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

[Supplemental Material]

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.

OUTLINE

This supplemental material is organized as follows.

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 algorithm). 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 (evaluation) 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 constraints.

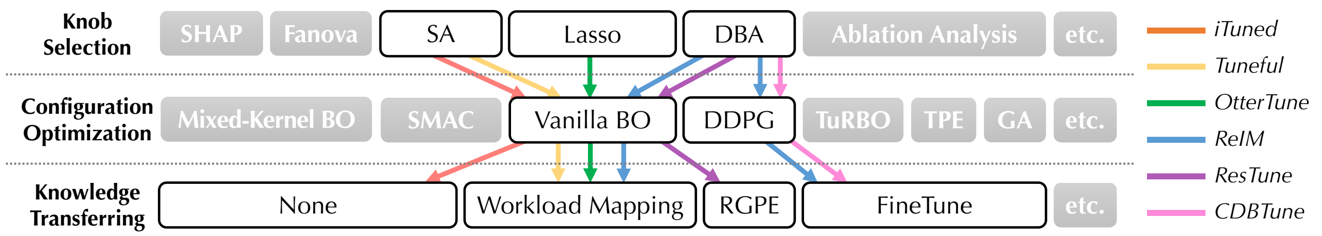


Figure S1: Detailed Intra-algorithm Designs of Configuration Tuning Systems (The Full Picture): Black boxes denote the algorithms adopted by existing database tuning systems (indicated by colored paths), and grey boxes denote the algorithms in the HPO field. SA denotes sensitivity analysis based on Gini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

S2.1 Ablation Analysis

Ablation analysis [14] 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 $\boldsymbol{\theta}_{default}$ and a target configuration $\boldsymbol{\theta}_{target}$ (usually a better one), ablation analysis first computes the feature differences $\Delta(\boldsymbol{\theta}_{default}, \boldsymbol{\theta}_{target})$ between the default and target configurations. Next, an ablation path $\boldsymbol{\theta}_{default}, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_{target}$ is iteratively constructed. In each iteration i with previous ablation path configuration $\boldsymbol{\theta}_{i-1}$, we consider all remaining feature changes $\delta \in \Delta(\boldsymbol{\theta}_{default}, \boldsymbol{\theta}_{target})$ and apply the change to the previous ablation path configuration and obtain the candidate $\boldsymbol{\theta}_{i-1}[\delta]$. Each parameter change δ is a modification of one feature from its value in $\boldsymbol{\theta}_{i-1}$ to its value in $\boldsymbol{\theta}_{target}$, along with any other feature modifications that may be necessary due to conditionally constraints in $\boldsymbol{\Theta}$. The next configuration on the ablation path $\boldsymbol{\theta}_i$ is the candidate $\boldsymbol{\theta}_{i-1}[\delta]$ 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 the 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 [56] (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 [56]. The exact computation of SHAP values is challenging, which can be estimated by Shapley sampling values method [78] or Kernel SHAP method [56].

S2.3 SMAC

SMAC [36] 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 $\hat{\mu}(\boldsymbol{\theta})$ and variance $\hat{\sigma}^2(\boldsymbol{\theta})$ for a new configuration $\boldsymbol{\theta}$ as the empirical mean and variance of the Gaussian distribution . SMAC uses the random forest model to select a list of promising parameter configurations. To quantify how promising a configuration $\boldsymbol{\theta}$ is, it uses the model's predictive distribution for $\boldsymbol{\theta}$ to compute its expected positive improvement $EI(\boldsymbol{\theta})$ [39] over the best configuration seen so far. $EI(\boldsymbol{\theta})$ is large for configurations $\boldsymbol{\theta}$ with high predicted performance and for those with high predicted uncertainty; thereby, it offers an automatic trade-off between 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 [28] 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 $\boldsymbol{\theta}$ is given by:

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

where w_i is the weight of base surrogate M_i, and μ_i and σ_i^2 are the predictive mean and variance of the base surrogate M_i. The weight w_i reflects the similarity between the previous task and the current task. Therefore, M_{meta} utilizes the knowledge on previous tuning 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:

$$L(M_j, H_T) = \sum_{j=1}^{n_t} \sum_{k=1}^{n_t} \mathbb{1}\Big((M_i(\boldsymbol{\theta}_j) \leq M_i(\boldsymbol{\theta}_k)) \oplus (y_j \leq y_k)\Big), \tag{7}$$

where $\oplus$ is the exclusive-or operator, n_t denotes the number of tuning tasks and $M_i(\boldsymbol{\theta}_j)$ means the prediction of M_i on configuration $\boldsymbol{\theta}$. 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 [61].

