DOT: Dynamic Knob Selection and Online Sampling for Automated Database Tuning

Yifan Wang
Orange / Inria / Univ. Lille, France
yifan.wang.etu@univ-lille.fr

Debabrota Basu
Inria / Univ. Lille / CNRS, France
debabrota.basu@inria.fr

Pierre Bourhis
CNRS / Univ. Lille / Inria, France
pierre.bourhis@univ-lille.fr

Romain Rouvoy
Univ. Lille / CNRS / Inria, France
romain.rouvoy@univ-lille.fr

Patrick Royer
Orange, France
patrick.royer@orange.com

ABSTRACT

Database Management Systems (DBMS) are crucial for efficient data management and access control, but their administration remains challenging for *Database Administrator*s (DBAs). Tuning, in particular, is known to be difficult. Modern systems have many tuning parameters, but only a subset significantly impacts performance. Focusing on these influential parameters reduces the search space and optimizes performance. Current methods rely on costly warm-up phases and human expertise to identify important tuning parameters. In this paper, we present DOT, *a dynamic knob selection and online sampling DBMS tuning algorithm.* DOT uses *Recursive Feature Elimination with Cross-Validation* (RFECV) to prune low-importance tuning parameters and a *Likelihood Ratio Test* (LRT) strategy to balance exploration and exploitation. For parameter search, DOT uses a *Bayesian Optimization* (BO) algorithm to optimize configurations on-the-fly, eliminating the need for warm-up phases or prior knowledge (although existing knowledge can be incorporated). Experiments show that DOT achieves matching or outperforming performance compared to state-of-the-art tuners while substantially reducing tuning overhead.

PVLDB Reference Format:
Yifan Wang, Debabrota Basu, Pierre Bourhis, Romain Rouvoy, and Patrick Royer. DOT: Dynamic Knob Selection and Online Sampling for Automated Database Tuning. PVLDB, 19(4): 589 - 602, 2025.
doi:10.14778/3785297.3785302

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Orange-OpenSource/dot.

1 INTRODUCTION

In recent years, the exponential growth in data volumes [36] has significantly increased the demands on DBMS resources and intensified the challenges faced by DBAs. As complex, specialized software designed to manage data efficiently, DBMS requires substantial computational resources (e.g., CPU, RAM) and expert administration. Among these administrative tasks, database tuning is critical, as configuration adjustments directly influence system performance [34]. Optimal performance hinges on precise knob tuning, where misconfigurations can lead to significant performance degradation. Therefore, ensuring that each knob is appropriately tuned is not only essential for maintaining system efficiency but also for achieving reliable and scalable operations.

Modern DBMS, such as MySQL [1], expose hundreds of configurable knobs, resulting in a vast tuning space that is both difficult for humans to comprehend and impractical to explore exhaustively. Importantly, the influence of these knobs on performance varies—some can have a significant impact, while others contribute minimally—and their importance is often contingent on the specific workload and hardware configuration. This variability makes knob selection equally critical: identifying and focusing on the most influential knobs can dramatically reduce the search space, improve convergence rates, and lower computational overhead for tuning algorithms. In the work of [49], it is demonstrated that the large number of knobs to tune, once they surpass a given threshold, will no longer yield better performance but only to increase the computational overhead. Consequently, most state-of-the-art DBMS tuning algorithms limit the tuning scope by selecting only the 10 to 20 most influential knobs [7, 44, 48]. Emphasizing both knob tuning and informed knob selection is therefore fundamental to achieving efficient and effective DBMS performance optimization.

Many previous studies have tackled knob selection and DBMS tuning using a variety of approaches that fall into four broad categories: experience-based, screening-based, ML-based, and *Large Language Model* (LLM)-based methods. Experience-based systems, like CDBTune [48], use expert-driven knob selection to shrink the search space to apply *Deep Deterministic Policy Gradients* (DDPG) [23], UDO [44] extends this with index tuning via *Monte Carlo Tree Search*, but still leaves knob choice to the user. Screening-based approaches, such as Plackett & Burman [9] and sensitivity analysis [37], collect knobs-performance samples and apply statistical methods to rank the knob importance. ML-based approaches exemplified by OtterTune [41], which applies Lasso [38] to prune knobs before BO, and Hunter [7], which uses Random Forest with *Classification And Regression Trees* (CART) [6]–automatically ranking and eliminating low-importance knobs. Recent LLM-based methods, such as DB-BERT [40] and GPTuner [22], leverage models, like BERT [10] and ChatGPT [26], to parse documentation for knob importance, then hand off tuning to algorithms—like DQN [16] or SMAC [32].

Proceedings of the VLDB Endowment, Vol. 19, No. 4 ISSN 2150-8097.
doi:10.14778/3785297.3785302

Although these techniques effectively reduce dimensionality, they often incur complex data collection processes and significant computational overhead. ML-based approaches require extensive pre-collected database performance samples (configuration-to-performance mappings) for model training and knob ranking, making them highly resource-intensive. For example, OLTP workload sample collection involves stress-testing the DBMS under various configurations and computing the average *Transactions Processed per Second* (TPS), whereas OLAP workloads typically require sequential execution of analytical queries, measuring their total run-time under each configuration. In practice, higher sample counts yield better precision in importance estimates and increase statistical confidence in the rankings. Although Hunter [7] reported that collecting only two samples per knob was sufficient for knob ranking, conventional heuristics recommend gathering ten samples per knob to ensure model validity [29]. In a DBMS tuning study [49], more than fifteen samples per knob were used to construct a reliable ranking. Consequently, accurately ordering a large set of knobs can demand hundreds of measurements, prolonging the sample-collection and knob selection phase. Screening methods, like Plackett & Burman [9] and sensitivity analysis [37], are much more lightweight—typically requiring on the order of twice the number of knobs in experimental runs—yet their importance scores remain fairly coarse, since they do not capture interactions or the full performance-configuration surface.

Although experience-based methods leverage human judgment, they are not without their own challenges. According to [45], even most experienced DB experts struggle with DBMS tuning, therefore significant time and effort is required to accurately identify and rank the most critical knobs. Lastly, LLM-based methods are also associated with significant costs and complexities. Document collection and organization can be cumbersome, particularly given variations across DBMS versions and types. Furthermore, even after documentation compilation, interactions with language models remain time-consuming. GPTuner [22], for example, reported more than five hours spent solely on LLM-based knob selection, accompanied by financial overhead exceeding $50 in API consumption per tuning. In summary, current knob selection strategies impose substantial overheads that hinder their widespread adoption. Together, these requirements in time, compute, and specialized expertise place current methods out of reach for users with limited resources or domain familiarity, highlighting the need for a more efficient and accessible knob-selection solution.

To bypass complex configurations and extensive data requirements, we introduce a lightweight solution with minimal overhead: DOT (*Dynamic knob selection and Online sampling for automated database Tuning*). The core concept of DOT is straightforward: during DBMS optimization, we continuously collect performance samples and fully exploit them to rank knob importance, prune low-impact knobs, and introduce new knobs dynamically, rather than pre-ranking knobs before tuning. By dynamically controlling the search-space dimension through online sampling, DOT focuses only on the most impactful knobs. Specifically, it applies RFECV to prune low-importance knobs based on samples collected during tuning, while an LRT strategy determines whether to explore by adding knobs or to exploit by refining the current set. To further reduce benchmarking overhead, DOT incorporates an adaptive benchmarking mechanism that accelerates tuning and lowers resource consumption. For efficient parameter search, DOT employs BO to optimize DBMS configurations on-the-fly, eliminating costly warm-up phases and prior knob-importance knowledge. Additionally, when prior knob-importance information is available, DOT can seamlessly integrate it to achieve even faster tuning.

Contributions In this paper, we share the following contributions: **(1)** We introduce an experimentation protocol that improves the reproducibility of DBMS tuning algorithms in noisy, stochastic tuning settings; **(2)** We conduct an exhaustive empirical study of knob selection in DBMS tuning, revealing the inherent complexity of knob selection and its dependence on workload specificities that motivates DOT; **(3)** Unlike recent DBMS tuners that rely on extensive pretraining [7, 41, 48] or heavy LLM-based documentation analysis [22, 40], we present DOT, a lightweight tuning framework that requires no dedicated warm-up yet can seamlessly incorporate prior knob-importance information when available; and **(4)** Experimental results demonstrate that DOT matches or outperforms state-of-the-art tuners while substantially reducing tuning overhead, with or without prior knob-importance knowledge.

The paper is organized as follows: Section 2 analyzes reproducibility challenges and details our robust evaluation protocol. Then, Section 3 presents the exhaustive knob-selection experiments that motivate DOT. Finally, Section 4 describes the design and operation of DOT and Section 5 evaluates its performance and efficiency against baselines and leading algorithms.

2 REPRODUCIBILITY & EVALUATION CHALLENGES IN DBMS AUTO-TUNING

In this section, we highlight reproducibility challenges in DBMS tuning, demonstrate the high variance in both algorithms and performance measurements, and introduce a robust evaluation protocol for our following experiments.

State-of-the-art DBMS auto-tuning methods promise significant performance improvements, yet their reproducibility remains challenging due to inherent complexities and environmental variability. Recent state-of-the-art approaches rely heavily on extensive pretraining and substantial preliminary datasets, which drastically reduce tuning times but introduce reproducibility barriers:

- **Heavy data requirements:** Pre-training typically requires large, high-quality datasets that are rarely shared publicly and costly to collect [41, 48].
- **System Diversity:** Variations in hardware, system configurations, and data collection methods frequently impede accurate reproduction of published results.
- **Complex setup:** Configuring and warm-starting advanced tuning algorithms, such as *Deep Reinforcement Learning* (DRL) like DDPG, demand specialized expertise and hyper-parameter tuning of the algorithm itself [11].

Specific systems exhibit different degrees of these challenges. For instance, CDBTune requires substantial DBA expertise alongside thousands of high-quality samples; Hunter mandates dedicated infrastructure to generate extensive initial datasets; OtterTune relies on large-scale metric aggregation pipelines; *UDO* demands manual selection of knobs by users; and recent LLM-based methods (e.g., *DB-BERT*, GPTuner) hinge on extensive documentation parsing,

dependent on the capabilities of the underlying LLM. Consequently, setup complexity and data collection overhead inflate overall tuning duration, impair reproducibility, and limit applicability.

