# Towards Adaptive and Practical Automation in DBMS Administration

*Practitioner Insights, Dynamic Resource Optimization, and Lightweight Tuning*

**Yifan WANG**




MADIS Doctoral School
Inria Center at the University of Lille
CRIStAL — Spirals research team

Thesis presented and defended on 12 November 2025
to obtain the degree of Doctor of Computer Science

***Supervisor and Co-supervisor***

**Romain Rouvoy**
*Full Professor, University of Lille*

**Pierre Bourhis**
*Senior Researcher, CNRS*

**Patrick Royer**
*Engineer, Orange Innovation*

**Fabrice Chaillou**
*Manager, Orange Innovation*

***Reviewers***

**Jalil Boukhobza**
*Full Professor, ENSTA Bretagne*

**Ioana Manolescu**
*Senior researcher, INRIA Saclay*

***Examiners***

**Anthony Cleve**
*Full Professor, University of Namur*
*Chair of the Thesis Jury*

**Debabrota Basu**
*Research Fellow, INRIA*

# Vers une automatisation adaptative et pratique de l'administration des SGBD

*Retours d'expérience, optimisation dynamique des ressources et réglage léger*

**Yifan WANG**




École doctorale MADIS
Centre Inria de l'Université de Lille
Équipe de recherche CRIStAL — Spirals

Thèse présentée et soutenue publiquement le 12 novembre 2025
pour l'obtention du grade de Docteur en Informatique

***Directeurs et co-encadrants de thèse***

**Romain Rouvoy**
*Professeur, Université de Lille*

**Pierre Bourhis**
*Directeur de recherche, CNRS*

**Patrick Royer**
*Ingénieur, Orange Innovation*

**Fabrice Chaillou**
*Manager, Orange Innovation*

***Rapporteurs***

**Jalil Boukhobza**
*Professeur, ENSTA Bretagne*

**Ioana Manolescu**
*Directrice de recherche, Inria Saclay*

***Examinateurs***

**Anthony Cleve**
*Professeur, Université de Namur*
*Président du jury de thèse*

**Debabrota Basu**
*Chargé de recherche, Inria*

# Résumé

Les systèmes de gestion de bases de données sont essentiels pour une gestion efficace des données et un contrôle des accès. Toutefois, en raison de leur complexité, leur administration demeure une tâche difficile pour les administrateurs de bases de données. Cette thèse s'intéresse à l'automatisation de l'administration des SGBD à travers trois contributions complémentaires, visant une meilleure adaptabilité et une applicabilité concrète.

Tout d'abord, nous présentons une étude qualitative des pratiques des administrateurs de bases de données, qui met en évidence un ensemble de difficultés : sur-allocation de ressources, procédures de réglage sous-optimales et faible adoption des outils d'automatisation existants. À partir de ces constats, nous nous concentrons sur deux problèmes majeurs : la sur-provision de mémoire et le besoin d'outils d'auto-tuning des SGBD plus pratiques.

Dans cette optique, nous proposons DOT, un cadre léger de sélection et d'optimisation des paramètres de configuration. DOT combine l'élimination récursive de variables avec validation croisée et l'optimisation bayésienne afin d'identifier rapidement les paramètres les plus influents. En réduisant l'espace de recherche sans nécessiter de longues phases de mise en chauffe, DOT permet un réglage des bases de données plus efficace et économique que les autotuners traditionnels.

Nous présentons ensuite MicroTune, un gestionnaire adaptatif de mémoire tampon piloté par apprentissage par renforcement, qui ajuste en continu l'allocation mémoire en fonction de la charge. En adaptant dynamiquement la taille des tampons aux besoins de la charge de travail, MicroTune réduit le gaspillage tout en respectant les contraintes de SLA, offrant ainsi une solution concrète d'optimisation des ressources en production.

En combinant des enseignements issus de la pratique avec des outils adaptatifs conçus sur mesure, cette thèse propose des avancées plus adaptatives et pragmatiques pour l'automatisation de l'administration des SGBD.

# Abstract

*Database Management Systems* (DBMSs) are crucial for efficient data management and access control, but due to the complexity of DBMSs, their administration remains challenging for Database Administrators. This thesis investigates DBMS administration automation through three complementary contributions for better adaptability and real-world applicability.

First, we introduce a qualitative study of *Database Administrator* (DBA) workflows that surfaces persistent issues: resource over-allocation, sub-optimal tuning routines, and limited uptake of automation tools. From these findings, we concentrate on two pressing pain points: the need for more practical DBMS auto-tuning and excessive memory provisioning.

Building on this, we propose DOT, a lightweight knob selection and optimization framework. DOT combines recursive feature elimination with cross-validation and Bayesian optimization to quickly identify the most impactful configuration knobs. By narrowing the search space without lengthy warm-up phases, DOT enables more efficient and cost-effective DBMS tuning compared to traditional auto-tuners.

To address inefficient memory use, we introduce MicroTune, an adaptive reinforcement learning driven buffer manager that continuously adjusts memory allocation in real time. By rightsizing buffers to match workload demands, MicroTune cuts waste and meets SLA requirements, showcasing a hands-on approach to resource optimization in production.

# Acknowledgments

I want to begin by expressing my deepest gratitude to my supervisors, Patrick, Pierre, and Romain, whose unwavering guidance and support throughout these three years have been invaluable. I am also grateful to the entire DBO team at Orange—under the leadership of Fabrice—for their warm welcome and constant encouragement. Your trust and the freedom you granted me allowed me to explore new avenues and fully immerse myself in the academic world.

My thanks also go to the members of the examination committee for taking the time to evaluate this work. Your insightful feedback and constructive advice have significantly enhanced both its depth and clarity, and I am truly appreciative.

I owe a special debt of thanks to the SPIRALS team. Although my visits to Lille were infrequent, each one was an opportunity to learn from passionate and inspiring colleagues—and, of course, to share a few beers.

I am deeply grateful to my loved ones for their continual support over the course of my doctoral studies:

- My parents and grandparents
- My friends in Hangzhou, Shanghai, Lille, Paris, and beyond
- My family and in-laws
- Those dear to me who are no longer with us

Finally, to Yuyuan: no words suffice to express the joy and love you bring into my life.

# Contents

# List of Figures

# List of Tables

# List of Acronyms

| | |
|---|---|
| **RL** | Reinforcement Learning |
| **SLA** | Service Level Agreement |
| **TPS** | Transactions Processed per Second |
| **QPS** | Queries Processed per Second |
| **DBMS** | Database Management System |
| **DBA** | Database Administrator |
| **VM** | Virtual Machine |
| **ML** | Machine Learning |
| **DNN** | Deep Neural Network |
| **LLM** | Large Language Model |
| **PCA** | Principal Component Analysis |
| **DDPG** | Deep Deterministic Policy Gradients |
| **DRL** | Deep Reinforcement Learning |
| **TCO** | Total Cost of Ownership |
| **PPO** | Proximal Policy Optimization |
| **RFECV** | Recursive Feature Elimination with Cross-Validation |
| **LRT** | Likelihood Ratio Test |
| **BO** | Bayesian Optimization |

# Chapter 1

# Introduction

## 1.1 Context

Enterprise data continues to grow at an unprecedented pace, projected to exceed approximately 180 zettabytes by 2025. This surge in both volume and heterogeneity places extraordinary pressure on DBMS [1], which must now operate across multi-node clusters, containerized microservices, and hybrid cloud infrastructures. Each of these environments requires careful resource allocation to strike a balance between throughput, latency, and cost.

Managing such systems has turned DBMS administration into a highly specialized and labor-intensive role. DBAs must not only perform upgrades, backups, and anomaly detection, but also invest significant effort in performance tuning and capacity planning. These activities alone can consume nearly half of a DBA's working time, and in many organizations, the personnel costs of skilled DBAs exceed both software licensing fees and hardware investments [2, 3]. This imbalance underscores the need for automation: intelligent tools that can relieve routine burdens, adapt to shifting workloads, and ultimately reduce the *Total Cost of Ownership* (TCO) of large-scale DBMS deployments.

Over the last decade, both academia and industry have made important strides towards such automation. Fully autonomous systems, such as Oracle Autonomous Database [4], NoisePage [5, 6], and CrimsonDB [7], aim to manage the entire DBMS life-cycle with minimal human intervention. Meanwhile, commercial engines such as IBM Db2 and Microsoft SQL Server have introduced intelligent advisors, like Db2's autonomic workload management [8, 9] and SQL Server's Query Store and Automatic Tuning [10, 11]. On the research side, prototypes such as UDO [12], which uses cost-based descriptor tuning, and CDBTune [13], which applies reinforcement learning to configuration settings, demonstrate the promise of fine-grained adaptation.

Yet, despite their sophistication, these solutions have seen limited adoption in production. To understand why, we conducted a qualitative study at Orange, a leading European telecommunications operator [14]. Through interviews and observation, we found that practitioners often lack time to integrate new automation, face difficulties reproducing research prototypes, and end up over-allocating resources to maintain acceptable performance. These realities highlight the gap between ambitious proposals and practical needs: what is required are adaptive, lightweight approaches that DBAs can realistically use day to day.

## 1.2 Problem Statement

Two challenges emerge as especially pressing: the lack of a practical DBMS auto-tuning, and the widespread habit of excessive memory provisioning.

Modern DBMSs expose hundreds of configuration parameters—often referred to as *knobs*. These knobs determine throughput, latency, resource utilization, and ultimately, the total cost of ownership. Because many knobs interact in nonlinear, workload-dependent ways, manual tuning is slow, error-prone, and rarely sustainable. Misconfigurations can degrade throughput, while the fear of SLA violations drives DBAs to allocate excessive CPU and RAM, inflating operational expenses.

The essence of the tuning problem is to select an effective combination of knob settings that balances latency, throughput, and cost while respecting service-level constraints. The search space grows exponentially with the number of knobs, and each benchmark run is costly. This makes naive exploration infeasible. Complicating matters further, workloads do not stay fixed: traffic surges, evolving query mixes, and shifting data distributions quickly render yesterday's optimal configuration ineffective. Without continuous monitoring and adaptive re-tuning, DBAs are left with two bad options: constant manual retesting or blanket over-provisioning.

Bridging this gap calls for practical auto-tuning solutions: methods that can efficiently explore large parameter spaces, adapt dynamically as workloads evolve, and do so without imposing prohibitive runtime overhead. In this dissertation, I pursue exactly this goal.

First, I introduce DOT, a lightweight framework for dynamic knob selection and online sampling, which converges rapidly to effective configurations with minimal overhead. Second, I present MicroTune, an adaptive memory allocator that uses reinforcement learning to continuously adjust buffer sizes in response to workload fluctuations, reducing over-provisioning while maintaining SLA compliance. Importantly, both contributions operate externally to the DBMS kernel. This non-intrusive design reflects industrial constraints, where kernel changes are rarely possible, and ensures portability across different engines, even if it limits us to the interfaces the DBMS provides.

Together, these contributions aim to bring DBMS administration automation closer to everyday practice—practical, efficient, and deployable in real-world systems.

## 1.3 Contributions

This thesis is organized into three parts, each addressing a key aspect of automating DBMS administration. In addition, **Chapter 2** discusses the current status of DBMS administration automation by surveying existing approaches, from fully automated self-driving DBMS solutions to specialized autotuners for DBMS configuration, and outlines open challenges that motivate the rest of this thesis. Chapter 3 then presents a qualitative study based on grounded-theory interviews that identifies practical challenges and opportunities in DBMS automation."

1. **Challenges and Opportunities (Chapter 3):** We present findings from an in-depth qualitative study involving interviews with seasoned DBMS experts at Orange. Applying grounded theory, we distill insights into the current pain points and potential automation opportunities in DBMS administration. These results inform recommendations for DBAs, DBMS users, tool developers, and researchers.

2. **Lightweight Knob Selection for Performance Tuning (Chapter 4):** We describe DOT, a lightweight tuning framework that combines recursive feature elimination with cross-validation and Bayesian optimization. DOT efficiently identifies the most impactful DBMS configuration knobs without lengthy warm-up phases, drastically cutting computational overhead compared to traditional tuners.

3. **Dynamic Resource Optimization via Reinforcement Learning (Chapter 5):** We introduce MicroTune, an adaptive buffer-management system based on rein-

forcement learning. MicroTune dynamically reallocates RAM in real time, reducing resource over-provisioning while respecting SLA constraints, thereby minimizing operational costs.