S3 CONSTRUCTION FOR DATABASE CONFIGURATION TUNING BENCHMARK

This section presents the procedures to construct the database configuration tuning benchmark and discusses 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. In principle, we could construct surrogates using observation data gathered by any means, but of course, we prefer to collect 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,

Table S1: 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 S3: Scale factors of workload from OLTP-Bench

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

Table S2: 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

to collect data for training the surrogate, we used the data gathered by ruining optimizers (e.g., Vanilla BO, DDPG, etc.) tuning the database, and as well as conducting LHS sampling .

Model Selection and Hyper-parameter Tuning. We considered a broad range of commonly used regression algorithms as candidates for our surrogate benchmarks. Table S1 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 [36]. We also experimented with k-nearest-neighbors (KNN), ridge regression (RR), and two SVM methods. We implement all the methods using scikit-learn [69] (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 S2 presents the hyper-parameters we set for RF.

S4 MORE DETAILS AND RESULTS ABOUT EXPERIMENT

In this section, we present additional experimental details.

S4.1 More Details about Workloads

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

The Reasons for Workload Selection. When answering **Q1** and **Q2**, we analyze the tuning performances over OLTP and OLAP scenarios. We use an OLAP workload – JOB and an OLTP workload – SYSBENCH. The reason is that the two workloads are often adopted in evaluating database configuration tuning methods and involve the scenarios of an online transaction/analytical processing. For example, JOB is adopted by QTune [50] and SYSBENCH is adopted by CDBTune [94], QTune [50] and ResTune [95]. When conducting knowledge transfer experiments (**Q3**), we focus on the OLTP scenarios since there are fewer OLAP workloads suitable for constructing source workloads of tuning tasks, except JOB and TPC-H. We choose three OLTP workloads – SYSBENCH, TPC-C, Twitter as the target tuning workloads, which have been adopted in previous studies. For example, OtterTune [5], CDBTune [94], and ResTune [95]) has adopted TPC-C for evaluation and ResTune has also adopted Twitter. We use additional four OLTP workloads (i.e., SEATS, Smallbank, TATP, Voter, SIBENCH) as source workloads and configure them with various sizes, read-write ratios as shown in Table 4. SIBench is a microbenchmark designed to explore snapshot isolation in DBMSs [40]. Based on our observations, the tuning opportunity for SIBench is limited. We add SIBench to the source workloads with the purpose of increasing the diversity.

Implementation of The Workloads. For JOB, we use the same setup illustrated in [48]. 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 S3. 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.

S4.2 Knob Selection

Top impacting knobs with high tunability for OLTP workloads. We further conduct an experiment using SHAP to generate a ranking of the most impacting knobs across OLTP workloads and hardware instances. 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 **??**. Then we adopt SHAP to generate an importance ranking respectively and count the number of times that each knob appears in the top 20 of all the rankings to measure their overall importance. Table S4 shows the top-20 important knobs for OLTP workloads and their brief description. We believe this ranking could provide database practitioners a 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 gain the most, which is aligned with

Table S4: The Top-20 important knobs selected by SHAP 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.
innodb_flush_log_at_trx_commit	Categorical	Yes	Logging	Controlling the balance between ACID compliance for commit operations and performance.
innodb_max_dirty_pages_pct_lwm	Integer	Yes	Logging	The percentage of dirty pages at which preflushing is enabled to control the dirty page ratio.
innodb_log_files_in_group	Integer	No	Logging	The number of log files in the log group.
innodb_buffer_pool_size	Integer	Yes	Memory	The size in bytes of the buffer pool.
innodb_online_alter_log_max_size	Integer	Yes	Logging	An upper limit on the size of the log files used during online DDL operations for InnoDB tables.
key_cache_age_threshold	Integer	Yes	Memory	The demotion of buffers from the hot sublist of a key cache to the warm sublist.
binlog_cache_size	Integer	Yes	Memory	The size of the cache to hold changes to the binary log during a transaction.
innodb_purge_rseg_truncate_frequency	Integer	Yes	Logging	The frequency with which the purge system frees rollback segments.
query_cache_limit	Integer	Yes	Memory	The minimum size of cached results.
innodb_sort_buffer_size	Integer	No	Memory	The sort buffer size for online DDL operations that create or rebuild secondary indexes.

Table S5: Hardware configurations for more instances.