In contrast, DOT adopts a lightweight strategy, dispensing entirely with pre-training, thus minimizing preliminary data collection and expert intervention. This significantly enhances reproducibility and broadens potential industrial adoption.

Beyond reproducibility barriers inherent to current tuning strategies, another critical challenge is the variability inherent to noisy DBMS tuning environments. Performance measurements of DBMS benchmarks display substantial noise due to factors such as transient network congestion [5, 8], OS/VM background activities [19], intrinsic benchmark randomness [3, 20], and hardware side-effects like CPU throttling [2, 27]. We evaluated performance variability across three representative workloads (two OLTP workloads: TPC-C and Sysbench, and one OLAP workload: TPC-H). Table 1 quantifies run-to-run variation for 10 runs on the same MySQL database under default configuration.

Table 1: Variability of performance measurements

Benchmark	Type	Mean	SD	CV (%)
TPC-H	OLAP	80.13 sec.	0.42	0.52
TPC-C	OLTP	79.33 TPS	4.95	6.24
Sysbench	OLTP	608.63 TPS	26.53	4.36

For OLAP workloads, we use the total execution time as a measurement of their performance, while for OLTP workloads, we use the TPS as a performance metric. It is observed that even under an *identical* configuration, the coefficient of variation (CV= $\frac{\text{SD}\times 100}{\text{Mean}}$) reaches 6.2% for TPC-C and 4.4% for Sysbench. TPC-H presents a relatively lower variance of 0.52%. Additionally, ML-based tuning algorithms, such as *Bayesian Optimization* (BO), exhibit inherent stochasticity [14], leading to considerable variability in tuning speed and convergence behavior. To systematically assess convergence, we define a *near-optimality* criterion based on the workload type.

For OLTP workloads, we declare convergence once throughput stabilizes within noise. Formally, let t^*_{OLTP} be the first iteration (after 100 steps) where the best throughput over the past 100 steps improves by less than 5%. The optimal throughput is $P^* = \max_{t<t^*_{\text{OLTP}}} \text{max\_tps}(t)$, and convergence occurs at $Y_{\text{OLTP}} = \min\{t \mid \text{max\_tps}(t) \geq 0.95\, P^*\}$, i.e., when throughput reaches 95% of P^* (matching the ∼5% noise level).

For OLAP workloads, we apply the same idea to query-batch execution time. Here t^*_{OLAP} is the first iteration (after 100 steps) where the best time improves by less than 1%, reflecting the lower noise of OLAP benchmarks. The optimal execution time is $E^* = \min_{t<t^*_{\text{OLAP}}} \text{total\_exec\_time}(t)$, and convergence is reached at $Y_{\text{OLAP}} = \min\{t \mid \text{total\_exec\_time}(t) \leq 1.01\, E^*\}$.

Table 2 highlights the inherent stochasticity of ML-based optimizers, illustrating significant variability in iterations and time to reach convergence. The number of iterations required to reach near-optimality varies sharply: TPC-C shows the widest absolute spread (standard deviation of about 122 iterations), whereas TPC-H is the noisiest in relative terms (coefficient of variation of about 73%), confirming that convergence speed can differ markedly from run to run. The time needed to achieve near optimality exhibits similar volatility. BO needs 3.4 h for Sysbench, 6.9 h for TPC-C, and 1.2 h for TPC-H, on average, but the corresponding standard deviations (1.2 h, 4.4 h, and 0.9 h) translate into coefficients of variation of 35%, 64%, and 73%, respectively.

Table 2: Convergence variability of the BO tuner: iterations & time to reach near-optimal performance across benchmarks

Benchmark	Iterations			Time (hours)		
	Mean	SD	CV (%)	Mean	SD	CV (%)
TPC-H	57.0	41.3	72.5	1.21	0.9	72.9
TPC-C	215.6	122.2	56.7	6.9	4.4	64.3
Sysbench	118.4	40.3	34.1	3.4	1.2	34.6

Given the substantial noise introduced by both the execution environment and the tuning algorithms, we adopt a more rigorous evaluation protocol to mitigate bias:

(1) **Replication:** We execute each tuning process *five* times with distinct random seeds, performing a full DBMS reboot between runs to minimize warm-up and caching artifacts.
(2) **Aggregate reporting:** results are summarized by the sample mean and standard deviation across all trials, with 95% confidence intervals shown in our figures to convey uncertainty.
(3) **Statistical testing:** We assess the significance of performance differences using Welch's two-tailed t-test [46] for pairwise comparisons and the non-parametric Friedman test [13] for multiple comparisons. Both tests rely on their corresponding p-values to quantify the likelihood of observing the reported differences under the null hypothesis. We adopt a significance threshold of $p < 0.05$.

While this protocol reduces bias, it cannot eliminate it entirely. In principle, a larger number of repetitions would yield more definitive comparisons, but each full tuning run requires two machines for over 10 hours, making extensive replication prohibitively expensive; hence, we settle on five repetitions in this study.

3 EXPERIMENTAL STUDY OF KNOB SELECTION & CHALLENGES

In this section, we analyse different knob selection strategies and present experimental evidence that underscores the inherent challenges of knob selection in DBMS tuning. These difficulties motivate the need for a dynamic, efficient, and cost-aware approach—precisely what our proposed method, DOT, is designed to deliver.

Two OLTP workloads (Sysbench with 10 tables of 2 million rows and 50 concurrent threads, implementation from [20]; TPC-C at scale 100 with 32 threads, implementation from [30]) and one OLAP workload (TPC-H at scale 1, implementation from [12]) are used for the following evaluations. Experiments were conducted on MySQL 8 running on a VM (4 vCPU, 16 GB RAM) in Orange's FlexEngine [33] cloud platform.

3.1 Analysis of Knob Selection Strategies

First, we evaluate the performance of different knob-selection strategies by comparing six methods on a fixed set of 22 expert-identified

Table 3: Performance tuned with TOP-5 knobs selected by different methods

Workload	Sensitivity	P & B	Lasso	CART	Experience	LLM
Sysbench (TPS)	1411.1	939.1	1469.04	1588.7	1294.36	1411.1
TPC-H (Exec. s)	46.22	74.35	46.02	45.76	45.72	46.35

knobs using two representative benchmarks: Sysbench (OLTP) and TPC-H (OLAP).

Each method—P&B [9], Lasso [41], CART [7], Sensitivity Analysis [37], an experience-based approach, and LLM knowledge based on GPTuner's prompt[22] to ask ChatGPT 4o to produce a ranking of the 22 knobs. We collected 48 configuration–performance samples for P&B, 220 samples each for CART and Lasso, and 44 samples for Sensitivity Analysis, each following their respective ranking protocols. In contrast, ChatGPT 4o required no sampling, significantly reducing the overhead of knob ranking and the experience-based approach requires input of a senior DBA.

Subsequently, we conducted an exhaustive grid search over the top five knobs identified by each method, evaluating integer knobs at five discrete values and boolean knobs at two states, generating up to $5^5 = 3,125$ configurations per knob set (approximately 78 hours of benchmarking each). Due to the computational burden, we sampled the performance–configuration space only once (despite benchmarking noise) and temporarily suspended the five-repetition protocol, accepting the resulting increase in variance as an unavoidable trade-off to make this small-scale comparison.

Grid search ensures unbiased evaluation by eliminating tuning algorithm variability, allowing a pure assessment of knob selection methods. Although real-world configurations often include many more knobs, exhaustively searching them would incur exponential computational costs; therefore, we limit our study to five. Table 3 compares the performance under different knob sets:

(1) **CART** achieves the highest Sysbench throughput (1,588.7 TPS) and among the lowest TPC-H execution times (45.76 s), indicating superior overall knob-selection effectiveness. However, CART requires the most extensive data collection (220 samples).
(2) **Lasso** and **LLM knowledge** match CART at specific points in Sysbench tuning but underperform for OLAP. Lasso has a high sampling cost as CART, whereas ChatGPT achieves comparable rankings with low overhead (a single prompt).
(3) **Experience-based** and **Sensitivity Analysis** methods yield moderate improvements over default settings but do not reach CART's best performance.
(4) **P& B** demonstrates modest gains with the least sampling requirements, reflecting a lower resource–performance trade-off.

These preliminary findings—derived from single exhaustive grid searches per configuration—demonstrate CART's strong knob selection performance alongside its substantial sampling overhead, clearly illustrating the trade-off between sampling cost and performance gains and underscoring that sub-optimal knob choices directly impair tuning results. As initial guidance, they highlight the challenge of balancing sampling effort against tuning efficacy.

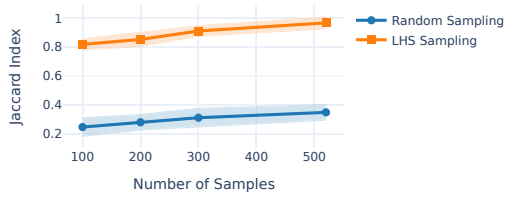

Figure 1: Variability of CART knob selection across varying sample sizes & sampling strategies

3.2 Cost of Robust Knob Rankings

To further investigate the cost of robust knob ranking, we adopt a more practical setting to knob selection and DBMS tuning by moving beyond the narrow 22-knob subset. Recognizing that many MySQL variables govern security, file paths, or other non-performance concerns—and constrained by our compute budget—we filtered the full set of tunable parameters down to the 52 most performance-relevant knobs (primarily InnoDB settings such as "innodb_purge_threads", "innodb_read_io_threads", "innodb_buffer_pool_size"). More details of the experimental setup and the complete knob lists and configuration scripts are available in the Availability statement.

This targeted reduction both removes irrelevant options and keeps the global optimization search space realistic.

We delve deeper into the computational expense and variability of the CART method from Hunter, which constructs a Random Forest and aggregates knob importance via majority voting among CART trees [7]. Given that ML algorithms such as CART are inherently stochastic, their knob rankings may vary with different random seeds, making reproducibility an important concern.

To quantify this variability, we use CART to rank the pre-selected 52 knobs, choosing the Top 20 as in Hunter [7]. We train CART models with varying sample sizes and random seeds, then compute the Jaccard index between each pair of Top 20 sets to measure how consistently the same knobs are selected. The Jaccard index is defined as $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$, where $J(A,B) \in [0,1]$, with higher values indicating greater similarity between two sets.