In addition to addressing the research goal, tools and data generated throughout this dissertation have been made publicly available as open-source or open-data in repositories: https://github.com/Orange-OpenSource/microtune and https://github.com/Orange-OpenSource/dot. This encourages the reproducibility of our results and facilitates future research in the field.

## 1.4 List of Scientific Publications

Parts of this thesis are adapted from the following publication:

1. Y. Wang, P. Bourhis, R. Rouvoy, P. Royer. *Challenges and Opportunities in Automating DBMS: A Qualitative Study*, ASE'24: Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, Pages 2013–2023, https://doi.org/10.1145/3691620.3695264

2. Y. Wang, D. Basu, P. Bourhis, R. Rouvoy, P. Royer. *DOT: Dynamic Knob Selection and Online Sampling.* Accepted to VLDB'26: International Conference on Very Large Databases

Two works are under submission:

1. Y. Wang, P. Royer, R. Féraud, D. Delande. *MicroTune: RL-based DBMS Buffer Pool Auto-Tuning for Optimal and Economical Memory Utilization.*

# Chapter 2

# State of the Art

This chapter surveys the state of the art in automating DBMS administration, with a particular focus on auto-tuning. We begin by situating auto-tuning within the broader context of DBMS administration automation, highlighting its role as a central and particularly challenging task. We then narrow our attention to auto-tuning itself, reviewing the main techniques that underpin modern tuners—ranging from knob selection and transfer learning to real-time adaptability and evaluation practices. Finally, we discuss the limitations and adoption barriers that prevent many of these methods from reaching widespread deployment in practice. This review not only provides a technical overview of the field but also outlines the open challenges that motivate the contributions of this thesis.

## 2.1 DBMS Administration Automation

Operating a modern DBMS entails handling vast amounts of data—but administering one remains a complex, labor-intensive endeavor. Database administrators (DBAs) must handle a broad spectrum of operational tasks, including:

1. **Configuration Tuning:** Right-size memory caches, I/O queues, and parallelism settings to match both workload patterns and hardware characteristics, continually revisiting parameters to sustain peak performance and avoid bottlenecks.

2. **Physical Design:** Craft an optimal index strategy to accelerate queries; partition or shard large tables; and balance I/O by strategically placing tablespaces and data files.

3. **Patching & Upgrading:** Orchestrate maintenance windows, apply security fixes and engine updates, then rigorously test in staging. Validate application compatibility and maintain rollback plans in case issues arise.

4. **Backup & Recovery:** Design full, differential, and incremental backup regimes that meet strict Recovery Point Objective and Recovery Time Objective targets; regularly practice restores and document workflows to guarantee rapid, reliable recovery.

5. **Capacity Planning:** Monitor CPU, memory, storage, and network usage to project future resource needs; collaborate with infrastructure teams to provision hardware, cloud instances, or additional storage before budgets and demands collide.

Beyond these core activities—tuning, design, forecasting, patching, backup, and planning—DBAs also manage security, real-time monitoring, and incident response, plus change management and documentation. These human-centric areas remain especially

resistant to full automation, as they must absorb application-specific nuances and organizational policies.

Both industry and academia have therefore invested heavily in automating routine DBA tasks. From self-driving platforms and integrated advisors to specialized auto-tuners and anomaly detectors, a rich ecosystem of projects now targets everything from end-to-end DBMS autonomy to fine-grained tuning assistants. Many research and industry initiatives have pursued the goal of a fully autonomous DBMS—one that, once deployed, requires no manual intervention. While this ambitious vision remains challenging, many projects have made significant strides by integrating AI/ML components into core database services. The Peloton project, introduced by Pavlo et al. [15], was among the first open-source efforts to realize a self-driving architecture. Built as an in-memory SQL engine by the CMU Database Group [16], Peloton embedded an AI-driven planner directly within the database kernel. This planner continuously monitors live workloads, forecasts future demands, and dynamically adjusts physical design and tuning parameters—thereby reducing the need for traditional, post-hoc advisory tools.

Although Peloton was discontinued in 2018, its successor NoisePage emerged in 2018 as a complete redesign targeted at full autonomy [17]. NoisePage replaces Peloton's monolithic planner with a suite of specialized ML agents. Each agent focuses on a distinct subsystem—such as query latency, buffer pool utilization, or index access patterns—and applies tuning actions (knob adjustments, index creation or dropping, data layout reorganization) at runtime without necessitating downtime. This modular, agent-based approach yields faster convergence to optimal configurations and greater adaptability to workload fluctuations.

Other projects have followed a similar pattern of embedding AI within narrowly scoped subsystems. For example, CrimsonDB [18], developed at Stratos Idreos's lab, automatically designs and optimizes its physical data structures—indexes and layouts—in response to evolving workloads and hardware changes. Despite its early promise, CrimsonDB also appears inactive after several years of development. Together, these projects illustrate both the potential and the practical hurdles of realizing truly autonomous database administration. In the next section, we delve into recent advances in specialized automation for DBMS tuning.

On the industrial front, cloud providers have pushed autonomy even further by embedding end-to-end automation into their managed database services. Three flagship offerings illustrate this shift:

- **Oracle Autonomous Database** leverages machine-learned workload classification and adaptive indexing to tune itself for OLTP or data-warehouse workloads. Automated patching, backup, and scaling are carried out transparently, with minimal or no downtime [4].

- **Azure SQL Database** continuously collects indexed telemetry to drive built-in performance recommendations and automatic tuning. Geo-replicated backups and brief, automated failovers; automated patching reduces exposure windows while maintaining availability [19].

- **Amazon Aurora**'s high availability and rapid recovery [20]. It provides a distributed, self-healing storage layer with continuous crash recovery and automatic failover. Amazon DevOps Guru [21] for RDS uses machine learning on Performance Insights metrics to detect emerging performance anomalies and surface actionable findings.

Meanwhile, traditional on-premise systems have evolved their "advisor" toolkits into deeply integrated automation modules:

| | Auto Tuning | Auto Physical design | Workload Forecast | Auto Patching/updates | Auto Backup-Recovery | Capacity Planning | Data Exploitation | Open source |
|---|---|---|---|---|---|---|---|---|
| Peloton | X | | X | | | | | X |
| NoisePage | X | X | X | | | | | X |
| CrimsonDB | | X | | | | X | | |
| Oracle Autonomous DB | X | X | | X | X | X | X | |
| Microsoft SQL Server | X | X | | X | X | X | X | |
| AWS Aurora | | | | X | X | X | X | |
| IBM DB2 | X | X | | | X | X | X | |

Table 2.1: DBMS administration automation projects

- **IBM Db2 Design Advisor** uses optimizer cost models and hybrid search over compressed workload traces to recommend indexes, materialized query tables, partitioning schemes, and clustering strategies [22].
- **Db2 Self-Tuning Memory Manager** reallocates cache memory dynamically among buffer pools to maximize hit rates without manual intervention [23].
- **Microsoft SQL Server Query Store** tracks plan drift and, together with Automatic Tuning, enforces optimal execution plans and suggests missing indexes, while Intelligent Insights issues proactive anomaly alerts [10, 11].

Table 2.1 summarizes the automated capabilities delivered by these leading vendors alongside academic prototypes. Academic research has focused primarily on self-tuning and automated physical design, with CMU's Peloton and NoisePage projects adding a dedicated workload forecasting component for deeper autonomy. In contrast, cloud-native platforms extend beyond tuning and design to include seamless patching, backup/recovery, failover, and capacity planning. These supplementary functions exploit cloud features such as rolling updates for minimal-impact online patches, elastic storage for rapid backups, and auto-scaling to match resource allocation to workload demand.

Finally, many industrial offerings now bundle advanced data exploitation, tools, integrated analytics dashboards, visualization platforms, and even LLM-powered "copilots"—to help DBAs and developers derive insights directly from their managed databases. However, despite their varying degrees of autonomy, these platforms still require periodic expert oversight. For instance, Oracle Autonomous Database reports "up to 68% more efficient than DBA teams and up to 63% lower total TCO [4], but it does not eliminate the need for human intervention.

Moreover, even though industrial offerings deliver end-to-end automation and high availability, their licensing and operational costs can be prohibitive. Hosting an Oracle Autonomous Database could be significantly more expensive than running Oracle Database Enterprise Edition Extreme Performance on comparable infrastructure [24]. While managed services from AWS, Microsoft Azure and Oracle streamline provisioning, patching and scaling, the total cost of ownership often escalates quickly. As a result, many enterprises—including Orange—opt to deploy open-source DBMS on-premises or in private clouds, retaining full control over configuration, scaling and budget.

The systems surveyed in this section illustrate that many facets of DBMS administration can be automated to varying degrees, from provisioning and backup to patching, failover, and capacity planning. However, among all these tasks, configuration and resource tuning play a disproportionately important role in practice: they directly determine query performance, hardware utilization, and ultimately the total cost of ownership (TCO) of the database stack. Covering the full breadth of administration automation is beyond the scope of this thesis. In the remainder of this chapter, we therefore narrow our focus to *DBMS knob tuning* as the central lever through which automated systems influence performance and resource consumption.

## 2.2 Core Concepts and Auto-Tuning Problems

In this section, we turn our attention to the pivotal challenge of automatic configuration tuning. Modern DBMSs present a bewildering array of configuration knobs whose optimal settings depend on workload dynamics, hardware characteristics, and complex, nonlinear interactions.

Administering a modern DBMS involves controlling numerous subsystems—memory, I/O, concurrency, transaction management, logging, physical design, and monitoring.

Table 2.2: Comparison of Auto-tuning Algorithms

| **Tuner** | **Knob Sel.** | **Resource Tune** | **Perf. Tune** | **Idx Tune** | **Pretrain** | **Hist. Data** | **3rd-Party API** | **Core Algo.** |
|---|---|---|---|---|---|---|---|---|
| PGTune | Expert-defined | – | Hardware-based recommendations | – | – | – | – | Heuristic |
| iTuned | Sensitivity analysis | – | Gradient-based sampling | – | – | – | – | Greedy search + Sensitivity ranking |
| iBTune | Expert-defined | Leverage stats to reduce mem. | – | – | – | Use hist. stats for mem. | – | Heuristic |
| BestConfig | User-defined | – | Search-based sampling | – | – | – | – | Divide-&-Diverge |
| OtterTune | Lasso feature selection | – | Bayesian opt. with large-scale ML | – | Large pretrain dataset req. | Integrates hist. tuning data | – | Bayesian opt. |
| ResTune | Expert-defined | Constrained BO for I/O, CPU & RAM | Constrained BO recommendations | – | Heavy warm-up req. | Transfer learning accelerated | – | Constrained BO |
| UDO | User-defined | – | Hierarchical Opt. (HOO) | HOO for index tuning | – | – | – | Delayed HOO |
| CDBTune | Expert-ranked | – | DDPG for config. search | – | High-qual. warm-up samples req. | Needs hist. data for warm-up | – | DDPG |
| Hunter | CART feature selection | – | GA warm-up + DDPG fine-tune | – | GA sampling & CART sel. req. | Uses hist. data for warm-up | – | GA + DDPG |
| DB-BERT | LLM parsing | – | BERT parse + Double DQN | – | Doc. collection req. | – | Local BERT/LLM API req. | Double DQN |
| GPTuner | LLM parsing | – | ChatGPT parse + SMAC fine-tune | – | Doc. collection req. | – | LLM API req. | Coarse-to-Fine BO |

Although DBMSs provide extensive functionality, effectively operating them remains a complex and error-prone task requiring expert knowledge of system internals and workload behavior. This section introduces the core concepts that underpin DBMS auto-tuning and prepares the ground for the subsequent sections.

### Knobs and Configuration Parameters

A central concept in DBMS automation is the *configuration knob*: a tunable parameter that influences how the DBMS allocates resources or processes queries. Modern engines expose hundreds of knobs across memory, I/O, concurrency, metadata caching, and background task subsystems. For example, InnoDB provides:

- **Memory and buffer pool knobs:** `innodb_buffer_pool_size`, `innodb_buffer_pool_instances`, `tmp_table_size`.
- **I/O knobs:** `innodb_read_io_threads`, `innodb_write_io_threads`, `innodb_io_capacity`, `innodb_read_ahead_threshold`, `innodb_flush_neighbors`.
- **Concurrency knobs:** `innodb_thread_concurrency`, `thread_cache_size`, `innodb_spin_wait_delay`, `innodb_sync_spin_loops`.
- **Background maintenance knobs:** `innodb_purge_threads`, `innodb_page_cleaners`, `innodb_lru_scan_depth`.
- **Metadata and hash index knobs:** `table_open_cache`, `table_open_cache_instances`, `innodb_adaptive_hash_index`, `innodb_adaptive_hash_index_parts`.

