mean and variance of for the Gaussian distribution . SMAC uses the random forest model to select a list of promising parameter configurations. To quantify how promising a configuration θ is, it uses the model's predictive distribution for θ to compute its expected positive improvement $EI(\theta)$ [37] over the best configuration seen so far. $EI(\theta)$ is large for configurations θ with high predicted performance and for those with high predicted uncertainty; thereby, it offers an automatic trade-off between
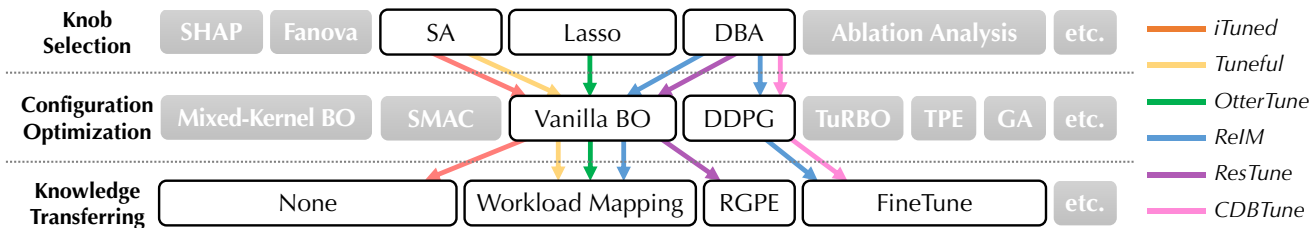

Figure S1: Detailed Intra-algorithm Designs of Configuration Tuning Systems (The Full Picture): Black boxes denote the algorithms adopted by existing database tuning systems (indicated by colored paths), and grey boxes denote the algorithms in the HPO field. SA denotes sensitivity analysis based on Gini score and GA denotes Genetic algorithm.

Table S1: Scale factors of workload from OLTP-Bench

Workload	TPCC	Twitter	Smallbank	SIBench	Voter	Seats	TATP
Scale Factor	200	1500	10	1000	10000	50	100

exploitation (focusing on known good parts of the space) and exploration (gathering more information in unknown parts of the space). To gather a set of promising configurations with low computational overhead, SMAC performs a simple multi-start local search and considers all resulting configurations with locally maximal *EI*.

S2.4 RGPE

RGPE [26] is a scalable meta-learning framework to accelerate BO-based optimizer. First, for each previous tuning task T_i, it trains a base Gaussian process (GP) model M_i on the corresponding observations from H_i. Then it builds a surrogate model M_{meta} combine the base GP models, instead of the original surrogate M_T fitted on the observations H_T of the target task only. The prediction of M_{meta} at point θ is given by:

$$y \sim N(\sum_i w_i \mu_i(\theta), \sum_i w_i \sigma_i^2(\theta)), \tag{6}$$

where w_i is the weight of base surrogate M_i, and μ_i and σ_i^2 are the predictive mean and variance from base surrogate M_i. The weight w_i reflects the similarity between the previous task and the current task. Therefore, M_{meta} carries the knowledge of tuning on previous tasks, which can greatly accelerate the convergence of the tuning in the target task. We then use the following ranking loss function L, i.e., the number of misranked pairs, to measure the similarity between previous tasks and the target task:

$$L(M_j, H_T) = \sum_{j=1}^{n_t} \sum_{k=1}^{n_t} \mathbb{1}\Big((M_i(\theta_j) \le M_i(\theta_k)) \oplus (y_j \le y_k)\Big) \tag{7}$$

where $\oplus$ is the exclusive-or operator, n_t denotes the number of tuning tasks and $M_i(\theta_j$ means the prediction of M_i on configuration θ. Based on the ranking loss function, the weight w_i is set to the probability that M_i has the smallest ranking loss on H_T, that is, $w_i = \mathbb{P}(i = \arg\min_j L(M_j, H_T))$. This probability can be estimated using MCMC sampling [58].

S3 MORE DETAILS ABOUT WORKLOADS

While we have presented general profile information for workloads in Table 4 in the paper, we detailed the implementation and the reason we select those workloads in this section.

Implementation of The Workloads. For JOB, we use the same setup illustrated in [45]. For SYSBENCH, we load 150 tables each of which contains 800000 rows, and adopt the read-write mode. For workloads from OLTP-Bench, we use the scale factor to determine the data size (e.g., the number of warehouses in TPCC) as shown in Table S1. In addition, the parameter `terminal` is set to 64 for each workload. We keep other parameters as the default value as OLTP-Bench provided, including `isolation` and `weights` of transactions.