Different sampling strategies may also influence the results. We compare *Latin Hypercube Sampling* (LHS)—used by OtterTune [41] and Hunter [7]—against pure random sampling (as in CDBTune [48]). Figure 1 plots Jaccard indices for the Sysbench workload on MySQL when using 100, 200, 300, and 520 samples (520 aligns with ten samples per feature [29]). Under random sampling, Jaccard indices remain low: different seeds yield very different Top 20 sets, indicating high stochasticity. In contrast, LHS produces higher Jaccard indices and more consistent knob selections. In both cases, increasing the number of samples improves stability. However, collecting samples is costly: each sample requires one client machine running Sysbench against a DBMS server for about 100 seconds, so 520 samples take roughly 14.4 hours on two machines.

Additionally, we employ BO (implemented using Scikit Optimize [17]) to tune the DBMS with the Top 20 knobs identified through CART rankings derived from different numbers of samples generated by LHS. Although BO inherently introduces variability, grid search becomes infeasible due to the relatively high dimensionality (20 knobs), making BO a more practical choice. Following the

protocol detailed in Section 2, we record the final throughput and total tuning time to evaluate how effectively each ranking identifies the most impactful knobs.

Intuitively, providing with more samples enhances knob-ranking quality by supplying richer information. Table 4 presents the tuned DBMS performance, Friedman test rankings, and sampling times using knob sets selected via CART at varying sample sizes via BO. Generally, as the sample size increases, tuned DBMS performance does improve. However, performance gains beyond 200 samples become marginal and statistically insignificant according to the t-test. Specifically, increasing from 100 to 200 samples notably enhances performance (from 1,602.3 TPS to 1,690.8 TPS), but further increments (e.g., from 200 to 300 or 520 samples) offer limited benefits.

Regarding tuning time, results indicate no clear correlation with sample size, as tuning duration fluctuates without a distinct pattern. Importantly, sampling time significantly surpasses tuning time in all cases, dominating the total effort. Sampling 100 configurations already requires approximately 167 minutes, rising sharply to about 333 minutes (over 5 hours) for 200 samples, and further ballooning to approximately 867 minutes (over 14 hours) for 520 samples.

These findings underscore a crucial trade-off: increasing sample sizes provides diminishing returns in performance improvement but incurs growing sampling costs. Our experiments suggest that selecting around 200 samples achieves an optimal balance between improved performance and manageable sampling effort, although this optimal point could vary with different workloads and scenarios. Thus, careful consideration of this trade-off is essential, as proper and robust knob selection methods are inherently costly.

Table 4: Sysbench tuning with CART on knob sets from varying sample sizes (statistically significant at p < 5E-2)

Samples	Rank (Max TPS) Test stat: 7.32 p-value: $6.2E-2$	Rank (tuning time [min]) Test stat: 1.32 p-value: $7.2E-1$	Sampling time (min)
520	1.8 (1,708.0 ± 29.1)	2.8 (216.2 ± 32.6)	866,7
300	2.0 (1,684.3 ± 48.9)	2.0 (160.8 ± 79.5)	500,0
200	2.4 (1,690.8 ± 65.2)	2.8 (214.9 ± 116.9)	333.3
100	3.8 (1,602.3 ± 59.5)	2.4 (181.3 ± 36.1)	166.7

3.3 Workload-Sensitive Variations of Top Knobs

Additionally, knob importance can vary dramatically between workloads. To demonstrate this, we drew 520 samples each via LHS and used CART to pick the Top 20 knobs (out of 52) for Sysbench, TPC-C, TPC-H workload. We then computed the Jaccard similarity between each pair of Top 20 sets and a general Top 20 knobs selected by expertise without workload specificity (so-called EXP) and presented in Table 5. Overall, these results reveal relatively low overlap among workloads and expert rankings. Contrary to our expectation that the two OLTP workloads (TPC-C and Sysbench) would overlap most, TPC-C actually shares more knobs with OLAP workload (TPC-H).

These findings show that a precise knob selection must be repeated for each workload—and if that step is too costly, it can bottleneck any DBMS auto-tuning approach. Yet we also uncovered a surprising degree of transferability. Table 6 compares Sysbench

Table 5: Jaccard index between knob sets.

	Sysbench	TPC-C	TPC-H	EXP
Sysbench	—	0.317 ± 0.035	0.325 ± 0.033	0.258 ± 0.016
TPC-C	0.317 ± 0.035	—	0.472 ± 0.041	0.243 ± 0.029
TPC-H	0.325 ± 0.033	0.472 ± 0.041	—	0.409 ± 0.024
EXP	0.258 ± 0.016	0.243 ± 0.029	0.409 ± 0.024	—

Table 6: Sysbench tuning with knob sets for different workloads ranked with CART (statistically significant at p < 5E-2)

	Rank (Max TPS) Test stat: 12.7, p-value: 2.7E-2	Rank (tuning time [min]) Test stat: 17.8, p-value: 3E-3
Sysbench	2.4 (1,708.0 ± 29.1)	2.4 (216.2 ± 32.6)
TPC-C	4.0 (1,653.2 ± 27.2)	3.0 (218.4 ± 65.3)
TPC-H	4.0 (1,638.8 ± 35.2)	2.8 (188.3 ± 57.9)
EXP	1.4 (1,725.2 ± 31.1)	2.6 (217.8 ± 42.7)
Shared	3.2 (1,677.1 ± 35.2)	5.8 (447.3 ± 26.2)
Random	6.0 (1,063.1 ± 82.9)	4.4 (308.8 ± 169.8)

throughput when tuned with 6 different knob sets. Specifically, we apply the top 20 knobs identified for each workload to optimize the Sysbench benchmark using BO:

- **Sysbench/TPC-C/TPC-H**: Top 20 knobs ranked by CART for the corresponding workload (520 LHS samples).
- **Exp**: Top 20 knobs ranked by experts.
- **Shared**: the 7 knobs appearing in all three Top 20 lists.
- **Random**: 20 knobs chosen at random from the full set.

To our surprise, the expert-ranked knob set (EXP) achieves the highest throughput (1,725 ± 31 TPS), while the Sysbench-specific knobs converge slightly faster (216.2 ± 32.6 min vs. 217.8 ± 42.7 min). However, their performance and tuning time are statistically equivalent according to the t-test—indicating that both represent strong, practical choices. Tuning the Sysbench knobs delivers excellent performance (1,708 ± 29 TPS), reinforcing the effectiveness of workload-specific ranking, while the EXP result highlights the potential of domain knowledge to generalize well.

Other workload-derived knob sets remain competitive: using the TPC-C and TPC-H rankings yields 1,653 ± 27 TPS and 1,639 ± 35 TPS, close to the expert-ranked result. Even the shared important knob set (7 knobs common to all workloads) remains close to the expert-level throughput (1,677 ± 35 TPS). In contrast, random knob selection performs poorly, losing nearly 40% of the attainable throughput (1,063 ± 83 TPS). For tuning time, the TPC-H knob set even outperforms others, reaching convergence in just 188.3 ± 57.9 minutes. However, restricting the search to shared knobs doubles the convergence time (447.3 ± 26.2 min), indicating that omitting workload-specific knobs can hinder optimizer guidance. Random selection also increases tuning duration (308.8 ± 169.8 min), despite the performance degradation.

While the knob sets derived from different workloads exhibit relatively low overlap, some of them achieve similar performance when transferred. This suggests that, despite divergent rankings, multiple knob subsets may still expose sufficiently informative dimensions of the configuration space for effective tuning. To summarize: **1) Cross-workload reuse is viable**: top knobs from a workload can be reused in another with modest loss in throughput. **2) Common knobs matter, but are not sufficient**: restricting to

universally important knobs retains much of the performance but incurs convergence delays. **3) Expert knowledge or workload-aware rankings remain best**: they yield the highest throughput and efficient tuning.

In practice, prior knowledge or expert insight can provide a useful warm start, skipping a full ranking phase. However, blindly reusing outdated or irrelevant knob sets risks missing critical parameters, leading to poor performance or longer tuning times.

3.4 Optimal Number of Tuned Knobs

Additionally, choosing how many knobs to tune is nontrivial: too few knobs may yield poor performance, while too many inflate overhead and extend search time. Table 7 reports DBMS performance when tuning different numbers of top knobs (expertise-based) using BO. On Sysbench, tuning only the Top 5 knobs yields lower performance than 10–30 knobs (or all 52) and does not minimize convergence time. The Top 20 set offers the best trade-off: near-optimal performance (comparable to 30 knobs) with the lowest tuning time. Although tuning times are statistically similar ($p > 0.05$), using all 52 knobs still incurs substantially longer convergence.

For TPC-C, tuning only five knobs performs significantly worse than larger sets. The Top 20 knobs give the best average performance, while the Top 10 offer the best speed–quality trade-off. As with Sysbench, tuning all knobs greatly prolongs convergence.

For TPC-H, performance is largely insensitive to knob count ($p > 0.05$). Tuning all knobs yields the best absolute result, but with heavy overhead, making the Top 5 knobs preferable for comparable performance at minimal cost.

These results underscore the challenge of tuning the right number of knobs to balance computational overhead against final performance, and highlight the different behavior for different workloads. Moreover, the optimal knob count (whether to minimize tuning time, maximize performance, or achieve both) varies across workloads. As a result, identifying a reasonable subset of knobs to balance performance gains against tuning cost often requires repeated experiments, making it difficult to apply in practice.

3.5 Summary of Knob Selection Challenges

To summarize, to achieve good DBMS performance tuning, knob selection is critical. However, in practice, it raises several challenges: 1) **Identifying the most influential knobs**—poor choices lead to sub-optimal performance and longer tuning cycles, and the cost of important knob identification is high; 2) **Workload dependency**—the set of top-ranked knobs varies by workload, indicating that knob importance depends not only on the DBMS itself but also on the specific workload characteristics; 3) **Transferability versus specificity**—although top-ranked knobs often transfer reasonably well across workloads, using workload-specific rankings yields better results; 4) **Sampling overhead**—robust knob selection requires extensive sampling to accurately rank parameters, which can be both time-consuming and resource-intensive; and 5) **Choosing how many knobs to tune**—balancing computational overhead, convergence time, and final performance is nontrivial—tuning too few knobs can hurt performance, while tuning too many increases search time without proportional gains. These observations highlight the need and the motivation for a more flexible and dynamic knob selection strategy.