These knobs interact in complex, non-linear ways. Increasing the buffer pool size may require retuning flushing knobs; enabling the adaptive hash index may help OLTP workloads but increase contention; adjusting thread or I/O parallelism can introduce subtle trade-offs. Thus, DBMS configuration tuning naturally forms a high-dimensional, workload-dependent optimization problem.

## Auto-Tuning Problems

Automated DBMS tuning encompasses several classes of optimization problems that differ in scope, objectives, and runtime constraints. Before discussing tuning methods, it is essential to distinguish the major categories of auto-tuning problems encountered in the literature: (1) intra-DBMS vs. outside-DBMS tuning, and (2) online vs. offline tuning. These distinctions clarify the assumptions made by different tuners and the contexts in which they operate.

### Intra-DBMS vs. Outside-of-DBMS Tuning

**Intra-DBMS tuning** refers to problems where the parameters being optimized belong to the DBMS itself. Typical examples include buffer pool sizes, I/O thread counts, lock manager settings, concurrency parameters, and background task configuration. The configuration vector $x \in \mathcal{X}$ directly corresponds to DBMS-exposed knobs such as those described earlier.

**Outside-DBMS tuning** expands the optimization scope beyond internal DBMS knobs, targeting layers of the software and hardware stack that affect DBMS performance indirectly. This includes:

- OS-level parameters (e.g., huge pages, I/O scheduler mode),

- VM or container parameters (e.g., CPU pinning, memory limits),
- storage configuration (e.g., RAID layout, filesystem options),
- hardware provisioning decisions (e.g., number of vCPUs, SSD tier).

In such settings, DBMS performance becomes a function of both internal knobs and external resource configurations.

#### Online vs. Offline Tuning

**Offline tuning** assumes that configuration adjustments are performed *between* workload executions. The system is tuned on a representative training workload, and the resulting configuration is deployed statically. This setting includes most classical advisors and search-based tools.

**Online tuning** assumes that configuration adjustments occur *during* workload execution, with the tuner continuously observing metrics and applying changes dynamically. Systems such as MicroTune operate in this regime, adjusting configurations while queries are running and without requiring downtime. This introduces additional constraints related to stability, convergence, and overhead, since tuning actions must not disrupt live traffic.

#### Formal Problem Definition

Across these categories, the core optimization problem can be expressed as follows. Let $x \in \mathcal{X}$ denote a DBMS configuration (internal knobs, external resource settings, or both), where $\mathcal{X}$ is the product of all admissible parameter domains. Given a fixed workload and deployment environment, the goal is to find a configuration $\hat{x}$ that maximizes a performance function $f(x)$—such as throughput, tail latency, or a composite utility metric—subject to runtime and resource constraints:

$$\hat{x} \;=\; \arg\max_{x \in \mathcal{X}} f(x) \quad \text{s.t.} \quad \text{elapsed time} \leq T_{\max}, \;\; x \in \mathcal{X}_{\text{resource}}.$$

Here $\mathcal{X}_{\text{resource}} \subseteq \mathcal{X}$ captures feasible configurations under memory, CPU, storage, or stability constraints. Online tuners may additionally impose constraints on the magnitude or frequency of configuration changes to maintain system stability.

When left untuned, DBMSs often suffer from suboptimal throughput, excessive tail latency, or resource overprovisioning. Manual tuning over a high-dimensional, mixed discrete–continuous configuration space is time-consuming, error-prone, and rarely repeatable. Automated tuning systems therefore aim to systematically navigate $\mathcal{X}$, adapt to workload dynamics, and apply changes safely—whether offline through batch optimization or online through continuous feedback-driven adjustment.

## 2.3 DBMS Knob Tuning Methods

Building on the concepts and problem formulation in Section 2.2, we now survey the main families of methods proposed to solve DBMS auto-tuning problems. Table 2.2 summarizes the principal differences across the main auto-tuners grouped by their dominant optimization or learning paradigm: heuristic/rule-based, search-based, Bayesian optimization, reinforcement learning, and LLM-based methods. Before turning to these families, we first discuss what these tuners actually learn and how they select which knobs to tune.

## Learning Targets and Knob Selection

Every auto-tuner must answer two fundamental questions: *what* to optimize and *what* to tune. The former concerns the optimization target (e.g., throughput, latency, tail percentiles, or composite utility), while the latter concerns the subset of configuration knobs that meaningfully affect these targets. Given that modern DBMSs expose hundreds of parameters, knob selection is a critical first step: it reduces the dimensionality of the search space, lowers sampling overhead, and accelerates convergence.

The literature on knob selection and search-space design can be broadly categorized into four groups of algorithms that tackle mainly intra-DBMS offline tuning:

- **Experience-based methods** rely on human expertise to define a fixed set of critical knobs. PGTune adopts static rule-based tuning guided by hardware specifications and best practices, while CDBTune [13] uses a predefined set of parameters during offline training. UDO [25] and BestConfig [26] similarly delegate the initial knob selection to DBAs, using domain intuition to distinguish between "light" and "heavy" knobs.

- **Screening-based techniques** apply statistical experiments or sensitivity analysis to quantify the influence of each knob. These methods typically require a small number of samples—often linear in the number of parameters—to identify high-impact knobs. Tools like iTuned [27] and SARD [28] exemplify this category through their use of design of experiments (e.g., Plackett–Burman designs) and gradient-based sensitivity ranking.

- **ML-based approaches** use data-driven models to automatically detect and prioritize impactful knobs. For instance, OtterTune [29] employs Lasso regression to prune low-impact parameters, while Hunter [30] uses decision-tree-based techniques (Random Forest with CART [31]) for feature-importance ranking. These methods adapt to workload characteristics and enable more targeted tuning.

- **LLM-based strategies** leverage large language models to interpret documentation, expert guides, or historical text and extract relevant tuning knowledge. DB-BERT [32] fine-tunes a BERT model [33] on database manuals to infer knob importance, whereas GPTuner [34] harnesses ChatGPT-style models to parse unstructured tuning advice and guide Bayesian optimization accordingly.

Each category involves trade-offs in complexity and resource demands. ML-based selection often requires hundreds of configuration–performance samples—a process that, for OLTP workloads, entails stress-testing for average TPS and, for OLAP workloads, demands sequential query runs—because statistical confidence in feature rankings grows with sample size. While Hunter reports that just two samples per knob may suffice [30], standard practice suggests collecting ten [35] to fifteen [36] samples per knob to ensure reliability.

Screening methods are far more lightweight—typically needing on the order of twice the number of knobs in experiments—but they capture only main effects, neglecting non-linear interactions. Experience-based selection avoids data-collection overhead but depends heavily on scarce expert time; even seasoned managers find tuning challenging [37]. LLM-based selection eliminates the need for workload traces but introduces its own costs: gathering and maintaining up-to-date documentation is laborious, and GPTuner reports spending over five hours and more than $50 in API fees solely on knob identification [34].

With this search space defined, different families of autotuners apply distinct optimization strategies, which we now review.

## Heuristic and Rule-Based Tuners

Heuristic and rule-based tuners encode expert knowledge or best practices into static or slowly evolving rules. They typically avoid expensive iterative search and provide quick, easy-to-apply recommendations but offer limited adaptability.

- **PGTune** is a heuristic-based tuning tool for PostgreSQL that applies a set of static, expert-defined rules to configure memory and planner parameters using coarse system information such as available RAM, number of cores, and storage type. It does not perform dynamic performance evaluation or search; instead, it relies on best-practice templates to set values for key knobs such as `shared_buffers`, `work_mem`, and `effective_cache_size`, providing quick guidance for non-expert users at the cost of workload-specific adaptivity.

- **iBTune** is an adaptive, rule-inspired tuning framework that dynamically adjusts the buffer pool size based on real-time workload observations [3]. It monitors metrics such as the LRU miss ratio and uses a $k$-nearest neighbors regression model to estimate a tolerable miss ratio under similar workloads. By inverting a power-law miss-ratio curve, iBTune derives an appropriate buffer pool size, applying changes online with SLA checks and automatic rollback to preserve stability.

## Search-Based Auto-Tuners

Search-based auto-tuners treat the DBMS as a black box and use general-purpose search algorithms (e.g., greedy local search, simulated annealing, genetic algorithms) to explore the configuration space.

- **iTuned** is an adaptive black-box tuner that optimizes configurations without prior workload knowledge [27]. It first performs a lightweight sensitivity analysis by perturbing knobs around an initial configuration to estimate their performance impact. It then focuses on influential knobs and applies a greedy local search to iteratively refine their values, achieving efficient tuning in high-dimensional spaces with limited sampling overhead.

- **BestConfig** is a general-purpose automatic configuration tuning framework designed to explore high-dimensional parameter spaces efficiently [26]. It combines *Divide-and-Diverge Sampling* (DDS), which ensures broad coverage via recursive partitioning and sampling, with *Recursive Bound-and-Search* (RBS), which refines search around promising configurations. BestConfig operates under a predefined tuning budget and remains agnostic to the underlying system, striking a balance between exploration and convergence.

## Bayesian-Optimization Auto-Tuners

Bayesian-optimization-based tuners build probabilistic surrogate models of the performance function and use acquisition functions to balance exploration and exploitation, making them well-suited for expensive, high-dimensional tuning tasks.

- **OtterTune** is a machine-learning-based configuration tuning system that leverages historical telemetry to model the relationship between knob settings and performance [29]. For a new workload, OtterTune identifies similar workloads in its repository, then applies Bayesian optimization to efficiently explore the configuration space and recommend knob adjustments. By reusing prior knowledge, it

reduces tuning time and sampling overhead while supporting diverse DBMSs and workloads.

- **ResTune** is a resource-aware auto-tuning framework that jointly optimizes CPU, memory, and I/O knobs under explicit SLA constraints [38]. It formulates tuning as a constrained Bayesian optimization problem, incorporating SLA requirements into the objective and using meta-learning to transfer tuning experience across workloads and environments. This warm-starting strategy accelerates convergence and yields robust multi-resource configurations.

- More recent BO frameworks refine classical BO for DBMS and data-analytics tuning. *TopTune* decomposes the configuration space into categorical and continuous subspaces, applying SMAC to the former and Gaussian Processes to the latter, with inter-space communication, dimensional projection, and batch BO for parallel trials; it reports up to 12.2× faster convergence and 10.7% throughput gains on MySQL and Dameng benchmarks [39]. *LOFTune* targets distributed engines such as Spark SQL, where each workload run is costly [40]. It uses a multi-task SQL representation model, a hybrid sampling strategy (Conformal Prediction + multi-armed bandit), and a recommendation–sampling–decoupled framework, reducing execution runs by up to 90% when sampling is allowed and cutting latency by 41% in zero-sampling scenarios.

## Reinforcement-Learning Auto-Tuners

Reinforcement-learning (RL) tuners cast configuration tuning as a sequential decision problem in which an agent interacts with the system and learns a policy for adjusting knobs based on reward signals derived from performance metrics.

- **UDO** (Universal Database Optimizer) is an RL-based framework that jointly optimizes configuration parameters and index selections [25]. It addresses delayed feedback via a delayed variant of Hierarchical Optimistic Optimization (delayed-HOO), batches and schedules expensive reconfigurations, and distinguishes "heavy" knobs (requiring restart) from "light" knobs (online) to minimize disruption. Beyond configuration tuning, UDO integrates automatic index tuning in a unified policy.

- **CDBTune** is an end-to-end configuration tuning framework that uses *Deep Deterministic Policy Gradients* (DDPG) to manipulate continuous-valued knob settings [13]. It pre-trains its agent offline using synthetic workloads to obtain a general policy, then refines this policy online using feedback from real workloads. This combination of offline pretraining and online adaptation yields both sample efficiency and workload specificity, making CDBTune suitable for complex tuning tasks.

- More recent works extend deep RL beyond generic knob control by incorporating query semantics and learned data structures. *Db2une* introduces QBERT, a transformer-based query embedding model, and uses Proximal Policy Optimization with a multi-phased, metadata-driven training strategy that interpolates optimizer costs and statistics, thus avoiding expensive query executions while outperforming IBM experts and prior query-aware tuners on analytical workloads [41]. *LITune* adapts DRL to learned index structures (LIS), with an adaptive training pipeline and an $O^2$ online updater to track workload drift; by tuning LIS parameters (e.g., node size, split policy, gap ratio), it reduces runtime by up to 98% and boosts throughput 17× relative to expert defaults [42].