The Reasons for Workload Selection. Our evaluations analyze the tuning performance over OLTP and OLAP senarios. We mainly use an OLAP workload – JOB and an OLTP workload – SYSBENCH.

This is due to that the two workloads are most used in the evaluation of database configuration methods [47, 91] and cover the scenarios of an online transaction/analytical processing. To construct the knowledge transfer scenario and to recommend the important knobs across workloads, we use the eight OLTP workloads (i.e., SYSBENCH, TPC-C, Twitter, SEATS, Smallbank, TATP, Voter, SIBENCH) and configure them with various sizes, read-write ratios as shown in Table 4. When conducting knowledge transfer experiments, we choose three workloads – SYSBENCH, Twitter, TPC-C as the target tuning workloads, and leave the other workloads as diverse source workloads.

S4 MORE DETAILS AND RESULTS ABOUT EXPERIMENT

In this section, we present additional experimental details.

S4.1 Knob Selection

Importance Measurements Comparison Results. While we have presented the overall rank of each importance measurements in Table 6 in the paper, we provide detailed ranking and performance improvements in Table S2 on JOB and SYSBENCH workload respectively. We observe that that SHAP achieves first place in 7 of 8 cases (indicated by the bold values).

Incremental Knob Selection. We implement the same incremental tuning setting as OtterTune and Tuneful, except we decrease the knobs every 20 iterations instead of 10 for Tuneful since we have more tuning budget. Ottertune begins with tuning the top four knobs and increases the number by two every four iterations. Tuneful begins with tuning all the knobs and decreases the number by 40% every 10 iterations.

Top impacting knobs with high tunability for OLTP workloads. Although most optimizations in the paper are over the impacting knobs for SYSBENCH and JOB, we further conduct an experiment using SHAP to generate a ranking of the most impacting knobs across OLTP workloads and hardware instances listed in Table 5. And we use this ranking to conduct an evaluation for *knowledge transfer* component across OLTP workloads in Section 7 in the paper. We use the seven OLTP workloads listed in Table 4 in the paper and perform LHS to collect 1250 samples for each workload on the four hardware instances listed in Table 5. Then we adopt SHAP to generate a ranking respectively and count the number of times that each knob appears in the top 10 of all rankings to measure their overall importance. Table S3 shows the top-10 important knobs and their brief description. We believe this ranking provides database practitioners guidance for choosing the knobs to tune. Dynamic variables can be changed at runtime using the SET statement, while others can only be set at server startup using options on the command line or in an option file. We have that configuring the maximum number of threads and the size of the log file can contribute to the performance the most, which is aligned with the previous analysis [3, 4, 92]. We note that six of them are related to memory allocation, which indicates that the default setting of memory allocation may not be appropriate across the workloads and hardware. We leave the important knobs ranking for OLAP workloads as future work, as there are fewer OLAP workloads suitable for database tuning tasks, except JOB and TPC-H.

Table S2: Performance improvement and ranking when tuning top 5/20 knobs ranked by different importance measurements. We calculate the performance improvement (i.e., %imp) against based on the default performance. VBO denotes vanilla BO. Bold values are the best.

Measurements			Gini		Lasso		Fanova		Ablation		SHAP	
			% Imp.	Rank	% Imp.	Rank	% Imp.	Rank	% Imp.	Rank	% Imp.	Rank
5 knobs	JOB	VBO	60.81	3	55.38	4	62.65	2	20.82	5	**71.10**	**1**
		DDPG	66.99	2	59.23	3	58.58	4	1.70	5	**71.28**	**1**
	SYSBENCH	VBO	48.98	4	1.24	5	59.52	3	86.20	2	**137.97**	**1**
		DDPG	61.03	3	75.92	2	57.83	4	45.71	5	**107.83**	**1**
20 knobs	JOB	VBO	54.94	4	62.17	3	62.53	2	34.21	5	**62.67**	**1**
		DDPG	34.69	4	53.71	3	**62.12**	**1**	7.86	5	61.71	2
	SYSBENCH	VBO	119.19	3	34.20	5	60.26	4	123.49	2	**198.36**	**1**
		DDPG	159.37	2	59.10	3	51.57	4	50.17	5	**166.23**	**1**
	Overall			3.13		3.50		3.00		4.25		**1.13**