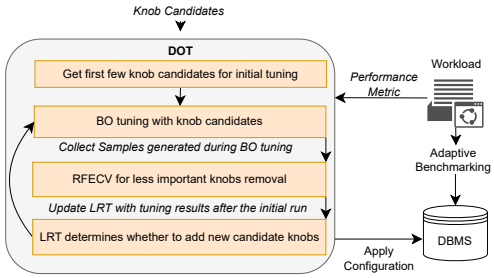

Figure 2: DOT overview

4 SYSTEM OVERVIEW

4.1 DOT

To address the above-mentioned practical challenges of knob selection in DBMS tuning, we present DOT—a dynamic and adaptive framework that continuously adjusts the active knob set during the tuning process for the given workload, focusing on the most impactful knobs as new evidence is collected (Figure 2). Unlike traditional methods that perform an up-front knob selection phase, DOT continuously updates the set of knobs being tuned *on the fly*. It uses RFECV to assess knob importance from performance samples gathered during the tuning process itself, eliminating the need for costly analysis. BO is applied to explore high-potential configurations efficiently. When available, DOT can use prior knowledge for more efficient tuning.

Algorithm 1 describes the procedure. Here, k_0 is the initial number of knobs to be tuned; it is set to 20 in our main experiments, following common practice in prior work ([7]) and to ensure fair comparison with baselines. Although k_0 is hard-coded during initialization, DOT is relatively insensitive to its specific value (see Section 5.7). If a ranked knob list is available, DOT tunes knobs in order of importance, beginning with the top k_0 knobs. In the absence of such prior knowledge, DOT instead initializes by randomly selecting k_0 knobs. By "sorted list" we mean knobs ordered by an estimated importance (e.g., from expert knowledge, documentation, or ranking heuristics), which allows DOT and other tuning algorithms to prioritize impactful knobs early and avoid wasted trials. We distinguish between two practical contexts: (i) when prior knowledge is available (e.g., a knowledgeable user or previously studied workload), DOT and baselines start from this ranked list; (ii) when no such knowledge is available (e.g., less experienced users or unseen workloads), DOT proceeds without ordering and relies entirely on its internal mechanism to discover impactful knobs.

BO with surrogate model Gaussian Process tunes over an epoch with a length of 5 times the number of newly introduced knobs. During each iteration, we generate one configuration–performance sample (x, y), yielding five samples per knob to ensure stability and reliable knob selection for RFECV (Section 4.3). After the epoch completes, RFECV is applied to the collected samples to identify the most impactful knobs. Next, a likelihood-ratio test (LRT; Section 4.4)

Table 7: Performance, tuning time, and Friedman-test ranks for different numbers of knobs (statistically significant at p < 5e-2)

	Sysbench		TPC-C		TPC-H	
	Rank (Max TPS) Test stat: 13.3 p-value: 1E-2	Rank (tuning time [mins]) Test stat: 9.1 p-value: 5.8E-2	Rank (Max TPS) Test stat: 12.0 p-value: 1.7E-2	Rank (tuning time [mins]) Test stat: 5.1 p-value: 2.8E-1	Rank (Exec (s)) Test stat: 6.1 p-value: 1.9E-1	Rank (tuning time [mins]) Test stat: 11.8 p-value: 1.9E-2
Top 5 knobs	4.6 (1,486.8 ± 159.0)	3.0 (311.5 ± 194.2)	5.0 (120.5 ± 22.1)	2.4 (151.7 ± 161.1)	3.6 (47.7 ± 0.3)	1.6 (46.1 ± 54.5)
Top 10 knobs	4.2 (1,674.9 ± 34.4)	2.6 (272.0 ± 122.8)	3.2 (338.5 ± 13.8)	2.0 (153.8 ± 186.7)	3.8 (47.7 ± 0.5)	2.0 (32.7 ± 14.6)
Top 20 knobs	2.0 (1,725.2 ± 31.1)	2.0 (217.8 ± 42.7)	1.8 (354.2 ± 14.8)	3.4 (411.9 ± 264.9)	3.2 (47.8 ± 1.4)	3.2 (73.0 ± 53.2)
Top 30 knobs	2.0 (1,728.4 ± 46.3)	2.6 (267.9 ± 136.3)	2.4 (347.6 ± 25.8)	3.2 (261.0 ± 287.1)	2.8 (47.8 ± 2.0)	3.6 (189.6 ± 286.5)
All 52 knobs	2.2 (1,720.0 ± 33.0)	4.8 (732.6 ± 266.7)	2.6 (346.0 ± 7.5)	4.0 (560.4 ± 327.7)	1.6 (46.5 ± 0.6)	4.6 (334.1 ± 175.5)

decides whether to continue fine-tuning the reduced knob set or to expand the search space by appending additional knobs. The LRT's reward signal is the per-call performance gain: if exploiting the current set is statistically justified, DOT proceeds with BO on those knobs; otherwise, it explores by adding more knobs.

Algorithm 1 DOT: Dynamic Knob Tuning

Require: Sorted knob list $\mathcal{K}$; tuning budget $T_{\max}$; initial size k_0=20; step size Δk=5

procedure DOT($\mathcal{K}, T_{\max}, \Delta k$)
  $\mathcal{K}_{\text{curr}} \leftarrow$ top-$k_0(\mathcal{K})$
  $\mathcal{D} \leftarrow \varnothing$ ▷ full-space observations (x, y)
  $b \leftarrow -\infty$ ▷ best performance so far
  $E \leftarrow 5 \cdot k_0$ ▷ initial epoch length
  elapsed $\leftarrow 0$
  while elapsed $< T_{\max}$ **do**
    Step 1: BO Exploration ▷ Explore current subspace with BO, record best value and update full observations
    $(\mathcal{D}, y^*) \leftarrow \text{BO}(\mathcal{K}_{\text{curr}}, E, \mathcal{D} \to \mathcal{D}_{\text{proj}})$
    Step 2: Knob Selection ▷ Select most relevant knobs via RFECV in current subspace
    $\mathcal{K}^* \leftarrow \text{RFECV}(\mathcal{K}_{\text{curr}}, \mathcal{D} \to \mathcal{D}_{\text{proj}})$
    Step 3: Knob Set Expansion ▷ Decide shrink/expand using likelihood ratio test
    $a \leftarrow \text{LRT\_Step}(b, y^*, E)$; $b \leftarrow \max(b, y^*)$
    $\mathcal{K}_{\text{curr}}, E \leftarrow \begin{cases} (\mathcal{K}^*, 10), & a = 0 \\ (\mathcal{K}^* \cup \text{next } \Delta k \text{ knobs}, 5 \cdot \Delta k), & a = 1 \end{cases}$
    ▷ $a = 0$: Shrink/Stay, $a = 1$: Expand
    elapsed $\leftarrow$ elapsed + E
  end while
  return best configuration found in $\mathcal{D}$
end procedure

Additionally, to reduce heavy benchmarking time, we introduce an adaptive benchmark budget to reduce tuning time. This process repeats until the overall evaluation budget $T_{\max}$ is exhausted. Throughout, DOT reuses prior samples $(x, y) \in \mathcal{D}$ that are projected to the current knob set $\mathcal{D}_{\text{proj}}$, as new knobs are initially set to default values—ensuring that earlier configurations remain valid for initializing BO when the knob set $\mathcal{K}_{\text{curr}}$ is updated.

This design lets DOT dynamically adjust tuning complexity, balance exploration versus exploitation, and efficiently converge on optimal configurations. DOT—comprising tuning, assessing knob importance, and refining the search space—follows these principles:

(1) **Exploration when impact is limited:** If recent tuning yields minimal improvement, the LRT policy triggers expansion by appending Δk new knobs from the ranked list $\mathcal{K}$, increasing the tuning space.
(2) **Exploitation when knobs are effective:** If current knobs drive sufficient gains, RFECV prunes them to retain the influential knobs– reducing dimensions and improving convergence.
(3) **Dynamic re-evaluation:** Knob importance is continuously revisited after each epoch, allowing DOT to drop previously selected knobs when more impactful ones emerge.
(4) **No warm-up required:** DOT starts tuning immediately without a separate knob-selection phase. If prior ranking is available, it can be used to initialize $\mathcal{K}$, focusing early tuning on high-potential knobs while still supporting pruning and exploration.

By integrating BO with RFECV and LRT in a feedback-driven cycle, DOT allocates tuning resources where they yield the greatest gains—minimizing overhead and maximizing performance across diverse DBMS workloads. It thus addresses the following challenges: 1) **Identifying impactful knobs** — Done iteratively via RFECV using samples collected during tuning with minimal overhead. 2) **Workload dependency** — Adaptively adjusts based on workload specific feedback. 3) **Transferability vs. specificity** — Can use prior rankings but corrects them if needed via RFECV. 4) **Sampling overhead** — No up-front selection phase; tuning and selection are merged. 5) **Choosing how many knobs** — Controlled dynamically via LRT decisions and RFECV pruning.

4.2 Bayesian Optimization via Gaussian Process

BO with GP[35] surrogate is our tuning algorithm of choice for expensive, noisy objectives [24, 31, 37, 41, 47]. In DOT, BO is initialized with the current knob set $\mathcal{K}_{\text{curr}}$, the number of optimization iterations E, and the set of prior samples $\mathcal{D}$ projected onto $\mathcal{K}_{\text{curr}}$ as $\mathcal{D}_{\text{proj}}$ (or to sample 10 random configurations if $\mathcal{D}$ is empty). BO first fits the GP model to $\mathcal{D}_{\text{proj}}$, then for each of the E iterations selects the next configuration via an acquisition function, evaluates it on the DBMS using adaptive benchmarking (Section 4.5), and augments $\mathcal{D}_{\text{proj}}$ with the resulting performance measurements. After E rounds, BO returns the enriched set of configuration–performance pairs, often finding better configurations with far fewer evaluations than grid or random search, thereby reducing computational cost and enabling effective tuning under resource constraints.

4.3 Recursive Feature Elimination with Cross-Validation

RFECV is a widely adopted algorithm for feature selection in ML projects [25]. DOT uses it to prune tuning knobs prior to optimization. RFECV is initialized with the current knob set $\mathcal{K}_{\text{curr}}$ and the projected sample set $\mathcal{D}_{\text{proj}}$. An estimator—e.g., a random forest with CART as in [7]—is trained on all candidate knobs; the least important knob(s) are removed and the model re-evaluated via cross-validation. Unlike plain CART, which only ranks knobs and requires an arbitrary cutoff, RFECV leverages cross-validation to automatically identify the optimal subset by iteratively eliminating features until the highest average CV score is achieved. To prevent over-pruning and ensure sufficient search diversity, we enforce a minimum of 10 knobs in the pruned subset (avoid over-pruning to preserve BO diversity). By discarding knobs that minimally impact DBMS performance, RFECV reduces dimensionality and accelerates subsequent tuning. We adopt Scikit-Learn's RFECV implementation in DOT to focus the search on the most impactful knobs, thereby enabling faster optimizer convergence.