## LLM-Based Auto-Tuners

LLM-based tuners leverage large language models to mine documentation, best practices, and textual workload descriptions, injecting domain knowledge into the tuning process and often guiding a downstream optimization algorithm.

- **DB-BERT** fine-tunes BERT on DBMS manuals to rank and suggest critical knobs [32]. It iteratively aggregates textual hints into candidate configurations via a greedy covering algorithm, evaluates these candidates on DBMS benchmarks, and updates the model using observed performance rewards, returning the best configuration within a tuning budget.
- **GPTuner** integrates *Large Language Model*s (LLMs), such as ChatGPT-style architectures, into the tuning pipeline to bridge expert knowledge and automated optimization [34]. The LLM extracts and structures configuration knowledge from documentation and best practices, which is then used to guide a Bayesian optimization loop. Lightweight benchmarking validates proposed configurations, enabling fast feedback and iterative refinement in a human-in-the-loop paradigm.
- Other LLM-based tuners further tighten the coupling between LLMs and BO. The *Rabbit* framework complements LLMs with retrieval-augmented generation (RAG), building a knowledge base from graph-encoded documentation and historical tuning experience [43]. It performs dependency-aware key-knob selection and multi-agent domain pruning to shrink the search space, then uses LLMs as few-shot surrogates with adaptive transitions between pruned and broader domains, achieving robust gains on DBMS benchmarks. *LATuner* integrates LLMs directly into the BO pipeline using zero-shot warm-starts, LLM-guided sampling, and a multi-armed bandit strategy that adaptively switches between LLM and Gaussian Process surrogates, yielding faster convergence and lower cost than prior BO-based tuners across diverse workloads [44].

## Transfer Learning & Historical Data

Beyond the choice of optimization algorithm, many tuners exploit transfer learning and historical data reuse to reduce the search space and accelerate convergence. These techniques leverage past tuning experience or existing performance telemetry to inform current optimization tasks, avoiding expensive trial-and-error from scratch.

Several state-of-the-art tuners, such as OtterTune [29] and ResTune [38], explicitly incorporate historical data to warm-start the search process. OtterTune constructs workload-specific performance models from a large repository of prior tuning results and uses Bayesian optimization guided by workload similarity. ResTune, in turn, applies meta-learning to transfer knowledge across workloads, improving sample efficiency in constrained Bayesian optimization for CPU, memory, and I/O knobs.

While these methods significantly reduce the number of required benchmark iterations, they rely on access to high-quality, large-scale historical datasets, which can be costly to collect and maintain. This poses a limitation for new users or systems with limited instrumentation, as reproducing the results without comparable data can be difficult. Pretraining-heavy methods such as CDBTune [13] and Hunter [30] require curated warm-up datasets or offline synthetic workloads to bootstrap their deep learning models, raising concerns about generalizability and applicability across different environments.

In contrast, tuners like iTuned [27] and BestConfig [26] operate without any prior data, relying solely on online exploration. While potentially slower to converge, these methods offer better portability and reproducibility in settings where historical data is

unavailable or system-specific, but they typically require human expertise to predefine or constrain the search space (e.g., select knobs or bound ranges), which may introduce manual bias.

Overall, transfer learning and historical data integration offer substantial acceleration benefits, but their effectiveness hinges on the availability of rich prior knowledge and the similarity between past and current workloads. Their reliance on high-quality datasets can hinder widespread adoption and make reproducibility a key challenge for real-world deployment.

### Real-Time Adaptability

A key requirement for practical DBMS tuning is the ability to adapt in real time to changes in workload characteristics. In production environments, workloads are rarely static—they evolve with time-of-day patterns, user behavior, query mix, and system load. Consequently, static configurations optimized for a single workload snapshot may quickly become suboptimal, leading to performance degradation or SLA violations. Real-time adaptability is thus crucial for maintaining robust performance in dynamic environments.

However, most existing auto-tuning systems are designed for static or quasi-static workload settings. Tools such as PGTune and BestConfig generate a configuration based on an initial hardware profile or short-term benchmark, but do not adjust parameters once the workload shifts. Similarly, OtterTune, Hunter, and CDBTune typically assume that the workload remains consistent during both training and deployment phases. While OtterTune can map new workloads to previously seen ones, this mechanism is limited by the diversity of its historical dataset and cannot guarantee fine-grained responsiveness to workload drift.

Few tuners implement mechanisms for continuous monitoring and adaptation. iBTune stands out by periodically adjusting memory allocations based on real-time LRU miss ratios, but its capabilities are narrowly scoped and do not address broader tuning aspects such as multi-resource configurations or workload reclassification.

This lack of online reactivity reveals a major limitation in the current landscape of auto-tuning research. As database systems move toward greater automation, future tuning frameworks must incorporate feedback loops, workload-change detection, and policy updating mechanisms to remain effective under non-stationary conditions. Without such real-time adaptability, even the most sophisticated tuning algorithms risk becoming obsolete in the face of workload dynamics.

## 2.4 Challenges and Motivation

Taken in isolation, the methods reviewed so far could give the impression that "there is a tuner for everything": rules and heuristics for quick configuration, Bayesian optimizers for careful exploration, RL agents for sequential control, and LLM copilots to glue everything together. However, the reality observed in practice is more sobering: DBAs still spend considerable time debugging configurations, compensating for workload drift, and validating benchmark claims.

This gap between the promise of autonomous tuning and its actual deployment motivates a systematic analysis of the underlying challenges. In what follows, we identify the main sources of friction—measurement noise, workload dynamics, data and expertise requirements, and evaluation limitations—and use them to frame the motivations for our own work.

## Reproducibility & Evaluation

Additionally, 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 pre-training 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 [13, 29].
- **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 [45].

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 industrial applicability.

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 [46, 47], OS/VM background activities [48], intrinsic benchmark randomness [49, 50], and hardware side-effects like CPU throttling [51, 52]. We evaluated performance variability across three representative workloads: two OLTP workloads, TPC-C [50] and Sysbench [49, 53, 54], and one OLAP workload: TPC-H [55]. Table 2.3 quantifies run-to-run variation for 10 runs on the same MySQL database under default configuration.

Table 2.3: 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 0.5%. Additionally, ML-based tuning algorithms, such as *Bayesian Optimization* (BO), exhibit inherent stochasticity [56], 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 define the stopping iteration $t^*_{\text{OLTP}}$ as the earliest iteration $t$ (after at least 100 steps) where the maximum throughput over the previous 100 iterations does not improve by more than 5%. The threshold of 5% is chosen due to typical measurement noise in OLTP benchmarks, which is approximately 5%. We then denote the optimal throughput as:

$$P^* = \max_{t<t^*_{\text{OLTP}}} \Big(\text{max\_tps}(t)\Big).$$

Convergence is declared at the earliest iteration $Y_{\text{OLTP}}$ satisfying: $Y_{\text{OLTP}} = \min\Big\{t \mid \text{max\_tps}(t) \geq 0.95\, P^*\Big\}$, indicating when throughput reaches at least 95% of the identified optimal value. The 95% threshold is chosen empirically to approximate the noise in OLTP benchmarks.

**For OLAP workloads**, we define the stopping iteration $t^*_{\text{OLAP}}$ similarly, but considering total query-batch execution time. Specifically, $t^*_{\text{OLAP}}$ is the first iteration $t$ (after at least 100 steps just to make sure that no more tuning potential is present) where the minimum execution time over the previous 100 iterations does not decrease by more than 1%. This threshold of 1% reflects the lower variability (below 1%) typically observed in OLAP benchmarks. We denote the optimal execution time as:

$$E^* = \min_{t<t^*_{\text{OLAP}}} \Big(\text{total\_exec\_time}(t)\Big).$$

Convergence for OLAP workloads is declared at the earliest iteration $Y_{\text{OLAP}}$ satisfying: $Y_{\text{OLAP}} = \min\Big\{t \mid \text{total\_exec\_time}(t) \leq 1.01\, E^*\Big\}$, indicating that the total execution time is within 1% of the optimal runtime. This 101% threshold is chosen empirically to approximate the noise in OLAP benchmarks.

Table 2.4 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. *Bayesian Optimization* (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 2.4: 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 | 1.2 | 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 apply Welch's two-tailed $t$-test [57] for pairwise comparison alongside the non-parametric Friedman-test [58] for multiple results comparison to ensure robust significance comparisons.

More precisely, Welch's two-tailed $t$-test evaluates whether the means of two independent samples differ significantly by computing a $p$-value—the probability of observing an effect at least as extreme under the null hypothesis—whereas the Friedman-test first ranks observations within each test (assigning average ranks), aggregates these ranks across related conditions, and computes a chi-square–based statistic reflecting the variability of those rank sums, using its $p$-value to assess the likelihood of the observed differences under the null hypothesis; we use a significance level of $p < 0.05$. While this protocol reduces bias, it cannot eliminate it. 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.

## Limitations of DBMS Auto-Tuners

The landscape of DBMS auto-tuning has evolved significantly, introducing a wide range of techniques from expert-driven heuristics to deep reinforcement learning and large language models. These tuners offer various benefits—reduced manual effort, improved performance, and faster convergence—but they also face critical limitations that hinder their widespread adoption in real-world systems.

Many tuners optimize well under controlled, static workloads but lack the flexibility to adapt to runtime changes. Only a handful (e.g., iBTune) include limited online adaptability mechanisms, and even those are narrowly scoped. Transfer learning and pretraining methods such as those in OtterTune, Hunter, and CDBTune accelerate tuning but rely on high-quality historical data, which is expensive to gather and difficult to generalize across systems. Reproducibility is further challenged by algorithmic stochasticity, complex setup procedures, and hardware/software variability, making many of these methods hard to replicate or deploy outside of research environments.

Another persistent challenge lies in balancing tuner complexity with practicality. Sophisticated approaches (e.g., DRL or meta-learning) may offer theoretical performance gains but introduce high overhead, hyperparameter sensitivity, and steep learning curves for deployment. Lightweight tuners (e.g., PGTune, iTuned) are easier to use but lack adaptivity and often depend on manual search space constraints.

Overall, no existing solution fully achieves the trade-off between adaptability, robustness, and ease of use. These limitations highlight a critical need for practical, lightweight, and real-time adaptive tuning strategies that operate effectively under dynamic workloads and realistic resource constraints.

In the following chapters, we address this gap by proposing two novel systems: a practical, sampling-efficient configuration tuner and a fully adaptive resource optimization framework capable of real-time adjustments. Together, they aim to bring DBMS auto-tuning closer to operational feasibility in industrial settings.

# Beyond Technical Barriers: Motivation for a Qualitative Inquiry

While prior sections have reviewed the technical landscape of DBMS automation, highlighting algorithmic innovations and practical limitations, an equally critical, and often overlooked, dimension is the human factor. In practice, even sophisticated tuning and automation tools see limited adoption in enterprise settings. For instance, within Orange we observed that despite rapid progress in AI/ML-based automation for DBMS configuration and resource management, actual usage remains limited.

This mismatch between technological capability and organizational uptake raises an important question: *What prevents DBAs and system administrators from embracing automated solutions in practice?* Addressing this adoption gap requires moving beyond benchmarking and algorithmic comparisons to examine how users perceive, interact with, and trust automation in real operational contexts.

Guided by these questions, the next chapter (chapter 3) reports a qualitative study that explores the real-world challenges and opportunities for DBMS automation adoption using grounded theory.

# Chapter 3

# Challenges & Opportunities in Automating DBMS: A Qualitative Study

Whereas the prior chapters synthesized technical advances in DBMS automation, this chapter turns to the *human and organizational* dimensions through a qualitative inquiry. Prior work has noted the importance of human factors for automation adoption [59], but often relies on anecdotal observations without a rigorous methodological backbone. In adjacent areas of software engineering and system management, qualitative methods have proven effective at discovering such barriers: for example, Smith et al. [60] used semi-structured interviews to examine how developers detect security vulnerabilities; Ournani et al. [61] combined focus groups and surveys to uncover energy-optimization challenges; and Habchi et al. [62] mixed interviews with log analysis to study static-analysis tool adoption in mobile development.

Building on these precedents, we employ *grounded theory* to investigate how practitioners perceive, evaluate, and integrate DBMS automation into real operations. We conduct semi-structured interviews and analyze transcripts using iterative coding with constant comparison, memoing, and category refinement—following open, axial, and selective coding phases [63, 64]. This approach enables us to develop a mid-range conceptual model rooted in practitioners' experiences rather than presupposed hypotheses.