Table S3: The Top-10 impacting knobs with high tunability for OLTP workloads

Knob	Type	Dynamic	Module	Description
innodb_thread_concurrency	Integer	Yes	Concurrency	The maximum number of threads permitted inside of InnoDB.
innodb_log_file_size	Integer	No	Logging	The size in bytes of each log file in a log group.
max_allowed_packet	Integer	Yes	Replication	The upper limit on the size of any single message between the MySQL server and clients.
innodb_io_capacity_max	Integer	Yes	IO	The maximum number of IOPS performed by InnoDB background tasks
tmp_table_size	Integer	Yes	Memory	The maximum size of internal in-memory temporary tables.
query_prealloc_size	Integer	Yes	Memory	The size in bytes of the persistent buffer used for statement parsing and execution.
max_heap_table_size	Integer	Yes	Memory	The maximum size to which user-created memory tables are permitted to grow.
innodb_doublewrite	Categorical	No	Memory	Whether the doublwrite buffer is enabled.
transaction_alloc_block_size	Interger	Yes	Memory	The amount in bytes by which to increase a per-transaction memory pool which needs memory.
join_buffer_size	Interger	Yes	Memory	The minimum size of the buffer that is used for joins.

Table S4: Average ranking of optimizers in terms of the best performance they found.Bold values are the best.

Optimizer		Vanilla BO	Mixed-Kernel BO	SMAC	TPE	TURBO	DDPG	GA
Small Configuration Space	JOB	4.00	**2.00**	2.67	5.33	3.00	4.33	6.67
	SYSBENCH	5.00	**2.00**	3.00	5.00	3.67	4.33	5.00
	Average	4.50	**2.00**	2.83	5.17	3.33	4.33	5.83
Medium Configuration Space	JOB	4.33	2.67	**1.00**	6.67	3.00	4.33	6.00
	SYSBENCH	4.33	**1.67**	1.67	6.00	3.67	5.00	5.67
	Average	4.33	2.17	**1.33**	6.33	3.33	4.67	5.83
Large Configuration Space	JOB	7.00	7.00	**1.00**	7.00	7.00	2.00	7.00
	SYSBENCH	6.67	3.00	**1.00**	5.67	3.00	4.00	4.67
	Average	6.83	5.00	**1.00**	6.33	5.00	3.00	5.83
Overall		5.22	3.06	**1.72**	5.94	3.89	4.00	5.83

S4.2 Configuration Optimization

Average ranking of optimizers in terms of the best performance they found. While we have presented the average rankings of optimizers in Table 7 in the paper, we detail the rankings on each workload and configuration space as shown in Table S4. In addition, we present how we calculate the average ranking of optimizers. For each workload and configuration space, we run three tuning sessions for an optimizer and sort the three sessions in terms of the best performance they found within 200 iterations. Then, we rank the optimizers based on the best performance in their best session, and then rank them based on their second session, and lastly the worst session. Finally, we average the three ranks of an optimizer, which corresponds to a row in Table S4.

S4.3 Knowledge Transfer

We have demonstrated the average performance enhancement (i.e., PE), speedup, and absolute performance ranking (i.e., APR) in Table 8 and omit the performance plot in the paper due to space limitations. Figure S2 plots the absolute performance over iteration

of each baseline (i.e., the combination of transfer framework and base learner). On TPCC, both RGPE (Mixed-kernel BO) and RGPE (SMAC) find the approximately best performance in 200 iterations, while RGPE (SMAC) has a better speedup. On SYSBENCH, RGPE (SMAC) finds the best performance, though it takes a few more steps. On Twitter, RGPE (Mixed-kernel BO) finds the best performance, and at a fast speed. In general, we find that the combinations of RGPE and base learners have the best absolute performance as well as speedup.

S5 CONSTRUCTION FOR DATABASE CONFIGURATION TUNING BENCHMARK

In this section, we present the procedures to construct the database configuration tuning benchmark and discuss the rationale behind such construction in detail.

Data Collection. We collect observation data to train the surrogate model used in the tuning benchmark. Our ultimate goal is to ensure that optimizers perform similarly using the tuning benchmark as interacting with the real database with workload replay.
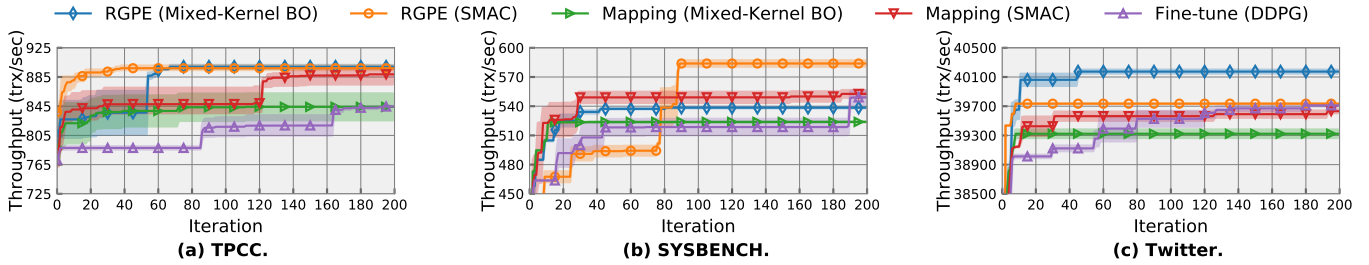
Figure S2: The absolute performance over iteration of each combination of transfer framework and base learner.