4.4 Likelihood-Ratio Test for Set Expansion

In DOT, LRT[43] is used to deterministically compare empirical success rates to select between two actions: **(0)** stay with current knobs or **(1)** expand the knob set. Each action $i \in \{0, 1\}$ tracks success and failure counts (s_i, f_i), from which it computes a smoothed empirical success rate $\hat{p}_i = s_i/(s_i + f_i)$. The decision is based on:

$$\Lambda \leftarrow \ln(\hat{p}_1/\hat{p}_0), \quad \text{action} = \mathbb{1}[\Lambda > 0].$$

This lightweight, adaptive rule requires only four scalars and no exploration schedule.

Reward function. Given best performance b_{t-1}, new value c_t, and step count n_t, define the per-call improvement as $g_t = (c_t - b_{t-1})/(b_{t-1} \cdot n_t)$. A binary reward is given by:

$$r_t = \mathbb{1}[g_t > \delta], \quad \text{e.g., } \delta = 0.001.$$

We empirically set the threshold to 0.001 because, over 100 iterations, it corresponds to roughly a 10% performance improvement, which is significant even with benchmarking noise. The chosen action's success/failure counts are updated accordingly. LRT selects to expand or to stay with the current knob set based on the feedback.

The rationale behind the LRT is to drive exploration when the initial knob set yields poor performance improvement and to favor exploitation when the increase of performance is strong, avoiding unnecessary expansion of the search space, which would incur additional computational overhead.

4.5 Adaptive Benchmarking

Benchmarking can dominate tuning time —e.g., 69.5% for OLTP Sysbench with 20 knobs under BO, and 57% for OLAP TPC-H scale-1 with 20 knobs under BO. To address this, DOT employs an *adaptive benchmarking* that early-terminates runs unlikely to surpass the best configuration, using partial results as proxies. Below, we outline this approach for OLTP and OLAP workloads.

OLTP Benchmarks. For workloads like Sysbench and TPC-C, performance is measured as steady-state throughput (TPS) after a warm-up phase. In our experiments, the canonical benchmark runs

Algorithm 2 LRT for Knob Set Expansion

Require: Best past performance $b \in \mathbb{R}$; Best observed performance in the steps $y^* \in \mathbb{R}$; Number of iterations $E \in \mathbb{N}$

State: Success/Failure counts s_0, f_0, s_1, f_1, all initialized to 1; Flag *init* $\leftarrow$ True
Parameter: reward threshold $\delta \leftarrow 0.001$
procedure LRT_Step(b, y^*, E)
  if *init* **then**
    Draw $a_t \sim$ Bernoulli(0.5)
    init $\leftarrow$ False
    return a_t
  end if
  $\hat{p}_0 \leftarrow \dfrac{s_0}{s_0 + f_0}, \quad \hat{p}_1 \leftarrow \dfrac{s_1}{s_1 + f_1}$
  $\Lambda \leftarrow \ln(\hat{p}_1/\hat{p}_0)$
  $a_t \leftarrow \mathbb{1}[\Lambda > 0]$
  $g_t \leftarrow \dfrac{y^* - b}{b\,E}, \quad r_t \leftarrow \mathbb{1}[g_t > \delta]$
  $s_{a_t} \leftarrow s_{a_t} + r_t, \quad f_{a_t} \leftarrow f_{a_t} + (1 - r_t)$
  return a_t
end procedure

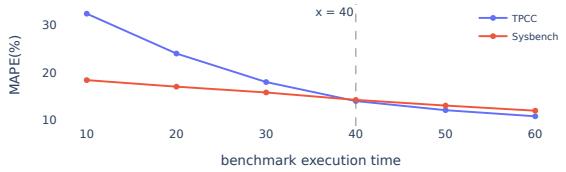

Figure 3: MAPE of Partial vs. Full Benchmark Results

for $T_{\max} = 90$ s, using only the final 30 s to compute the *ground-truth* TPS. To mitigate this, we run an early-cut window of $T_{\text{cut}} = 40$ s. If the average TPS at T_{cut} already exceeds the current best, we let the benchmark continue to $T_{\max}$ for a precise measurement; otherwise, we abort and use the partial result as a (noisier) proxy . Figure 3 shows the *Mean Absolute Percentage Error* (MAPE) between partial and full benchmark execution with more than 2,000 runs under various configurations. We observe that, at $T_{\text{cut}} = 40$ s, the 95$^{\text{th}}$-percentile MAPE is only 13.7 % for TPC-C and 14.2 % for Sysbench, with little benefit beyond this point.

OLAP Benchmarks. For workloads such as TPC-H, performance is the total execution time of a set of analytical queries, without warm-up. We randomly sample a subset of queries (40% in our experiments) and track their execution time with the current configuration. If this partial workload completes faster than the best

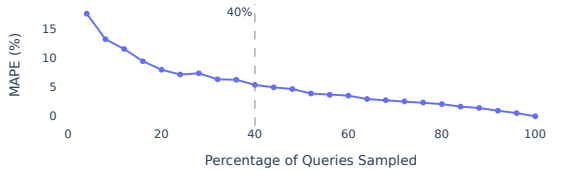

Figure 4: MAPE vs. Benchmark-Query Coverage

total time, we run the full workload to obtain the execution time; otherwise, we log the partial result as a proxy. Figure 4 quantifies the error due to sampling over 1,000 logs under different configurations: sampling more queries reduces error, and 40% sampling strikes a balance between accuracy and overhead reduction.

Note that while we empirically set our budget (T_{cut} = 40s and 40% target query coverage) using historical data, it can be estimated from a few pilot measurements at minimal cost—Section 5.7 shows that performance is insensitive to the exact budget chosen.

5 EXPERIMENTAL RESULTS

Besides DOT, we implemented two complementary baselines that *dynamically* adjust the number of tuned knobs so as to lower the dimensionality of the search space and enhance the tuning speed.

Incremental Knob Tuning. We follow OtterTune's progressive expansion: BO starts on the top-4 knobs for 25 iterations, then appends the next two per epoch while reusing all observations; the process runs until the budget ends. This keeps early search low-dimensional but can stall if early rankings omit a key knob.

Statistical Elimination (SE). SE proceeds inversely: start with all knobs, run $2N$ BO evaluations, then drop knobs that fail a per-knob Welch t-test (α=0.05); BO continues only on the survivors (others are frozen at their best value). This upfront pruning reduces dimensionality and speeds subsequent search.

5.1 Experimental Setup & Results

We evaluate DOT's effectiveness and efficiency on the Sysbench, TPC-C, and TPC-H benchmarks, using the same workloads, DBMS, and hardware settings as in Section 3. Global space is also capped at the top 52 expert-selected parameters to simulate a practical tuning scenario. We compare DOT against:

- **BO**: Classical GP-BO as in iTuned [37] and OtterTune [41] incorporated with different knob sets: top 20 knobs ranked by experts (EXP), top 20 knobs ranked by CART model with 10 samples per knob (CART), and all the knobs in the scope (full);
- **Incremental Knob Tuning/ Statistical Elimination**: Our two dynamic-search-space variants;
- **CDBTune**: Author's public implementations of CDBTune [18], the implementation currently only supports OLTP;
- **DB-Bert**: Author's implementations of DB-Bert [39], a LLM-based tuner, the implementation currently only supports OLAP;
- **LLM knowledge based**: An LLM-based baseline adapted from GPTuner [22], where we modified the prompt of GPTuner's knob selection phase to let ChatGPT decide both which knobs and how many to tune.

Other methods, like Hunter, DDPG++(optimized version of CDBTune proposed in [42]), and UDO, are not included in our comparison: Hunter, DDPG++ lacks a public implementation, while UDO focuses on index tuning outside our scope.

Section 3 showed that supplying a pre-ranked knob list greatly improves DBMS tuning by transferring prior knowledge, yet producing such rankings is costly, and they may be inaccurate. Consequently, we run each algorithm in two scenarios: **Expert ranking available.** We feed a trusted expert ranking to DOT, Incremental Tuning, BO limited to the Top 20 expert-ranked knobs, CDBTune restricted to those 20 knobs, and DB-BERT with its documentation-derived priors; **Erroneous or missing ranking.** We provide a randomly permuted list (equivalent to no ranking) and evaluate DOT, Incremental Tuning, BO over the random list, BO over the full 52-knob space, BO guided by LLM-selected knobs, and BO+CART (counting CART's ten samples per knob as a 14.4h overhead). SE, which does not use rankings, is applied in both scenarios.

5.2 Including Knob Importance Knowledge

Figure 5 shows best–so–far performance versus tuning time for Sysbench, TPC–C and TPC–H when knob importance knowledge is available. On Sysbench, DOT+EXP reaches its optimal plateau faster than any other method and delivers throughput indistinguishable from BO+EXP. Other approaches—Incremental+EXP, CDBTune, and Statistical Elimination—settle at lower performance. CDBTune's variability stems from its cold start; although a warm-up phase can improve results, it adds significant overhead.

On TPC-C, DOT+EXP not only converges quickly but also outperforms BO+EXP by a statistically significant margin, thanks to its iterative knob selection and occasional exploration of other knobs. All other methods remain clearly behind. On TPC-H, confidence intervals overlap across methods, reflecting this workload's relative insensitivity to tuning. Nevertheless, DOT+EXP still attains the lowest execution time, even if practical differences are small. These results show that, when knob-ranking priors are available, DOT+EXP consistently delivers the fastest convergence and either matches or exceeds the final performance of state-of-the-art tuners.

5.3 Excluding Knob Importance Knowledge

Figure 6 plots best–so–far performance when no knob-importance prior is available. DOT begins by exploring random knobs before focusing on the most important ones. In Sysbench and TPC-C, DOT+RAN still converges the fastest. In Sysbench, it matches BO+All knobs and BO+CART and on TPC–C, it achieves a statistically superior throughput. For TPC-H, differences between methods are minor, though DOT+RAN attains the best average execution time. BO+CART climbs steeply once tuning starts—reflecting CART's strong knob selection—but its costly pre-sampling adds significant overhead. BO-All knobs eventually plateaus alongside DOT, but only after a lengthy exhaustive search. BO-All knobs achieves similar performance as DOT yet takes a longer time; note that Incremental+EXP follows a similar staircase pattern with a lower ceiling as DOT. Both Incremental and DOT gradually incorporate new knobs, but DOT reaches higher scores more quickly.