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

the previous analysis [5, 6, 95]. And 11 of the top-20 knobs are related to memory allocation, which indicates that the default setting of memory allocation in MySQL may not be appropriate across the workloads and hardware instances. 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.

S4.3 Configuration Optimization

Average ranking of optimizers in terms of the best configuration 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 S6. 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 S6.

S4.4 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 constraints. 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 EVALUATIONS ON POSTGRESQL

Different databases have completely different configuration knobs, including different meanings, types, names and value ranges. In this selection, we conduct experiments for JOB workload on a different database system, PostgreSQL (v12.7). We first analyze the important knobs in PostgreSQL and compare the optimizers over small (top-5 important knobs) and medium (top-20 important knob) configuration spaces. Finally, we present the evaluation results via database tuning benchmark.

S5.1 Knob Selection on PostgreSQL

Similar to the procedure described in Section 5, we first collect 3,000 samples for JOB workload on PostgreSQL via Latin Hypercube Sampling and then use SHAP as the importance measurement to select the most important 5 and 20 knobs. Before the collection, we find that Postgres is more sensitive to knob settings, i.e., improper knobs configuration will lead to database crashes and termination of query execution. Therefore, we lower the upper limit of the memory-related knobs based on the hardware settings.

The top-5 important knobs and their brief description are demonstrated in Table S7, among which two are related to memory, two are related to logging, and one is related to concurrency. We observe that the knob with the largest tunability, *max_worker_processes* sets the maximum number of background processes that the system can support, which shares a similar function with MySQL's top-1 important knob as shown in Table S4 For *max_worker_processes* and *shared_buffers*, configuring larger values than the default can contribute to the decreasing of latency. And this conclusion is on par with the ones on MySQL. *wal_buffers* set the amount of shared memory used for WAL data that has not yet been written to disk, with a default setting of -1 that selects a size equal to about 3% of *shared_buffers.*, while based on our observation, a higher proportion at about 7% can achieve the best performance. To avoid flooding the

Table S6: Average ranking of optimizers in terms of the best configuration they found.Bold values are the best.

Optimizer		Vanilla BO	Oon-Hot BO	Mixed-Kernel BO	SMAC	TPE	TURBO	DDPG	GA
Small Configuration Space	JOB	5.00	2.67	**2.33**	3.33	6.33	3.67	5.00	7.67
	SYSBENCH	5.67	5.33	**2.00**	3.33	5.33	4.00	5.00	5.33
	Average	5.33	4.00	**2.17**	3.33	5.83	3.83	5.00	6.50
Medium Configuration Space	JOB	5.33	3.33	3.00	**1.00**	7.67	3.67	5.00	7.00
	SYSBENCH	5.00	4.33	**1.67**	1.67	6.67	4.33	6.00	6.33
	Average	5.17	3.83	2.33	**1.33**	7.17	4.00	5.50	6.67
Large Configuration Space	JOB	7.00	7.00	7.00	**1.00**	7.00	7.00	2.00	7.00
	SYSBENCH	7.67	6.00	3.33	**1.00**	6.00	3.00	4.33	4.67
	Average	7.33	6.50	5.17	**1.00**	6.50	5.00	3.17	5.83
Overall		5.94	4.78	3.22	**1.89**	6.50	4.28	4.56	6.33

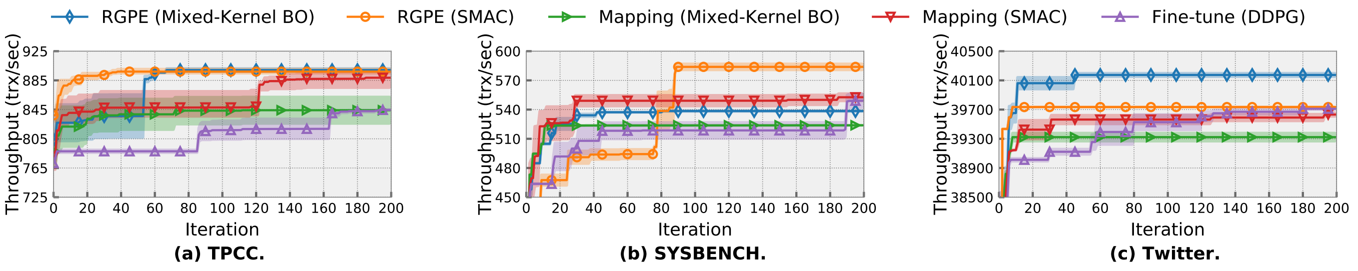


Figure S2: The absolute performance over iteration of each combination of transfer framework and base learner.