We organize the empirical results into two complementary perspectives: **Technical constraints**: safety, cost, and fit with evolving workloads and environments; **Human factors**: trust, need for control, and limited time/skills to adopt new tools. We close the chapter with implications and design guidelines that inform our later system contributions (configuration tuning and real-time resource management).

## 3.1 Methodology

To achieve our objective, we adopted a qualitative research approach with Grounded Theory methods [65], including procedures such as coding, memoing, and theoretical saturation [66]. Despite being complex and time-consuming [67], this approach allows us to generate new insights from the data rather than testing pre-determined research questions.

### 3.1.1 Interviews

Following empirical software engineering standards, we designed semi-structured interviews with initial and adaptive follow-up questions, guided by [76] for clear objectives

Table 3.1: List of Interviewees and their Profiles

| ID | DBMS Type | Current Role | Experience | Entity |
|---|---|---|---|---|
| I1 | XtraDB [68], Redis [69], MongoDB [70] | Manager | >20 years | Cloud Platforms |
| I2 | Oracle Database [71], MariaDB [72], PostgreSQL [73], SQLserver | Manager | >20 years | Cloud Service |
| I3 | MongoDB | DBMS expert | >20 years | Cloud Service |
| I4 | Oracle Database, PostgreSQL, MariaDB, MongoDB | DBMS expert | >20 years | Products Support |
| I5 | Oracle Database, PostgreSQL, Cassandra [74], MariaDB | DBMS expert | >20 years | Cloud Service |
| I6 | Oracle Database, MariaDB, MySQL, PostgreSQL | DBMS expert | >20 years | Cloud Service |
| I7 | MongoDB, MariaDB | Product Owner | >10 years | Financial Service |
| I8 | Oracle Database, PostgreSQL, MySQL, SQLserver | Manager | >20 years | Network |
| I9 | MongoDB, Elasticsearch [75], Redis | Architect Cloud | >10 years | Cloud Infrastructure |
| I10 | MongoDB, Oracle, Cassandra | Developer | >15 years | Financial Service |
| I11 | MongoDB, Oracle, MySQL, MariaDB, PostgreSQL | Developer | >20 years | Cloud Service |

and open dialogue. Our approach primarily utilized open-ended questions, with further probes for detail, centered around 12 core questions, allowing exploration based on participant responses.

1. Which databases and tools related to databases do you use?
2. What is the automation of database administration for you?
3. What importance do you give to database administration automation?
4. What tasks do you perform when managing DBMS?
5. How much time would you spend on each of the tasks?
6. How do you prioritize your tasks based on the impact and the importance?
7. Which tasks do you find 'painful' and why?
8. What are the difficulties you face when maintaining and managing DBMS?
9. Do/did you automate some of these tasks? What motivated this automation? Which tools do you use? When did it fail?
10. What are the benefits and limits of such routines/tools? What are the limits to the automation of some tasks?
11. Do/did you actively search for automation tools for database administration, and why?
12. What are the key properties and support that should be improved by the current database administration?

The questions (1) to (3) help to cover the participants' knowledge and awareness of DBMS administration automation. Then, questions (4) to (8) aim to discover the routine, daily activities, and difficulties encountered by our interviewees. Finally, questions (9) to (12) allow us to investigate further the challenges and opportunities in DBMS administration automation.

Interviews with I1 to 11 are conducted either in-person or via video conference, based on their availability and location. The interviews progressed through three phases: an initial narrative introduction detailing the study's aim and interview process, followed by a semi-structured segment, during which we posed the above-mentioned questions along with follow-up inquiries. It concluded with a segment dedicated to addressing participants' questions and sharing further information. The mean duration time of the interviews is 53 minutes, with a minimum duration of 31 minutes and a maximum duration of 66 minutes.

### 3.1.2 Interviewees

Participants were selected based on their extensive experience with DBMS, which facilitates a deeper understanding of the technology's details, strengths, and limitations. To ensure a diverse range of perspectives, all participants have at least 10 years of experience and continue to work with DBMS daily.

The participants in our study were volunteers who responded positively to our interview invitation and were actively engaged in DBMS projects at an international telecommunications company, boasting over 100,000 employees. The rationale behind choosing participants from the same company, such as in [77, 78], is to assess the role of the company in the practice of DBMS experts. However, our study specifically focuses on promoting DBMS administration automation within a company and providing detailed empirical analysis, without aiming to create a universally applicable model for companies

of varying sizes, sectors, or policies. It is important to note that the company is large and comprises many different entities, each with its own independent ecosystem and operational practices. This organizational structure ensures a wide range of perspectives and practices. We carefully selected participants to ensure diversity in their departments, roles, countries, and policies. This was done to avoid homogeneity in their approaches to DBMS and to enhance the generalizability and transferability of our findings.

Rather than setting a specific number of participants, we conducted interviews until we reached a level of data saturation as in [79], which occurred after 11 interviews. Despite differences in the technologies and project types mastered by the participants, we observed convergence in the collected data and thoughts, which is consistent with similar works that have studied similar populations [62, 60, 61]. To protect the privacy and confidentiality of our participants, we used code names ranging from I1 to I11 and omitted any sensitive information, such as team or project names.

Table 3.1 shows the characteristics of our interviewees, their familiar DBMS technologies, their functions, their experiences, and their entities. The main criteria for the interviewee selection are based on their experience, and we try to have diverse profiles among our interviewees. I1 manages a *Database-as-a-Service* team. I2 transitioned from managing a database optimization team to cloud services. I3, I4, I5, I6 are senior DBAs, experts in DBMS with different types of technologies. I7 is a technical product owner who oversees DBMS deployment and administration. I8 is a manager of a team of DBAs, who is in charge of operating a set of databases. I9 is a cloud architect who mainly works with managed solutions with cloud providers. I10, I11 are developers who work with an application that highly relies on DBMS. It is important to note that engaging experts from various departments and countries presents challenges, primarily due to the time their demanding schedules and zone differences.

### 3.1.3 Transcription

For each conducted interview, we transcribed the recording using a denaturalized transcription approach, which preserves the original features of speech—such as pauses, repetitions, and informal expressions—rather than editing them into written language. This method, also adopted in similar studies such as [62, 61], allows for a more faithful representation of participants' spoken discourse. This approach emphasizes the interview content and is less exhaustive than other methods, such as verbatim in [80], while still being trustworthy. The transcription was done in the same language as the interview, but we translated some parts into English to include participant opinions in Section 3.2.

### 3.1.4 Analysis

We employed the Straussian grounded theory coding procedure [66, 63] for data analysis. We start with the open coding phase, where we carefully read our transcripts multiple times to condense each chunk of data into a label, based on the inferred meaning of the text. These labels were referred to as "open codes". Subsequently, we conducted axial coding to establish the connections among the previously extracted open codes. We then utilized selective coding to pinpoint the central ideas that encompassed the collected data. Lastly, we reviewed the transcripts again and assigned all content segments to a core idea that was relevant to the selected data.

Table 3.2: Summary of interview content analysis

| Topics | Ideas | | I1 | I2 | I3 | I4 | I5 | I6 | I7 | I8 | I9 | I10 | I11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DBMS users' awareness & Knowledge of DBMS automation | I am doing DBMS automation | | ✓ | | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | | |
| | DBMS needs less automation than before | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | | ✓ | |
| | DBMS automation can reduce repetitive work and save time | | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ |
| | DBMS automation can reduce resource consumption | | | | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ |
| Difficulties encountered & Automation opportunities | Anomaly Detection | Scale | ✓ | | ✓ | ✓ | ✓ | | | ✓ | ✓ | | |
| | | Peripheral | | ✓ | | | | ✓ | ✓ | | | | |
| | | Application's behavior | ✓ | ✓ | ✓ | ✓ | | | ✓ | | | | |
| | Users have no adequate experience | | ✓ | ✓ | ✓ | | | ✓ | ✓ | | ✓ | | |
| | Query, execution plan optimisation | | | ✓ | | | | ✓ | | | | ✓ | ✓ |
| | Operation planning | | ✓ | ✓ | ✓ | ✓ | | ✓ | | ✓ | | | |
| | Capacity planning | | | | ✓ | ✓ | ✓ | | ✓ | | | | |
| Challenges & Constraints in DBMS automation application | Technical Factors | Automation can be dangerous | ✓ | ✓ | | ✓ | ✓ | | | ✓ | | | |
| | | The cost automation can be high | | | | ✓ | | | ✓ | | ✓ | | |
| | | Automation need to adapt to application | ✓ | | ✓ | ✓ | | | | | | | |
| | Human Factors | Lack of confidence for automation tools | | ✓ | | ✓ | | ✓ | | | ✓ | | |
| | | Users want absolute control over the product | | | | ✓ | | | ✓ | | ✓ | | |
| | | Users may lack support | | | | | ✓ | ✓ | | ✓ | | | |
| | | Users lack time to apply new tools | | | ✓ | | | ✓ | | | ✓ | | |
| | | Automation could cause lose of expertise | | | | | | ✓ | ✓ | | | | |

## 3.2 Observation & Findings

Table 3.2 summarizes the key insights from our qualitative study, with the core ideas that fit our objectives. Here, we use 'DBMS users' as a general term to refer to the interviewees, whose role and focus vary. In each cell of Table 3.2, a reference mentioned by the participant regarding every idea that falls under the core idea is indicated by the cross mark (✓). The following section delves into our observations, with each subsection focusing on a reported idea. Each idea is then given its dedicated paragraph for discussion. Similar ideas that convey related meanings and objectives are clustered together under a single category. To summarize the observations and findings and to offer our insights and recommendations, a discussion is provided at the end of each category.

### 3.2.1 DBMS Users' Awareness & Knowledge of DBMS Automation

#### Current Routines in DBMS Administration Automation

Our research indicates that all interviewees are knowledgeable about automating DBMS administration, with many having experience in this area. However, only a small number of them are familiar with AI/ML-based automation tools. The primary approaches to automation mentioned in our study include Kubernetes operators, schedulers, scripts, such as shell and infrastructure-as-code-based software, monitoring tools, and DBMS internal tools.

I1 reported that "*We use Kubernetes operator to automate the majority of the tasks*". I1's team provides managed database-as-a-service using mainly containers with the help of Kubernetes [81], which is a container orchestration system for automating software deployment, scaling, and management. Kubernetes Operators extend Kubernetes' capabilities for managing complex, stateful workloads, allowing for automated rolling updates, failure recovery, and many other functionalities.

The scheduler is raised to be an important part of DBMS administration automation. "*We use the scheduler to make maintenance operations such as rebuild index, reorganization when the workload is light*", reported I6. Recurrent operations, such as backup, are also programmed with the scheduler.

Another important aspect of DBMS automation concerns the use of external and DBMS-native tools. Especially for database monitoring, which can be heavy work without proper tools if visualization of the databases' status is required. Software, such as Grafana [82], Monolog [83], and native tools for different DBMS technologies, are essential for the diagnostic and to ensure the databases' status.

"*We develop and use scripts to automate tasks*" reported I3, I5, I6, I8, I9. Scripts are widely applied in DBMS administration automation, I5, I6, and I8 reported using scripts of infrastructure-as-code software, such as Terraform [84] and Ansible [85] to automate the database deployment process. Such software allows users to define infrastructure using a declarative configuration language, such as HashiCorp [86], therefore saving the manual process of database deployment.

Scripts are used in many cases, especially regarding repetitive tasks. "*We create our homemade scripts every time we see repetitive tasks*", reported I8. As for concrete examples, I6 reported using scripts to automate the statistic calculation process in Oracle databases, which is a crucial operation to keep Oracle databases efficient. DBMS updates and anomaly detection are also reported to be automated by using scripts by I9. Despite widespread script use, automation potential is constrained by the script developer's expertise. As I9 notes, automating administrative tasks demands a thorough understanding of the DBMS and significant time investment. "*It's always a human who writes the scripts*

*and programs the Scheduler. It's always the human who decides afterward what to do with the program and its frequency to be executed.*" reported I5.

**Discussion.** We observe that in DBMS administration, there are numerous recurrent and repetitive tasks, and the participants demonstrated a keen interest in automating these tasks using various methods, like custom scripts and schedulers. However, it appears that the level of automation achieved by these routines is limited and constrained by the developer's expertise, and the utilization of AI/ML-based approaches for DBMS administration automation remains quite limited among the participants.