We can also observe that DOT+RAN initially exhibits high variance, which steadily decreases as tuning progresses; as DOT explores more knobs, the set eventually converges to the most important parameters (Section 5.5).

5.4 Comparison of All Methods & Situations

Table 8 presents Friedman-test ranks for both performance (throughput or execution time) and tuning time across Sysbench, TPC–C and TPC–H. In every case, DOT variants occupy the top ranks and dramatically reduce tuning effort compared to BO+EXP, as it has a generally good performance across workloads, while other baselines lag in quality, speed, or both.

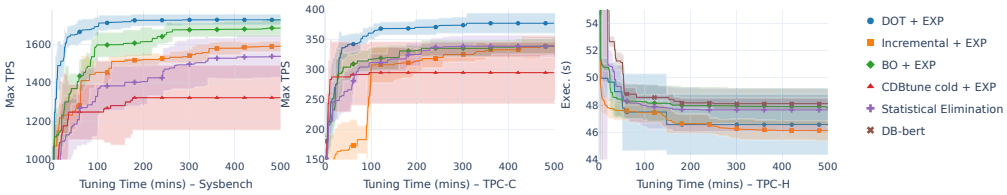

Figure 5: Performance vs. tuning-time of different algorithms on Sysbench, TPC-C, TPC-H with knob importance knowledge

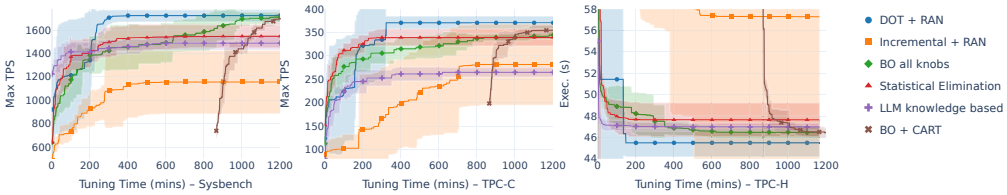

Figure 6: Performance vs. tuning-time of different algorithms on Sysbench, TPC-C, TPC-H without knob importance knowledge

Table 8: Performance, tuning time, and Friedman-test ranks for different methods (statistically significant at $p < 5E\text{-}2$)

	Sysbench		TPC-C		TPC-H	
	Rank (Max TPS) Test stat: 37.4 p-value: 2.2E-05	Rank (tuning time [mins]) Test stat: 32.2 p-value: 1.9E-04	Rank (Max TPS) Test stat: 31.4 p-value: 2.5e-04	Rank (tuning time [mins]) Test stat: 37.9 p-value: 1.7E-05	Rank (exec [s]) Test stat: 18.382 p-value: 3.1e-02	Rank (tuning time [mins]) Test stat: 38.4 p-value: 1.5E-05
DOT+EXP	2.6 (1728.3 ± 28.3)	2.0 (57.7 ± 28.2)	1.2 (377.7 ± 16.9)	2.4 (121.6 ± 72.1)	4.4 (46.6 ± 2.2)	3.4 (57.6 ± 54.3)
DOT+RAN	3.0 (1724.7 ± 50.6)	4.2 (171.1 ± 121.2)	2.0 (370.8 ± 13.6)	3.8 (187.3 ± 107.5)	2.6 (45.5 ± 1.3)	2.0 (33.9 ± 63.9)
LLM knowledge based	8.0 (1489.1 ± 57.5)	3.4 (129.3 ± 99.1)	9.2 (264.9 ± 10.1)	5.4 (288.0 ± 207.9)	6.6 (47.0 ± 0.4)	2.0 (21.1 ± 10.0)
BO+CART	3.6 (1708.0 ± 29.1)	10.0 (1080.3 ± 32.6)	5.0 (356.1 ± 9.4)	9.8 (960.4 ± 57.6)	3.6 (46.4 ± 0.5)	10.0 (1027.4 ± 106.6)
CDBTune cold+EXP	8.6 (1358.1 ± 152.9)	3.2 (218.6 ± 310.1)	8.2 (295.2 ± 51.7)	1.2 (33.5 ± 40.9)	-	-
DB-bert	-	-	-	-	8.6 (48.1 ± 0.4)	5.0 (114.0 ± 58.3)
Statistical Elimination	7.2 (1548.4 ± 97.3)	4.8 (231.8 ± 103.5)	6.4 (339.4 ± 18.2)	3.2 (155.7 ± 50.0)	6.0 (47.6 ± 1.6)	4.2 (79.8 ± 52.5)
BO All Knobs	3.0 (1720.0 ± 33.0)	8.8 (732.6 ± 266.7)	6.0 (348.4 ± 10.1)	7.8 (560.4 ± 327.7)	4.4 (46.5 ± 0.6)	7.8 (334.1 ± 175.5)
BO+EXP	2.8 (1725.2 ± 31.1)	5.0 (217.8 ± 42.7)	4.8 (354.2 ± 14.8)	6.2 (411.9 ± 264.9)	7.4 (47.8 ± 1.4)	4.6 (73.0 ± 53.2)
Incremental+RAN	9.6 (1157.2 ± 269.5)	7.0 (353.9 ± 86.0)	7.4 (281.6 ± 86.8)	8.0 (625.1 ± 113.6)	7.2 (57.3 ± 14.5)	8.6 (432.2 ± 74.0)
Incremental+EXP	6.6 (1620.0 ± 45.7)	6.6 (260.2 ± 82.7)	4.8 (352.5 ± 18.5)	7.2 (434.4 ± 149.0)	4.2 (46.1 ± 0.7)	7.4 (236.5 ± 40.5)

On Sysbench, DOT+EXP achieves a performance rank of 2.6 and a tuning-time rank of 2.0, delivering 1 728.3 ± 28.3 TPS in 57.7 ± 28.2 min versus BO+EXP's 1 725.2 ± 31.1 TPS in 217.8 ± 42.7 min—cutting tuning time by 73.6%. Even without priors, DOT+RAN (rank 3.0/4.2) matches final throughput (1 724.7 ± 50.6 TPS) in 171.1 ± 121.2 min, outperforming other baselines and matches BO+EXP.

On TPC-C, DOT+EXP leads with ranks 1.2 (performance) and 2.6 (time), achieving 377.7 ± 16.9 TPS in 121.6 ± 72.1 min—a 70.5% reduction in tuning time compared to BO+EXP's 411.9 ± 264.9 min. DOT+RAN (2.0/3.8) converges to 370.8 ± 13.6 TPS in 187.3 ± 107.5 min, while BO+CART, CDBTune, and DB-bert remain behind.

On TPC-H, DOT+RAN ranks 2.6/2.0 by reaching 45.5 ± 1.3 s in 33.9 ± 63.9 min, surprisingly edging DOT+EXP (4.4/3.4) at 46.6 ± 2.2 s in 57.6 ± 54.3 min due to TPC-H's low tuning sensitivity for different knobs. Both outperform BO+EXP (7.4/4.6) by shaving 21.1% (DOT+EXP) and 53.6% (DOT+RAN) off its tuning time.

State-of-the-art methods, like CDBTune and DB-BERT perform poorly here because we deliberately simulate a low-knowledge scenario. CDBTune requires an extended warm-up phase to gather data, and DB-BERT needs more complete documentation and carefully tuned hyperparameters. Under our "cold start" conditions, neither can exploit such prior information, so their performance lags. **LLM-based methods** face similar limitations. Approaches such as GPTuner-style prompt ranking rely on access to comprehensive DBMS manuals and often require multiple exchanges with the LLM to refine the ranking. While these techniques can reduce sampling overhead when full documentation is available, in our cold-start setting, they cannot leverage sufficient prior knowledge, leading to weaker performance.

Across all benchmarks, DOT+EXP and DOT+RAN not only secure the best or near-best ranks but also slash tuning time by up to 73.6% (on Sysbench) relative to BO+EXP, underscoring their efficiency and robustness compared to every other method.

These results underscore two key strengths of DOT. First, even without priors, DOT+RAN can match or exceed BO+EXP by dynamically discovering and exploiting the most influential knobs based on the current workload. Second, when knob-importance priors are available, DOT+EXP leverages them to converge even faster—on Sysbench and TPC-C with minimal overhead compared to DOT+RAN. In both cases, DOT delivers high-quality tuning with little extra cost, making it a practical choice whether or not prior knowledge is at hand.

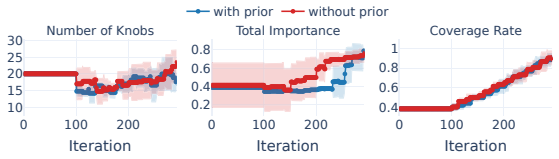

Figure 7: DOT behavior on Sysbench workload with and without knob importance prior

5.5 DOT Behavior Analysis

Figure 7 shows DOT's tuning behavior on a Sysbench workload, comparing runs with an expert-ordered knob-importance prior versus a randomized start (no prior).

Dynamic Tuned Knobs Count. As iterations proceed, DOT applies RFECV to prune low-value knobs and uses LRT to decide when to expand its search. Without a prior, more knobs are eliminated in early phases, since the random initial set has a lower average importance. In both setups, DOT settles on tuning roughly 20 knobs to avoid excessive overhead.

Total Importance of Tuned Knobs. With a prior, the cumulative importance of the active knobs remains nearly constant at first—high-impact variables cycle in and out but stay within the tuning pool—and then increases in a step-wise fashion. Without a prior, the initial knob set exhibits higher variance in importance, and its cumulative score rises quickly, eventually converging with the prior case. This occurs because, freed from expert bias, the no-prior run discovers impactful knobs discovered by CART that the expert had down-ranked. This highlights the trade-off between transferability and specificity. Incorporating a prior on knob importance can come at the expense of specificity, whereas a purely random search may more quickly home in on the most critical knobs. However, even though the "no-prior" approach can yield higher importance scores, it does not necessarily translate into better final performance, since CART-derived importances are not guaranteed to reflect the true ground truth. By the end, both algorithms retain about 20 knobs whose total importance exceeds 0.8 (out of 1).