Table S7: The Top-5 important knobs selected by SHAP for JOB on PostgreSQL

Knob	Type	Module	Description
max_worker_processes	Integer	Concurrency	Maximum number of background processes that the system can support.
shared_buffers	Integer	Memory	The number of shared memory buffers used by the server.
wal_buffers	Integer	Memory	The number of disk-page buffers in shared memory for WAL.
checkpoint_completion_target	Real	Logging	Time spent flushing dirty buffers during checkpoint, as fraction of checkpoint interval.
backend_flush_after	Integer	Logging	Number of pages after which previously performed writes are flushed to disk.

I/O system with a burst of page writes, *checkpoint_completion_target* controls the time of writing dirty buffers during a checkpoint. With the default value of 0.5 for *checkpoint_completion_target*, PostgreSQL is expected to complete each checkpoint in about half the checkpoint interval before the next checkpoint starts. In our experiments, a higher value of about 0.8 can improve the performance by reducing the I/O load from checkpoints. For *bakcend_flush_after*, the default is 0, i.e., no forced writeback, while a higher value can limit the amount of dirty data in the kernel's page cache, reducing the likelihood of stalls when a fsync is issued at the end of a checkpoint, or when the OS writes data back in larger batches in the background.

S5.2 Configuration Optimization on PostgreSQL

We further evaluate the optimizers' performance over small and medium spaces for workload JOB on PostgreSQL. Figure S3 presents the results, where the conclusion is similar as on MySQL. SMAC achieves the best performance in both cases. For the small space, mixed-kernel BO, one-hot BO, and vanilla BO demonstrate similar convergences since there are no categorical knobs. For medium space with categorical ones, mixed-kernel BO outperforms one-hot BO and vanilla BO. Overall, the BO-based methods outperform RL-based DDPG, while TPE performs poorly on medium space, where there exist knobs that interact with each other (e.g., *checkpoint_completion_target* and *checkpoint_timeout*).

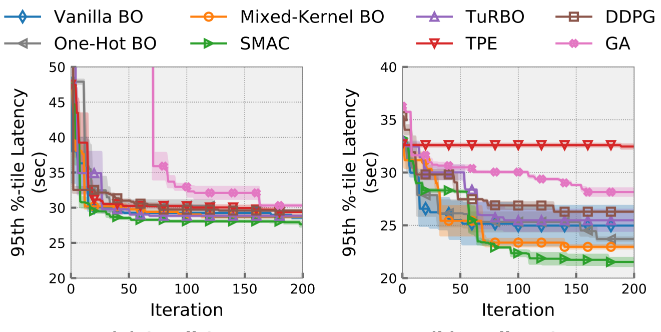
S5.3 Evaluation via Database Tuning Benchmark on PostgreSQL

To support efficient benchmarking over the configuration space of PostgreSQL, we fit two surrogate models based on RFs over the small and medium configuration space of JOB workload in PostgreSQL. And the surrogate model is available online in our repository [1]. Figure S4 depicts the best performance found by different optimizers using the tuning benchmark. We report means and quartiles across three runs of each optimizer. We observe that our tuning benchmark yields evaluation results closer to the result in Figure S3. The results on PostgreSQL demonstrate that researchers can conduct algorithm analysis and comparison efficiently under various setups with the help of the tuning benchmark.

S6 EXPERIMENTAL ENVIRONMENT AND REPRODUCTION INSTRUCTIONS.

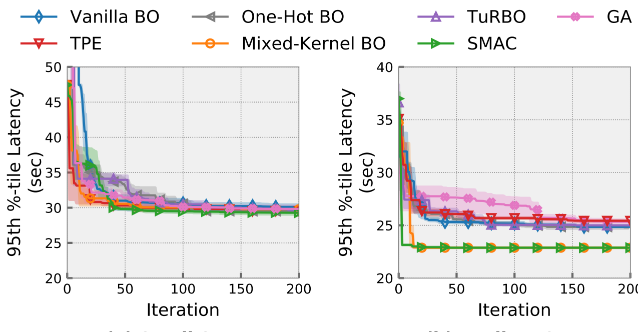
We conduct all experiments on Aliyun ECS. Each experiment consists of two instances. The first instance is used for the tuning sever, deployed on `ecs.s6-c1m2.xlarge`. The second instance is used for the target DBMS deployment, with four kinds of hardware

[1]https://github.com/PKU-DAIR/KnobsTuningEA



(a) Small Space **(b) Medium Space**

Figure S3: Tuning Performance on PostgreSQL.