### DBMS Automation Evolution

"*DBMS needs less automation than before*" is a mutual feeling shared among I1, I2, I3, I5, I6, I10, implying that the DBMS is getting easier to access and use compared to older versions of DBMS.

Firstly, one of the factors contributing to the increased accessibility of DBMS is the efforts made by DBMS vendors to enhance system stability and robustness. "*Investigating why the database failed is probably the most annoying task. Understanding why their nodes fall and restarting them, but these are things that we see less and less because little by little, the DBMS products have increased the robustness and eliminated the factors that caused the nodes to fall*" is witnessed by I1.

By enhancing the robustness of the DBMS product, the DBMS is considered more automated by our interviewees, since fewer DBMS incidents need to be managed. I1 reported a case where they provide a 3-site cluster, which is very sensitive to the latency of different sites. During slowdowns between sites, nodes of a site could become isolated or crash. XtraDB later added the timeout parameters at the communication level between the nodes, and the robustness is enhanced, and they have fewer problems.

I6 also observed that earlier versions of the Oracle database experienced multiple optimizer changes, impacting the performance of existing queries: "*In the past, for a slight update of the Oracle database, we were still working 2 months after the update. Because there were treatments that we hadn't detected. But since v11, v12, and so on, we no longer have these problems.*" As a result, close monitoring of database performance was necessary to prevent any production errors during updates and upgrades. However, this is no longer the case with the latest versions of Oracle databases, even though there are still many modifications made to the optimizer by Oracle according to their white book [87]. Although the improved compatibility among different versions is not tailored to enhance the automation of the DBMS, it eliminates the necessity for extensive human intervention, which once demanded automation and significant effort.

Secondly, new algorithms and tools have been introduced by the vendor to help with DBMS administration automation. "*Oracle database have implemented many algorithms to automate tasks such as the optimization of queries and database resizing*" reported I4. "*You have paid options in Oracle database, such as Advanced Security, to ensure the security of your databases and an encryption option to do TLS*", stated I6.

Thirdly, more recent non-relational databases (NoSQL) are considered to be lighter than traditional relational DBMS. "*For the PostgreSQL database, you need to do the memory tuning. But MongoDB will manage its memory in relation to the RAM that you have allocated to the OS, and it will manage its own memory of its WiredTiger engine. It does it rather well*", as reported by I7. The feeling is shared with I3: "*Apart from the backup and the deployment, in terms of administration and operation, reorganization of data is the only task that I feel like is necessary to be automated around MongoDB.*" I10 also thinks highly of the flexibility and self-sufficiency offered by MongoDB. As a

developer, I10 occasionally relies on the assistance of DBAs when dealing with relational DBMS like Oracle. However, due to MongoDB's comparative ease of management, he can handle tasks independently.

**Discussion.** DBMS publishers have significantly improved accessibility and user-friendliness; this evolution towards greater robustness and stability has reduced the need for extensive human intervention, once a critical necessity for automation and effort. Interviews reveal that many believe modern DBMS demands less automation compared to their predecessors.

#### Importance of DBMS Administration Automation

During our study, every interviewee expressed the belief that the automation of DBMS administration is crucial. "*Automation can reduce repetitive work and save time*", mentioned by the majority of our interviewees. According to I1: "*We try to eliminate repetitive work as much as possible [...]. Every time we see a task that needs to be done more than once, we try to automate that immediately.*"

"*Automation saves time and allows us to do other more interesting tasks*", stated I7. For I9, the benefit of DBMS administration automation is making complicated tasks straightforward, which allows people with less experience to solve problems in production. I10 also highlights the fact that not only does DBMS administration automation reduce repetitive work, but the automation of certain manual tasks also makes the process more reliable.

DBMS administration automation is also mentioned as being able to reduce computational resource consumption. "*For example, the benefit (automation with intelligent scripts instead of a fixed scheduler) is to avoid rebuilding an index when it is unnecessary and saves the CPU and RAM consumption*" stated I5.

**Discussion.** Automation of DBMS administration brings many benefits; the main benefits mentioned by our interviewees are: saving time, saving computational resources, and reducing repetitive work. These benefits are the key factors that drive the automation of DBMS administration.

### 3.2.2 Difficulties Encountered & Automation Opportunities

Although our participants find that the DBMS are getting increasingly automated and robust, and they have approaches such as scripts and schedulers to automate certain tasks, there remain some difficulties and challenges when automating DBMS administration. These challenges offer opportunities for researchers working in the field of DBMS.

#### Anomaly Investigation

The investigation and resolution of the alarm, incidents, and possible failure of the databases in the production environment, and repairing the system, is considered to be one of the top priorities (I1, I5, I7, I8, I9). "*The priority is to ensure the quality of service*", reported I1. Being one of the main difficulties encountered by DBMS users, anomaly investigation remains hard to automate for the following three reasons:

**Scale.** In large applications, we face multiple environments (different operating system releases, different database technologies, and versions) and multiple databases (up to $2,000$ instances reported by I1), and this complicates the investigation of the problem

and other administration tasks. "*It's okay when you have 2 environments, we can afford to verify them daily, but it's different when you have 200 different environments*", reported I4. And it is often difficult to make tests on different environments, "*You see problems in the production environment that you do not see in the test environment*", reported I9 when trying to write a script to hot update the databases.

"*What takes us most time is the fact that we are managing thousands of databases*" reported I1. Managing databases on a large scale is a widespread issue, according to I5, "*An average DBA in teams that manage large environments, has to manage several hundred environments*".

Cluster deployment, aimed at ensuring high availability and fail-safe protection, complicates DBMS management and anomaly resolution. "*More complex the architecture of the clusters is, more nodes we have, and we have more things to manage, we are going to have additional problems related to latency, anomalies, and organization of data within the shard clusters*", reported I3.

I7 reported a case with a MariaDB cluster, "*we have several databases in a cluster, if you do a physical backup, you won't be able to segregate the databases inside your backup, which means if you can't restore a single database, you have to restore all the databases. But it is not the case if you switch to logical backup.*"

**Peripheral.** Secondly, the problem can come from the peripheral devices and not the DBMS. "*I see an alert in production due to the loss of user connections. And it was not related to the base, it was a network failure, the problem was peripheral*", reported I2. I7 also reported backup failure only because there was not enough space on the disk allocated to the backup operations.

"*I have a query that takes too long. So we have to find if it is the query that is badly written, if it is incoming data that is not going well, if it is the settings that are not going well, if it is the disks, because sometimes the disks are slow*", witnessed I6. When investigating a problem, we must consider all the possible failures, and peripheral issues are not negligible.

**Application's Behavior.** Thirdly, databases are attached to applications. Unpredicted behaviors in the application can cause the database to malfunction, "*When we observe incidents, for me this is more related to the current activity of the application than operational tasks*", reported I3. I3 witnessed a case when an application did many inserts and deletes, which caused the database to disorganize and occupy very large disk space while only containing a small amount of data. In addition, I4 reported having seen projects launched multiple times with unnecessary batch treatments on tables, causing the database size to grow unexpectedly. I7 reported a case of a wrong database address used in the application, causing the database to fail. I2 stated that "*what we often see in backup and restore, are application errors, data corruption, and human errors.*"

I1 witnessed a case of database failure where an extreme and not previously previewed workload is applied to their databases (insert and delete of big images in short periods), "*The DBA needs to understand the database's application's behavior to understand the potential problems. When investigating a problem, the fact that we need to have interactions with clients to understand what they are doing to understand where they went wrong is where we have most trouble with*".

Yet, despite the fact that the anomaly of databases has various and complex causes, some anomalies are considered as problems that can be anticipated. "*You see certain errors in certain files, such as log files, that arrive from time to time, it makes the problem detectable in advance and can be dealt with proactively*", stated I7. Many basic anomalies can be detected using monitoring software and scripts, reported I9.

**Discussion.** It is hard to uncover the root causes of the databases' incidents, especially in the context of multi-environments, multi-instances, and multi-nodes in clusters. Databases' behavior is also related to the applications' behavior and the peripheral devices that they are attached to. These factors make the investigation of the databases' problem difficult to automate.

Earlier research on anomaly detection, such as [88, 89, 90], primarily concentrated on identifying malicious activities. Other studies [91] were beneficial in determining the timing of anomalies, but had limitations when it came to pinpointing their root causes. Although these works hold great significance, they do not inherently address the challenges we uncovered in our study, including issues of scale, application behavior, and peripheral concerns.

### Users Miss Adequate Experience

The lack of experience and expertise of users is another difficulty reported by I1, I2, I3, I6, I7, I9. Depending on the profile of the interviewee, the 'User' that they referred to varies from the application development team to database administrators.

"*Users and database administrators don't know the product*", stated I9. I9 reported a case where a DBA received an alarm indicating the lack of disk space. To solve this, the DBA removed an unnecessary index; however, the index was actually required for query execution, and its removal caused severe performance degradation and service unavailability. "*We can add CPU and RAM to meet performance requirements, but we are more constrained when we have an extreme workload due to a lack of knowledge or expertise of the developers*", reported I1. The lack of experience of the developers and DBA from the application team can influence the operation of the DBMS, and we can only acquire experience with time. "*I have people on the production side that I have known for several years, they are starting to be good but they knew nothing when they started*", witnessed I3.

**Discussion.** Being extremely complex software, DBMS is known to be hard to manage since they require experience and a thorough understanding of the system [29, 92]. Misjudgment due to the lack of experience can cause the system to fail, and the fact that most users and DBAs only have limited experience is considered to be an actual difficulty in the industry.

### Query & Execution Plan Optimisation

Depending on the profile and the technology used by our participants, query and plan optimization to meet performance requirements are also mentioned to be one of the difficulties that they face in production (I2, I6).

"*One of the things we miss is to be able to have a system that verifies if our plans, our path, or our way of doing things is optimal. The DBMS calculates a scoring with statistics and paths, which is used to find the execution plan. But very often, the scoring is flawed*" stated I2 when experiencing Oracle databases.

"*The problem that we often encounter, is the query that is badly written*" reported I6, even with the query optimizer integrated with the DBMS I6 still witnesses inefficient queries from inexperienced users causing the system to stall. The fact that I6 needs to deal with queries from inexperienced users strengthens our point in Section 3.2.2.

Despite the fact that I10 has many years of experience around MongoDB, he still struggles with how to set optimal indexes, "*We don't have performance issues (with MongoDB), but personally, I know that my configurations are not optimal, [...], and I don't have a plan regarding how to create proper indexes*". I11 shares a similar experience, revealing

that the most challenging task for I11 involves fine-tuning complex queries and identifying the optimal data structure to boost the database's performance.

**Discussion.** Despite the efforts made by DBMS publishers regarding query and plan optimization, it is still considered a difficult subject. However, I6 also indicates that only 20% of applications would require a high-level optimization; it is not a mandatory procedure in most cases. And I1 stated that there is always the option to add more RAM and CPU to meet performance requests. I10 also brings up the challenge of identifying the optimal indexes for MongoDB; however, I10 is not particularly concerned by DBMS' performance issues. In short, optimization and fine-tuning are a challenge, but it does not appear to be a major obstacle in the activities of our interviewees.

### Operation Planning

To maintain DBMS QoS, users perform maintenance tasks like disk reorganization and upgrades, impacting the production environment by using significant computational resources. Minimizing operation duration is crucial to preventing service interruptions.

Currently, maintenance scheduling is based on the application's profile and relies on human expertise."*On XtraDB in particular, we have 5 or 6 instances that need optimizations (free up disk) from time to time. For one of our customers, this is done every 6 weeks, for the others, it may be every 6 months*", reported I1. I5 stated a similar observation, "*The frequency of scheduling of these operations depends on the application. In other words, there may be applications where there are many activities and modifications such as deletes, updates, etc. that disorganize the objects, so it may be necessary to update or rebuild the indexes of certain instances every week, whereas on other applications, it will not be disorganized.*" I6 also reported to use of integrated schedulers to set a specific time to run maintenance operations while the workload on the database is light.

Maintenance needs can be identified using DBMS metrics such as the fragmentation ratio. However, automating these tasks is challenging due to potential service impact. "*You have to be able to determine the most favorable moment to run the operation and you have to be able to predict the duration of the operation. This is why all these operations are manually programmed. When we have to carry out large defragmentation operations, we simply stop the service*", reported I4. The fact that system shutdown is necessary for certain operations complicates the operation planning, "*Most of the systems work 24/7. So it is not so easy to find the patching window.*" reported I8 when having to patch his DBMS on a 24/7 application. Furthermore, depending on the application and the type of the company, sometimes it is not up to the DBA to decide when they can shut down the system; it is decided by the responsible party from the business side or responsible with a higher hierarchy (I8). Although updates and upgrades present a challenge in 24/7 systems, the rolling update is reported to be applied to update the system while providing a stable service in certain use cases (I1).