Coverage Rate. Under both scenarios, DOT covers the majority of the knob space. The no-prior case achieves higher coverage initially, demonstrating that the LRT favors exploration in the absence of informative priors while concentrating exploitation when such information is available. And once exploitation yields diminishing returns, DOT transitions to exploration for further fine-tuning.

In summary, DOT meets its goals—balancing exploration and exploitation, keeping the search space compact, and minimizing overhead—while preserving key knobs.

5.6 Periodic Workload

In addition to static workloads, we also evaluate DOT under a periodic workload mixture, where execution alternates between TPC-C and Sysbench following the protocol of [15]. Figure 8 illustrates this dynamic scenario. The DOT Adaptive curve shows the behavior of the tuning procedure, which adapts dynamically to workload changes and progressively accelerates convergence to high-performance configurations after each shift. The dotted curve represents the number of knobs modified between consecutive iterations. It initially shows a high volume of changes to adapt to both

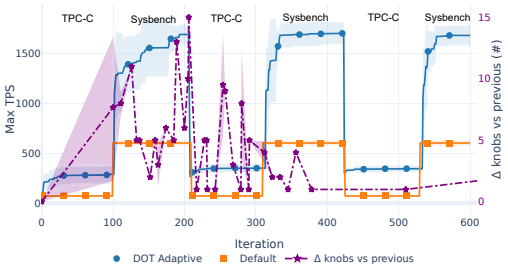

Figure 8: Tuning Performance of DOT on periodic workloads composed of TPC-C and Sysbench

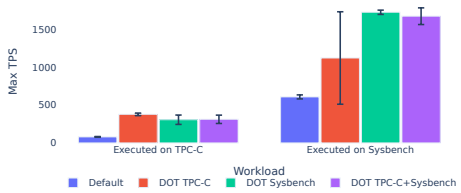

Figure 9: Max TPS Across Configurations and Workloads

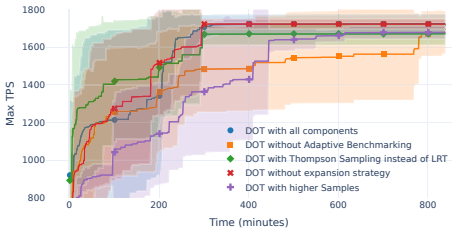

Figure 10: Tuning Performance of DOT on Sysbench for various configurations without knob importance knowledge

workloads, which eventually stabilizes, indicating that DOT adapts effectively while converging toward a stable, efficient knob set.

Figure 9 shows the best configuration found by DOT when trained on the periodic workload, denoted as DOT TPC-C+Sysbench. Its throughput is lower than that of workload-specific configurations, as it must jointly optimize across multiple workloads without a mechanism to distinguish them. Nevertheless, DOT TPC-C+Sysbench consistently outperforms both the default configuration and mismatched static baselines, and achieves performance that is statistically indistinguishable from DOT SYSBENCH. This demonstrates that DOT can be effectively trained on workload mixtures, delivering robust performance.

5.7 Ablation Study

In this section, we evaluate the performance of DOT on Sysbench by ablating its key components. Figure 10 demonstrates DOT performance under different components for Sysbench workload without any prior knob-importance knowledge:

(1) **DOT without adaptive benchmarking:** Removing adaptive benchmarking slows convergence: the best-so-far curve climbs more gradually and only reaches its plateau later, although the final throughput remains unchanged. This demonstrates DOT's resilience to noisy performance estimates.

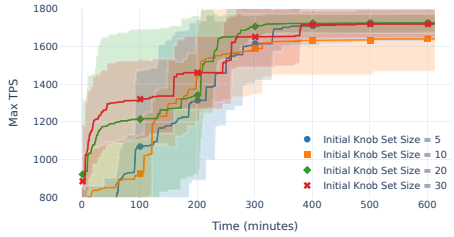

Figure 11: Tuning Performance of DOT on Sysbench for various k_0 without knob importance knowledge

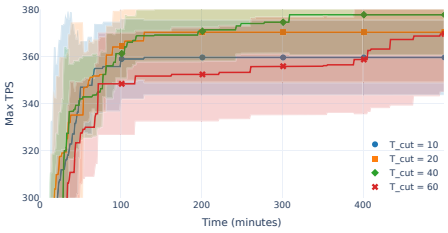

Figure 12: Tuning Performance of DOT on TPC-C for various T_{cut} budget with knob importance knowledge

(2) **DOT with Thompson sampling / without expansion strategy:** Replacing our LRT-based knob set expansion strategy with Binary Thompson Sampling [4], or eliminating expansion strategy (i.e., exploring knobs purely incrementally), yields virtually statistically equivalent results according t-test. This indicates that DOT's overall framework is robust to the choice of knob-selection heuristic. Naturally, purely incremental exploration performs well when no initial knob ranking is available, but LRT helps prevent over-exploration when knob-importance priors exist (Section 5.5). Thus, LRT serves as a generally applicable method, though other strategies can be plugged into the DOT framework to suit specific tuning scenarios.

(3) **DOT with higher samples:** Doubling BO epochs per knob—from 5 to 10—to feed RFECV more samples for pruning delivers no performance gain but delays convergence to the final performance plateau. Therefore, 5 epochs per knob offer a practical trade-off between RFECV accuracy and run-time efficiency.

Similarly, Figure 11 shows the performance of DOT under different initial knob sizes (k_0) on a Sysbench workload without prior knowledge. Larger k_0 values (20, 30) yield stronger initial performance, whereas smaller values (5, 10) start lower but still converge at roughly the same rate. This suggests that DOT remains insensitive to the initial number of knobs, though selecting an appropriate value (e.g., 20) can provide an early advantage.

Additionally, we evaluated DOT on TPC-C (which had the highest MAPE in Section 4.5) using prior knob-importance knowledge and varying the cutoff budget T_{cut}. Figure 12 shows that larger T_{cut} values slow convergence—since each benchmark run takes longer (e.g., $T_{\text{cut}} = 60$ converges much later than $T_{\text{cut}} = 10$). However, even a tiny budget ($T_{\text{cut}} = 10$) delivers final tuning quality nearly indistinguishable from that of high budgets for this **most noisy** benchmark. This insensitivity arises because our allocator concentrates precise measurements on promising configurations and uses

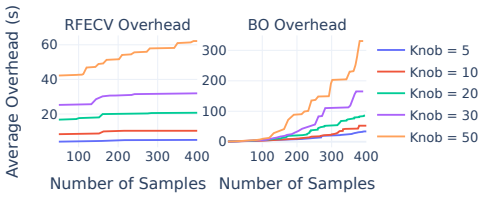

Figure 13: Bayesian optimization & RFECV overhead

coarser estimates elsewhere, letting BO exploit the best regions even without a finely tuned overall budget. Although approximate benchmarking can weaken the surrogate and require additional BO iterations—so a carefully chosen budget may yield faster convergence and better results—the end-to-end tuning time and ultimate performance remain rather robust to different T_{cut}. To summarize, these components collectively provide a robust, general-purpose solution for DBMS tuning.

5.8 System Overhead

The BO shows rapidly growing cost when the number of samples increases: once the knob count exceeds about 20, the GP-fit and acquisition step dominates, climbing from seconds to several minutes as samples accumulate (Figure 13). By contrast, RFECV—used only to prune knobs before the main search—adds a small, knob-dependent fixed tax that flattens after the first hundred samples and remains almost insensitive to further data, resulting in relatively low overhead. Across all tested scenarios, benchmarking dominates the wall-clock budget (56%), BO bookkeeping follows at around 27%, RFECV pruning contributes only about 0.4%, and the remaining other tasks (logging, configuration deployment, DBMS restarts, LRT training and inference) consume roughly 16%. Thanks to LRT's simple structure, its training and inference cost is negligible (<0.01%). This lightweight overhead of RFECV and LRT makes DOT more adaptable for wider adoption.

6 CONCLUSION & FUTURE WORKS

We introduced DOT, a database autotuning framework that needs neither upfront training nor knob-importance priors, yet can exploit them when available. Combining Bayesian optimization for strong proposals, RFECV with online sampling to prune low-impact knobs, and LRT to balance exploration and exploitation, DOT maintains a compact search space and avoids the curse of dimensionality. Across diverse workloads, it matches or surpasses state-of-the-art tuners while cutting tuning time by orders of magnitude. Even without reliable importance scores, DOT consistently identifies and focuses on the most impactful knobs.

Future directions include: (i) incorporating workload characterization [21, 28, 41] and other optimzaition objectives to better adapt to dynamic environments; (ii) exploring additional component optimizations or alternative heuristics to extend applicability; (iii) integrating more advanced hyperparameter optimization engines [49]; and (iv) validating DOT's scalability on real-world workloads and across multiple DBMSs. Overall, DOT demonstrates that adaptive dimensionality reduction combined with principled exploration can make database autotuning both efficient and practical.

REFERENCES

[1] 2025. MySQL. https://www.mysql.com/. Accessed: 24 March 2025.

[2] Bilge Acun, Phil Miller, and Laxmikant V. Kale. 2016. Variation Among Processors Under Turbo Boost in HPC Systems. In *Proceedings of the 2016 International Conference on Supercomputing* (Istanbul, Turkey) *(ICS '16)*. Association for Computing Machinery, New York, NY, USA, Article 6, 12 pages. https://doi.org/10.1145/2925426.2926289

[3] TPC Professional Affiliates. 1992. TPC-C is an On-Line Transaction Processing Benchmark. https://www.tpc.org/tpcc/

[4] Shipra Agrawal and Navin Goyal. 2012. Analysis of Thompson Sampling for the Multi-armed Bandit Problem. In *Proceedings of the 25th Annual Conference on Learning Theory (Proceedings of Machine Learning Research)*, Shie Mannor, Nathan Srebro, and Robert C. Williamson (Eds.), Vol. 23. PMLR, Edinburgh, Scotland, 39.1–39.26. https://proceedings.mlr.press/v23/agrawal12.html

[5] Theophilus A Benson, Ashok Anand, Aditya Akella, and Ming Zhang. 2009. Understanding Data Center Traffic Characteristics. In *ACM SIGCOMM Workshop: Research on Enterprise Networking* (acm sigcomm workshop: research on enterprise networking ed.). Association for Computing Machinery, Inc. https://www.microsoft.com/en-us/research/publication/understanding-data-center-traffic-characteristics/

[6] L. Breiman, J. Friedman, R. A. Olshen, and C. J. Stone. 1984. *Classification and Regression Trees* (1st ed.). Chapman and Hall/CRC. https://doi.org/10.1201/9781315139470

[7] Baoqing Cai, Yu Liu, Ce Zhang, Guangyu Zhang, Ke Zhou, Li Liu, Chunhua Li, Bin Cheng, Jie Yang, and Jiashu Xing. 2022. HUNTER: An Online Cloud Database Hybrid Tuning System for Personalized Requirements. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 646–659. https://doi.org/10.1145/3514221.3517882

[8] Jeffrey Dean and Luiz André Barroso. 2013. The tail at scale. *Commun. ACM* 56, 2 (2013), 74–80. https://doi.org/10.1145/2408776.2408794

[9] Biplob Debnath, David Lilja, and Mohamed Mokbel. 2008. SARD: A statistical approach for ranking database tuning parameters. *Proceedings - International Conference on Data Engineering*, 11–18. https://doi.org/10.1109/ICDEW.2008.4498279

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 4171–4186.