(a) Small Space **(b) Medium Space**

Figure S4: Tuning Performance over surrogate benchmark on PostgreSQL.

Table S8: 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

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 S8. 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 online repository[1].

REFERENCES

[1] 2022. fANOVA. https://www.automl.org/ixautoml/fanova/.
[2] 2022. How large should be mysql innodb buffer pool size? https://dba.stackexchange.com/a/91354.
[3] 2022. InnoDB Startup Options and System Variables. https://dev.mysql.com/doc/refman/8.0/en/innodb-parameters.html.
[4] 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.
[5] 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.
[6] 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.
[7] 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*.
[8] 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.
[9] Syrine Belakaria, Aryan Deshwal, and Janardhan Rao Doppa. 2019. Max-value Entropy Search for Multi-Objective Bayesian Optimization. In *NeurIPS*. 7823–7833.
[10] R. E. Bellman. 1957. A Markov decision process. *Journal of Mathematical Fluid Mechanics* 6 (1957).
[11] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. In *NIPS*. 2546–2554.
[12] 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).
[13] 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.
[14] 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.
[15] Leo Breiman. 2001. Random Forests. *Mach. Learn.* 45, 1 (2001), 5–32.
[16] Stefano Cereda, Stefano Valladares, Paolo Cremonesi, and Stefano Doni. 2021. CGPTuner: a Contextual Gaussian Process Bandit Approach for the Automatic Tuning of IT Configurations Under Varying Workload Conditions. *Proc. VLDB Endow.* 14, 8 (2021), 1401–1413.
[17] Surajit Chaudhuri and Vivek R. Narasayya. 2007. Self-Tuning Database Systems: A Decade of Progress. In *VLDB*. ACM, 3–14.
[18] Surajit Chaudhuri and Gerhard Weikum. 2006. Foundations of Automated Database Tuning. In *VLDB*. ACM, 1265.
[19] 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.
[20] Songyun Duan, Vamsidhar Thummala, and Shivnath Babu. 2009. Tuning Database Configuration Parameters with iTuned. *Proc. VLDB Endow.* 2, 1 (2009), 1246–1257.
[21] Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. 2004. Least angle regression. *The Annals of statistics* 32, 2 (2004), 407–499.
[22] Katharina Eggensperger. 2013. Towards an Empirical Foundation for Assessing Bayesian Optimization of Hyperparameters.
[23] 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.
[24] David Eriksson, Michael Pearce, Jacob R. Gardner, Ryan Turner, and Matthias Poloczek. 2019. Scalable Global Optimization via Local Bayesian Optimization. In *NeurIPS*. 5497–5508.
[25] Chris Fawcett and Holger H. Hoos. 2016. Analysing differences between algorithm configurations through ablation. *J. Heuristics* 22, 4 (2016), 431–458.
[26] 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.
[27] 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.
[28] 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.
[29] 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.
[30] Chen Gao, Quanming Yao, Depeng Jin, and Yong Li. 2021. Efficient Data-specific Model Search for Collaborative Filtering. In *KDD*. ACM, 415–425.

[31] Marius Geitle and Roland Olsson. 2019. A New Baseline for Automated Hyper-Parameter Optimization. In *LOD (Lecture Notes in Computer Science)*, Vol. 11943. Springer, 521–530.

[32] 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.

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

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

[35] 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.

[36] 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.

[37] 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.

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

[39] 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.

[40] 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.

[41] 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.

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

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

[44] 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.

[45] Olga Krakovska, Gregory Christie, Andrew Sixsmith, Martin Ester, and Sylvain Moreno. 2019. Performance comparison of linear and non-linear feature selection methods for the analysis of large survey datasets. *Plos one* 14, 3 (2019), e0213584.

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

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

[48] 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.

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

[50] 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.

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] 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)*.

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

[57] 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.

[58] 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.

[59] 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.

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

[61] 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.

[62] 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.

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

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

[65] 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).

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

[67] 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.

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

[69] 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.

[70] 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.

[71] 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.

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

[73] 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.

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

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

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

[77] 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.

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

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

[80] 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.

[81] 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.

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

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

[84] 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.

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

[86] 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.

[87] 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.

[88] 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.

[89] 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.

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

[91] 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.

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

[93] 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.

[94] 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.

[95] 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.