**Discussion.** Maintenance planning is closely linked to application behavior, often impacting the production environment and sometimes necessitating system downtime. This planning demands expertise in understanding the operation's effect on the DBMS and the application's behavior. Predicting short-term workloads is crucial for assessing whether maintenance might affect QoS. Although there have been advances in workload prediction [93, 94, 95], these methods require high-quality training data and additional computational resources for precise predictions. However, obtaining such data can be challenging due to legal and security constraints in the industry, limiting its practical use.

#### Capacity Planning

Estimating a project's capacity is crucial for creating a robust system with a DBMS, involving calculations for computational resources like RAM, CPU, and disk space. Yet, accurately predicting these needs is challenging in the industry. Accurate capacity planning hinges on predicting project workloads, a detail often unavailable early in projects, as mentioned by I3.

This planning predominantly relies on limited human expertise. I4 highlighted the difficulty of achieving accurate capacity predictions, "*Projects come up with charts saying, 'Here we have these volumetric projections over a year', and we realize that either it's oversized, or it's undersized, it's rarely well sized*". When lacking an accurate capacity estimation, DBMS users tend to oversize the database to ensure that the database is operational. "*We tend to always increase resources and we rarely reduce them, and that's a big problem today with Corporate Social Responsibility, we have to think differently*", reported I4. The amount of wasted resources may be large, as I7 reported having reduced 350 unnecessary CPUs and 400 GB of unnecessary disk space in a project.

With the capacity demands changing over time, development efforts are also increased when capacity planning is not enabled. "*At the beginning of a project, we can use a non-partitioned model with 100 GB. And let us imagine that the base grows to 10 TB. We can no longer remain in a non-partitioned model*", stated I4. I3 also reported a case where a project forgot to delete inactive accounts, causing a waste of disk space. These problems can be avoided with accurate capacity planning.

Furthermore, accurate capacity forecasting enables DBMS users to anticipate and address issues proactively, such as determining the need for additional resources to handle growing workloads (I3, I4). According to I3, "*If we were able to predict the future derivatives of your database, you would be able to easily integrate other actions. Without the capability, you will often be on very instantaneous issues (i.e., forced to react to immediate problems rather than anticipate them)*".

**Discussion.** Accurate capacity planning and workload prediction enable proactive problem management, minimize unnecessary computational resource allocation, and facilitate advanced architecture design, thus conserving development efforts. However, the scarcity of detailed information from DBMS users and product owners at project initiation complicates precise planning and forecasting. Conversely, services such as Amazon Web Services [96], Microsoft Azure [97], and Oracle Cloud [98] now offer auto-scaling features that monitor applications and automatically adjust capacity, thereby reducing the complexity of capacity planning. Nevertheless, within the industrial context, numerous projects, particularly legacy ones, are unable to migrate to the cloud due to financial and legal barriers.

### 3.2.3 Challenges & constraints in DBMS automation applications

In this section, we discuss the challenges and constraints of automation tool adoption in the industry revealed in our study. The ideas that we revealed in our study are categorized into 2 categories: human factors and technical factors.

#### Technical Factors

**Automation Can Be Dangerous** The QoS of the DBMS is very critical in production. Administration tasks, such as backup and index rebuild, consume computational resources and, therefore, harm the database (I2, I7). The modification of important parameters and

execution plans can all impact the performance (I1), not to mention that some operations require the database to shut down and restart (I4).

"*Automating everything is a bit dangerous. Many actions that can be done in a completely automated way, but it poses problems for the operation of the databases when you start to touch the volume, the data, and things like that. We need to put more safeguards in place*", reported I1. I4 reported that Oracle databases are integrating AI-based tools to automate and help users with administration tasks. Even though these tools are great in theory, the fact that these tools may change execution plans and destabilize DBMS performance is dangerous.

**Discussion.** The fact that many administrative tasks could influence the databases' performance is considered to be dangerous in production. This makes people reluctant to delegate sensitive tasks to automation tools, because they may cause unexpected consequences if not thoroughly designed.

**The Cost of Automation Can Be High** Even though DBMS administration automation brings many benefits, there is still a cost behind the automation. "*When the version of DBMS changes, I may have to change my scripts. The maintenance of the final scripts is expensive too*", reported I9. Updates and upgrades of the DBMS bring changes; human efforts are needed to maintain the automation tools.

Automation also has a computational cost. "*If we let Oracle database auto-tune. It is devastating at the level of memory resources consumed by the Oracle database as it does a lot of operations. All these operations are also stored in their dictionary, so all this has a cost at the CPU level and the disk level. Is it worth setting up this system? I am not sure*", reported I4.

Many commercial cloud providers deliver managed database service and strong automation tools, even though they largely reduce the DBMS administration workload, their price is still blocking user adoption (I7).

Adopting an automation tool also adds an extra layer of complexity. In case of an anomaly and a problem, it increases the difficulty of resolving the problem. "*Behind it, there are events that appear in the alert file, so afterward we have to filter what can be problematic from what is not*", reported I4 when using an automation tool to resize the database. "*If I have an error, is it an error of my script or is it the DBMS underneath?*", I9 shared similar concerns.

**Discussion.** The automation of DBMS administration has a human cost, computational cost, and even financial cost, and it could potentially have a cost of increasing the complexity of the system. It is important to control trade-offs between the cost of automation and the benefits that automation can bring.

**Automation Needs to Adapt to the Application** The DBMS is attached to an application and must adapt to the requirements of the application and the human operation behind the application. "*For me, it cannot be all automated, there are times when human interaction is needed to understand the customer's use*", reported I1. This adds extra complexity to the DBMS automation. For example, I4 reported a case where an erroneous query is sent to the Oracle database with the auto-tuning option on, then the DBMS can consume a lot of resources only to make bad tuning and optimization decisions based on the erroneous query and degrade the DBMS's performance.

**Discussion** The application must be taken into account when doing DBMS administration automation; the changing application and possible human errors could cause the automation to fail. This complicates the adoption of automation tools.

### Human Factors

**Lack of Confidence in Automation Tools** Given that automation of administration tasks can be dangerous, DBMS users are more careful when adopting automation tools. "*We are always afraid that the automation tool does not do it as well as we do. [...] We are always worried because, in the end, it's still us who will be bothered somewhere*", reported I4. I2's team already has a well-done automation tool for backup and restore, but since it is a very sensitive task, they still demand a DBA to make sure the operation is successful. "*The automation tool must be perfect, otherwise it does not make any sense*", reported I8 and I9. Because of the sensitivity of DBMS regarding data integrity and data security, people often lack confidence in automation tools.

**Discussion.** DBMS are very sensitive; all unexpected and incorrect operations could jeopardize the entire system. For an automation tool to be adopted, the tool must be reliable for DBMS users to have deep confidence. In the end, it is not only about the robustness of the automation under all conditions but rather whether the tool appears to be trustworthy to the decision-makers. [59] also highlights that gaining the trust of DBAs is a major obstacle for DBMS administration automation.

**User Wants Absolute Control Over the Product** "*We want to control everything from A to Z*", reported I4. As highlighted in Sections 3.2.3 and 3.2.3, automation poses risks and lacks full user trust, especially given the sensitivity of DBMS. Users frequently demand total control over the system.

"*Oracle database tried to integrate several automatic tools to do auto-tuning. But it is dangerous to leave them active. The queries execution plans are critical to the business if we want to guarantee the stability of the performances, we prefer to fix the plans*", stated I4. Meanwhile, I4 also highlights that once the data model evolves, the fixed plan could no longer be valid. Delegating the tuning task to the Oracle database allows the system to adapt to evolving data models, but I4 made the compromise and chose the stability and total control of the execution plan. For I7, it is good to have an automation tool, but it is also essential for him to know how the tool functions and how the decisions are made by the tool.

Another critical aspect of having absolute control is that DBMS users must be able to access the product to resolve issues when problems arise. I9 uses a managed database service provided by the DBMS publishers, while this reduces the workload of the DBMS administration, I9 stated that "*when you're in trouble, you can't get your hands on it*".

**Discussion.** When delegating administration tasks to automation tools, DBMS users could lose control and track the states of the DBMS. This is blocking the adoption of automation tools. To avoid this, automation tools must reveal complete information about the changes that it has made and allow users to have global control.

**User Lacks Support with Open-source, Academic Project** Due to the increasing license fee of commercial DBMS products, our participants tend to adopt open-source and community solutions (I1, I4, I5, I6, I7, I8). As we have more control over the open-source DBMS, this is favorable for the academic community to research and develop automation tools for open-source DBMS.

However, academia and the open-source community have fewer resources compared to commercial products. Therefore, they only provide limited support. "*With open-source projects, the DBA is, unfortunately, less confident, sometimes he is a bit on his own [...] I would say that the DBA is worried when there are breakdowns, bugs*", reported I5.

I6 reported a case of trying to adopt an open-source automation tool for DBMS migration, but it only solves a partial problem and has limited support. I6 then abandoned this tool and decided to develop it on his own. "*open-source solutions cost us more, more human power on the maintenance because open-source needs more work, not everything is done ideally*", reported I8. The lack of support increases DBMS users' development costs and limits the adoption of open-source solutions.

**Discussion.** Many DBMS automation tools have been introduced in the open-source community by the researchers [93, 99, 100, 13]. In contrast to the comprehensive support offered by major corporations to their commercial automation tools, academic projects tend to provide limited support. The authors and researchers involved in these academic projects often have various commitments, making it challenging to establish contact with them.

**User Lacks Time to Apply New Tools** "*We do not have the time to do it*", reported I3, I7, I9. Our participants are overloaded with their daily activities, making it impossible for them to be up to date with the state of the art of DBMS administration automation tools and research topics. Nevertheless, they keep showing interest and have an open mind about ongoing research, "*I do not have that much time, it is a bit of a shame, but if I had more time I think I would keep myself a bit more up to date*", reported I7.

**Discussion** It requires time and effort for DBMS users to be up to date with the novel and research solutions. The fact that DBMS users lack time to make such inquiries creates a gap between the industrial and research communities, therefore restraining further adoption of newly introduced automation tools.

**Automation Could Cause Loss of Expertise** Loss of expertise caused by automation is another challenge that was revealed in our study. "*What customers really want is managed databases. They want to install, make queries and that the database lives on its own. They do not intend to calculate statistics or any other kind of thing*", reported I2. This is very common as DBMS automation and managed solutions can largely reduce the workload, and require less expertise. "*It will also diminish his knowledge and I have problems with all these black boxes, people do not know why they are in trouble because they do not know what they do. We have pseudo DBAs who do not know much and who have no idea what they are doing*" reported I7.

**Discussion.** While automation tools enable DBMS users to delegate administration tasks, this could cause a loss of expertise. On the other hand, "pseudo DBA" being able to manage databases also means that less expertise is required to operate a complex DBMS.

## 3.3 Threats to Validity

**Generalizability.** The interviewees may not be representative of all populations. The sample size and the particularity within a specific company, though providing a certain level of data saturation [66, 79], may not be large enough to ensure better generalizability

to other populations. Moreoever, selecting only experts and experienced DBMS users who are more interested in the topic than average may not be the best representation of all scenarios.

**Credibility.** A potential threat to our study's validity is the accuracy of participants' responses. Although we interviewed experts, their answers might still be influenced by external sources. To address this concern, we emphasized that the interview process was not intended to be judgmental and made a conscious effort to ask for specific details related to the participants' projects and profiles during the interviews.

**Confirmability.** A potential issue that may affect the accuracy of our study is the analysis, particularly the coding step of the interviews. To mitigate this concern and enhance the reliability of our conclusions, all the obtained results are carefully discussed by the authors and an expert in this domain.

## 3.4 Implications

Our study yielded several sets of implications for DBMS users, notably DBAs, researchers, and tool creators. Although the results may not be generalizable to all organizations due to our focus on one particular company, we believe that these findings provide a valuable knowledge base. They can enhance the understanding of DBMS user activities and the routines of DBMS administration automation, thereby facilitating the adoption of automation tools.