[11] Theresa Eimer, Marius Lindauer, and Roberta Raileanu. 2023. Hyperparameters in Reinforcement Learning and How To Tune Them. In *Proceedings of the 40th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.), Vol. 202. PMLR, 9104–9149. https://proceedings.mlr.press/v202/eimer23a.html

[12] Electrum. [n.d.]. tpch-dbgen. GitHub repository. https://github.com/electrum/tpch-dbgen Accessed: 2025-05-30.

[13] Milton Friedman. 1937. The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance. *J. Amer. Statist. Assoc.* 32, 200 (1937), 675–701. https://doi.org/10.1080/01621459.1937.10503522 arXiv:https://www.tandfonline.com/doi/pdf/10.1080/01621459.1937.10503522

[14] Odd Erik Gundersen and Sigbjørn Kjensmo. 2018. State of the art: Reproducibility in artificial intelligence. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[15] Yaniv Gur, Dongsheng Yang, Frederik Stalschus, and Berthold Reinwald. 2021. Adaptive Multi-Model Reinforcement Learning for Online Database Tuning.. In *EDBT*. 439–444.

[16] Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double Q-Learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence* (Phoenix, Arizona) *(AAAI'16)*. AAAI Press, 2094–2100.

[17] Head, Tim and Kumar, Manoj and Nahrstaedt, Holger and Louppe, Gilles and Shcherbatyi, Iaroslav. 2023. Scikit-Optimize. https://scikit-optimize.github.io/stable/. Accessed: 2024-03-25.

[18] HustAIsGroup. [n.d.]. CDBTune: An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning. GitHub repository. https://github.com/HustAIsGroup/CDBTune Accessed: 2025-06-20.

[19] Younggyun Koh, Rob Knauerhase, Paul Brett, Mic Bowman, Zhihua Wen, and Calton Pu. 2007. An Analysis of Performance Interference Effects in Virtual Environments. In *2007 IEEE International Symposium on Performance Analysis of Systems & Software*. 200–209. https://doi.org/10.1109/ISPASS.2007.363750

[20] Alexey Kopytov. 2024. Sysbench: A System Performance Benchmark. https://github.com/akopytov/sysbench Accessed: 2024-07-15.

[21] Eugenie Y. Lai et al. 2023. Workload-Aware Deep Learning for SQL Query Recommendation. In *EDBT*.

[22] Jiale Lao, Yibo Wang, Yufei Li, Jianping Wang, Yunjia Zhang, Zhiyuan Cheng, Wanghu Chen, Mingjie Tang, and Jianguo Wang. 2024. GPTuner: A Manual-Reading Database Tuning System via GPT-Guided Bayesian Optimization. *Proceedings of the VLDB Endowment* 17, 8 (April 2024), 1939–1952. https://doi.org/10.14778/3659437.3659449

[23] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[24] Mohammad Masum, Hossain Shahriar, Hisham Haddad, Md Jobair Hossain Faruk, Maria Valero, Md Abdullah Khan, Mohammad A Rahman, Muhaiminul I Adnan, Alfredo Cuzzocrea, and Fan Wu. 2021. Bayesian hyperparameter optimization for deep neural network-based network intrusion detection. In *2021 IEEE International Conference on Big Data (Big Data)*. IEEE, 5413–5419.

[25] Adi Zaenul Mustaqim, Sumarni Adi, Yoga Pristyanto, and Yuli Astuti. 2021. The effect of recursive feature elimination with cross-validation (RFECV) feature selection algorithm toward classifier performance on credit card fraud detection. In *2021 International conference on artificial intelligence and computer science technology (ICAICST)*. IEEE, 270–275.

[26] OpenAI. 2023. ChatGPT: "A Large Language Model". https://chat.openai.com/. Accessed: 2025-06-25.

[27] Zakaria Ournani, Mohammed Chakib Belgaid, Romain Rouvoy, Pierre Rust, Joel Penhoat, and Lionel Seinturier. 2020. Taming Energy Consumption Variations In Systems Benchmarking. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering* (Edmonton AB, Canada) *(ICPE '20)*. Association for Computing Machinery, New York, NY, USA, 36–47. https://doi.org/10.1145/3358960.3379142

[28] Debjyoti Paul, Jie Cao, Feifei Li, and Vivek Srikumar. 2022. Database Workload Characterization with Query Plan Encoders. In *PVLDB*.

[29] P Peduzzi, J Concato, E Kemper, T Holford, and AR Feinstein. 1996. A simulation study of the number of events per variable in logistic regression analysis. *Journal of Clinical Epidemiology* 49, 12 (1996), 1373–1379.

[30] Percona-Lab. [n.d.]. tpcc-mysql. GitHub repository. https://github.com/Percona-Lab/tpcc-mysql Accessed: 2025-05-30.

[31] Edward O Pyzer-Knapp. 2018. Bayesian optimization for accelerated drug discovery. *IBM Journal of Research and Development* 62, 6 (2018), 2–1.

[32] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philip H. S. Torr, Jakob Foerster, and Shimon Whiteson. 2019. The StarCraft Multi-Agent Challenge. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems* (Montreal QC, Canada) *(AAMAS '19)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2186–2188.

[33] Orange Business Services. 2025. Flexible Engine. https://www.orange-business.com/en/products/business-vpn-galerie/cloud-services/flexible-engine. Accessed: 2025-03-6.

[34] Dennis Shasha, Philippe Bonnet, and Nancy Hartline Bercich. 2004. Database tuning principles, experiments, and troubleshooting techniques. *SIGMOD Rec.* 33, 2 (jun 2004), 115–116. https://doi.org/10.1145/1024694.1024720

[35] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian optimization of machine learning algorithms. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2* (Lake Tahoe, Nevada) *(NIPS'12)*. Curran Associates Inc., Red Hook, NY, USA, 2951–2959.

[36] Statista. 2024. Worldwide Data Created. https://www.statista.com/statistics/871513/worldwide-data-created/. Accessed: 12 March 2025.

[37] Vamsidhar Thummala and Shivnath Babu. 2010. iTuned: a tool for configuring and visualizing database parameters. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data* (Indianapolis, Indiana, USA) *(SIGMOD '10)*. Association for Computing Machinery, New York, NY, USA, 1231–1234. https://doi.org/10.1145/1807167.1807327

[38] Robert Tibshirani. 1996. Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)* 58, 1 (1996), 267–288. http://www.jstor.org/stable/2346178

[39] Immanuel Trummer. [n.d.]. dbbert: DB-BERT Database Tuning Tool Implementation. GitHub repository. https://github.com/itrummer/dbbert Accessed: 2025-06-20.

[40] Immanuel Trummer. 2022. DB-BERT: A Database Tuning Tool that "Reads the Manual". In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 190–203. https://doi.org/10.1145/3514221.3517843

[41] Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. 2017. Automatic Database Management System Tuning Through Large-Scale Machine Learning. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 1009–1024. https://doi.org/10.1145/3035918.3064029

[42] Dana Van Aken, Dongsheng Yang, Sebastien Brillard, Ari Fiorino, Bohan Zhang, Christian Bilien, and Andrew Pavlo. 2021. An inquiry into machine learning-based automatic configuration tuning services on real-world database management systems. *Proc. VLDB Endow.* 14, 7 (March 2021), 1241–1253. https://doi.org/10.14778/3450980.3450992

[43] Quang H Vuong. 1989. Likelihood ratio tests for model selection and non-nested hypotheses. *Econometrica: journal of the Econometric Society* (1989), 307–333.

[44] Junxiong Wang, Immanuel Trummer, and Debabrota Basu. 2021. UDO: universal database optimization using reinforcement learning. *Proceedings of the VLDB Endowment* 14 (09 2021), 3402–3414. https://doi.org/10.14778/3484224.3484236

[45] Yifan Wang, Pierre Bourhis, Romain Rouvoy, and Patrick Royer. 2024. Challenges & Opportunities in Automating DBMS: A Qualitative Study. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering* (Sacramento, CA, USA) *(ASE '24)*. Association for Computing Machinery, New York, NY, USA, 2013–2023. https://doi.org/10.1145/3691620.3695264

[46] B. L. Welch. 1947. The Generalization of 'Student's' Problem When Several Different Population Variances Are Involved. *Biometrika* 34, 1/2 (1947), 28–35. https://doi.org/10.2307/2332510

[47] Tong Yu and Hong Zhu. 2020. Hyper-Parameter Optimization: A Review of Algorithms and Applications. *CoRR* abs/2003.05689 (2020). arXiv:2003.05689 https://arxiv.org/abs/2003.05689

[48] Ji Zhang, Ke Zhou, Guoliang Li, Yu Liu, Ming Xie, Bin Cheng, and Jiashu Xing. 2021. CDBTune: An efficient deep reinforcement learning-based automatic cloud database tuning system. *The VLDB Journal* 30 (11 2021). https://doi.org/10.1007/s00778-021-00670-9

[49] Xinyi Zhang, Zhuo Chang, Yang Li, Hong Wu, Jian Tan, Feifei Li, and Bin Cui. 2022. Facilitating database tuning with hyper-parameter optimization: a comprehensive experimental evaluation. *Proc. VLDB Endow.* 15, 9 (May 2022), 1808–1821. https://doi.org/10.14778/3538598.3538604