Table S5: Overview of regression models we evaluated.

Regression Models	Hyper-Parameter
Random Forest (RF)	*n_estimators, min_samples_split, min_samples_leaf, max_features, max_depth, bootstrap*
Gradient Boosting (GB)	*n_estimators, min_samples_split, min_samples_leaf, max_depth, learning_rate*
Support Vector Regression (SVR)	*gamma, C*
Nu Support Vector Regression (NuSVR)	*nu, gamma, C*
K-Nearest-Neighbours (KNN)	*n neighbors*
Ridge Regression (RR)	*alpha*

Table S6: Hyper-parameters we use for random forest based surrogate model.

Hyper-Parameter	SYSBENCH	JOB
n_estimators	1400	800
min_samples_split	2	2
min_samples_leaf	1	1
max_features	auto	auto
max_depth	100	100
bootstrap	True	True

Table S7: Hardware configurations for database instances.

Type	CPU	RAM
ecs.s6-c1m2.xlarge	4 cores	8GB
ecs.s6-c1m2.2xlarge	8 cores	16GB
ecs.s6-c1m2.4xlarge	16 cores	32GB
ecs.n4.8xlarge	32 cores	64GB

In principle, we could construct surrogates using observation data gathered by any means, but of course, we prefer to gather data in a way that leads to the best surrogates. Since effective optimizers spend most of their time in high-performance regions of the configuration space, and relative differences between the performance of configurations in such high-performance regions tend to impact which configuration will ultimately be suggested, thus accuracy in this part of the space is more important than in regions of poor performance. Training data should therefore densely sample high-performance regions. We thus collect observation data primarily via runs of existing optimizers. It is also important to accurately identify poorly performing parts of the space to avoid the overly optimistic predictions of performance in poor parts of the space. We therefore also included performance data gathered by LHS as it can deal effectively with large configuration spaces. To this end, to collect data for training the surrogate, we used the data gathered by ruining optimizers (e.g., Vanilla BO, DDPG, etc.) tuning the database under the specific workload, and as well as conducting LHS.

Model Selection and Hyper-parameter Tuning. We considered a broad range of commonly used regression algorithms as candidates for our surrogate benchmarks. Table S5 details the regression models and their hyper-parameters. We considered two different tree-based models, Random Forest (RF) and Gradient (GB), which have been shown to perform well for non-smooth and high-dimensional problems [34]. We also experimented with k-nearest-neighbors (KNN), ridge regression (RR), and two SVM methods. We implement all the methods using scikit-learn [66] (version 0.22.2) and conduct randomized search on the hyper-parameters. As shown in Table 9, RF and GB perform similarly. As RFs are widely used with simplicity, we adopt RF as the surrogate for tuning benchmark. And Table S6 presents the hyper-parameters we set for RF.

S6 EXPERIMENTAL ENVIRONMENT AND REPRODUCTION INSTRUCTIONS.

We conduct all experiments on Aliyun ECS. Each experiment consists of two instances. The first instance is used for the tuning sever, deployed on ecs.s6-c1m2.xlarge. The second instance is used for the target DBMS deployment, with four kinds of hardware configurations : ecs.s6-c1m2.xlarge, ecs.s6-c1m2.2xlarge, ecs.s6-c1m2.4xlarge, and ecs.n4.8xlarge. The detailed physical memory and CPU information are demonstrated in Table S7. The operation system of each ECS is Linux 4.9. The Python version used is 3.7, and the detailed package requirements for our experiments are listed on our GitHub repository. Please check the requirements.txt in the root directory.

To reproduce the results of out experiments, please download and install the workloads we use following the instructions in README.md, and run train.py under directory script/ with specified arguments like below:

```
python train.py --method=VBO --knobs_num=5 --y_variable=lat
--workload=job  --dbname=imdbload --lhs_log=JOB5_VBO.res
```

```
--knobs_config=../experiment/gen_knobs/job_shap.json
```

More reproduction details are provided on our GitHub repository https://github.com/PKU-DAIR/KnobsTuningEA.