### 3.4.1 For DBMS users

DBMS users, including product owners, managers of teams that provide database service, DBMS experts, developers, DBAs, are the information source of our study. Here are a couple of implications retrieved for them:

- Many of our participants showed a deep understanding of DBMS administration automation, but only a small number of them are actively keeping up with the latest automation tools and instead sticking to their established routines. We believe that DBMS users should prioritize this activity by allocating more time to explore novel automation projects in the field.
- There are DBMS users who are reluctant to trust AI and ML-based tools, due to concerns about their reliability and stability. While it is true that these tools are still limited in their use cases, we encourage DBMS users to try them out not to miss out on potential automation opportunities.
- Based on our interviews, it appears that our participants are dissatisfied with the lack of expertise and experience in DBMS among many other DBAs and developers, which results in wasted time and effort in dealing with human errors. As a solution, we suggest that they invest more time in training those with less experience.

### 3.4.2 For researchers & tool creators

Our study revealed some of the difficulties encountered by DBMS users, and they present implications for interesting and challenging research opportunities. They give numerous guides and specifications for tool creators:

- Our participants have shown a preference for adopting open-source DBMS products. As a result, we suggest that researchers and tool creators focus more on studying these types of products, which would provide researchers with better access.

- Our study has uncovered a range of intriguing industrial challenges, such as capacity planning with limited data and operation planning with business constraints. Therefore, we encourage researchers to view these challenges as valuable opportunities for further investigation, taking into account the limitations under the industrial context.
- It is essential to strike a balance between the advantages and costs of automating DBMS administration. While automation can be computationally resource-intensive and require high-quality data, the costs of automation should not exceed the benefits it provides.
- Gaining the trust of DBMS users is a significant hurdle for DBMS administration automation tools. To overcome this obstacle, we recommend investing more effort into improving the transparency, applicability, and safety measures of the tools, to enhance their stability and reliability and, ultimately, earn the trust of DBMS users.
- Considering the sensitivity of DBMS, it is important to establish a suitable human-machine interface that grants DBMS users greater control over the process when required. This interface should also provide detailed information to avoid appearing like a black box and undermining users' expertise.
- DBMSs are closely related to applications, and the behavior of these applications is a significant contributor to DBMS anomalies. To address this issue, we recommend investing more effort into developing automation tools that are capable of handling the dynamic behavior of applications.

## 3.5 Conclusion

Building upon these insights, the next logical step is to translate user needs and practical constraints into concrete, adaptive solutions. In particular, one of the most prominent pain points revealed in our study is the challenge of tuning DBMS configurations in a way that balances effectiveness, transparency, and cost-efficiency. Addressing this challenge requires rethinking how tuning parameters—or knobs—are selected and optimized under workload-specific and resource-constrained conditions. In the following chapter, we introduce DOT, a dynamic tuning framework designed to address these issues by adaptively selecting and optimizing DBMS knobs with minimal overhead. DOT integrates machine learning, statistical reasoning, and benchmarking optimizations to provide an automated yet controllable tuning process that aligns with both user expectations and industrial realities.

# Chapter 4

# DOT: Dynamic Knob Selection & Online Sampling for Automated Database Tuning

Automated DBMS tuning hinges on the ability to identify and optimize the right configuration knobs without incurring prohibitive sampling costs. Yet, as shown in the previous chapter, existing knob-selection strategies are fragile: they require large amounts of data, show high variability across workloads, and often depend on static pre-selection that fails in dynamic settings.

This chapter introduces DOT, a framework designed to address these limitations. Instead of relying on an expensive offline ranking phase, DOT continuously refines the search space during tuning, adapting the number and identity of knobs as evidence accumulates. By combining statistical tests, feature elimination, and Bayesian optimization, DOT reduces overhead while preserving—or even improving—performance.

We begin by analyzing why traditional knob selection is costly and inconsistent, using experiments across OLTP and OLAP workloads to quantify sampling effort and variability. We then present the design of DOT, explaining how it integrates dynamic knob selection with adaptive benchmarking. Finally, we evaluate DOT against state-of-the-art methods, showing that it achieves competitive or superior performance while cutting tuning time.

## 4.1 Background and Challenges in DBMS Knob Selection and Tuning

In this section, we analyze different knob selection strategies discussed in Section 2.3 and present experimental evidence that underscores the inherent challenges of knob selection in DBMS tuning. These difficulties—such as lengthy search time, high overhead, and workload specificity—motivate the need for a dynamic, efficient, and cost-aware approach. DOT addresses this need by adopting a lightweight strategy that dispenses entirely with pre-training, thereby minimizing preliminary data collection and expert intervention. This design significantly enhances reproducibility and broadens the potential for industrial adoption.

Two OLTP workloads (Sysbench with 10 tables of 2 million rows and 50 concurrent threads, implementation from [49]; TPC-C at scale 100 with 32 threads, implementation from [101]) and one OLAP workload (TPC-H at scale 1, implementation from [102]) 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 [103] cloud platform.

### 4.1.1 Analysis of Knob Selection Strategies

First, we seek to obtain a deeper understanding of how different knob-selection strategies perform in practice, and how they compare against one another in terms of efficiency, accuracy, and practicality. While modern DBMS expose hundreds of tunable parameters, only a limited subset significantly impacts performance. We therefore focus on 22 knobs identified by prior expert studies [13], which capture the most influential levers while avoiding an intractably large search space. By evaluating six representative methods—Plackett & Burman (P&B), Lasso, CART, Sensitivity Analysis, an experience-based approach, and an LLM-based approach—we aim to characterize not only the quality of the resulting knob rankings but also the trade-offs each method entails. Some strategies rely on heavy empirical sampling, others on domain expertise, and others on learned or generative models; each comes with its own advantages and limitations. Through systematic experiments on two benchmarks, we assess where these methods align, where they diverge, and what their comparative strengths and weaknesses reveal about the broader problem of automated DBMS tuning. The choice of 22 knobs is motivated by prior work showing that, although modern DBMS expose hundreds of tunable parameters, only a small subset has substantial impact on performance [30].

The methods compared include: Plackett & Burman (P&B) [28], Lasso [29], CART [30], Sensitivity Analysis [27], an experience-based approach, and an LLM-based approach leveraging GPTuner's prompt [34] to query GPT-4o.

**Sample Collection.** Each method was applied following its canonical protocol. For P&B, we replicated the design of [28], collecting 48 samples per workload. For Lasso and CART, we followed Peduzzi et al. [35], requiring 220 samples each. Sensitivity Analysis was implemented by initializing all knobs to minimum, then varying each to maximum. Sensitivity Analysis required 22 samples. The experience-based approach relied on input from a senior DBA. Finally, the LLM-based method required no sampling at all—only issuing prompt queries—which highlights its low-overhead nature compared to empirical sampling.

**Knob Ranking and Selection.** All methods produced a ranking of the 22 knobs. To ensure comparability, we restricted evaluation to the top 5 knobs per method (Figure 4.1). This restriction ensures a fair, tractable comparison across methods: the search space grows exponentially with knob count, and five knobs represent a balance between experimental feasibility and capturing meaningful performance improvements. For the LLM-based approach, we modified GPTuner's original prompt to directly return the top 5 knobs (Listing 1).

**Exhaustive Grid Search.** We exhaustively explored the configuration space of the top 5 knobs selected by each method. Integer knobs were tested at five discrete values (default plus four evenly spaced values within the range), and boolean knobs at two states. This yielded up to $5^5 = 3{,}125$ configurations per knob set, requiring roughly 78 hours of benchmarking each. To keep the study feasible, we sampled each configuration once despite benchmarking noise, accepting higher variance as a necessary trade-off. We deliberately chose grid search over automated tuning algorithms: while computationally heavy, grid search removes the confounding effects of algorithmic heuristics and provides an unbiased ground truth for assessing knob selection quality.

**Benchmarks and Metrics.** We evaluated performance under two workloads:

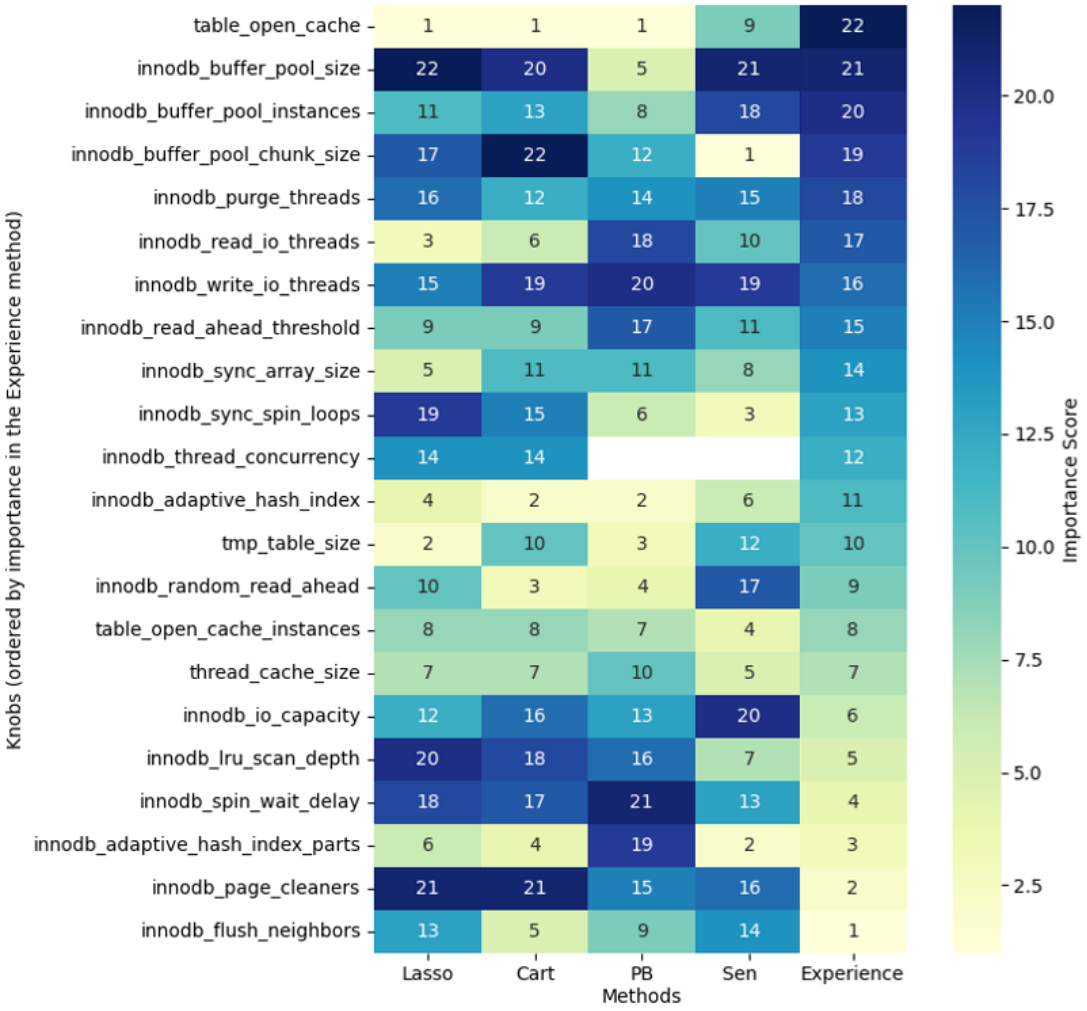


Figure 4.1: Knob importance under different methods for SYSBENCH

- SYSBENCH [49, 53, 54] OLTP: 90-second stress tests, reporting average throughput (TPS).

- TPC-H OLAP [55]: total execution time of queries.

Higher throughput (SYSBENCH) and shorter runtime (TPC-H) indicate better performance.

**Results.** Figures 4.1 and 4.2 show the knob rankings identified by each method, while Tables 4.1 and 4.2 report the achieved performance. Default performance was 636 TPS for SYSBENCH and 74 seconds for TPC-H. Due to overlaps among critical knobs selected by different methods, the total tuning effort required about 18,000 samples for SYSBENCH and 9,000 for TPC-H, or roughly 130 hours of runtime across 5 parallel machines. These results allow us to disentangle the effect of knob ranking strategies from that of tuning algorithms, offering a clean comparison of selection quality.

Based on the tuning results, we observe the following:

1. **Methods differ widely in effectiveness.** Plackett & Burman was the least effective, and experience-based rankings performed only slightly better due to their static nature. Sensitivity Analysis showed potential, but is still limited in performance due to its reliance on a single extreme baseline configuration. In contrast, Lasso, CART, and the LLM-based approach achieved more consistent results. Figures 4.1 and 4.2 illustrate knob rankings, while Tables 4.1 and 4.2 report the resulting performance.

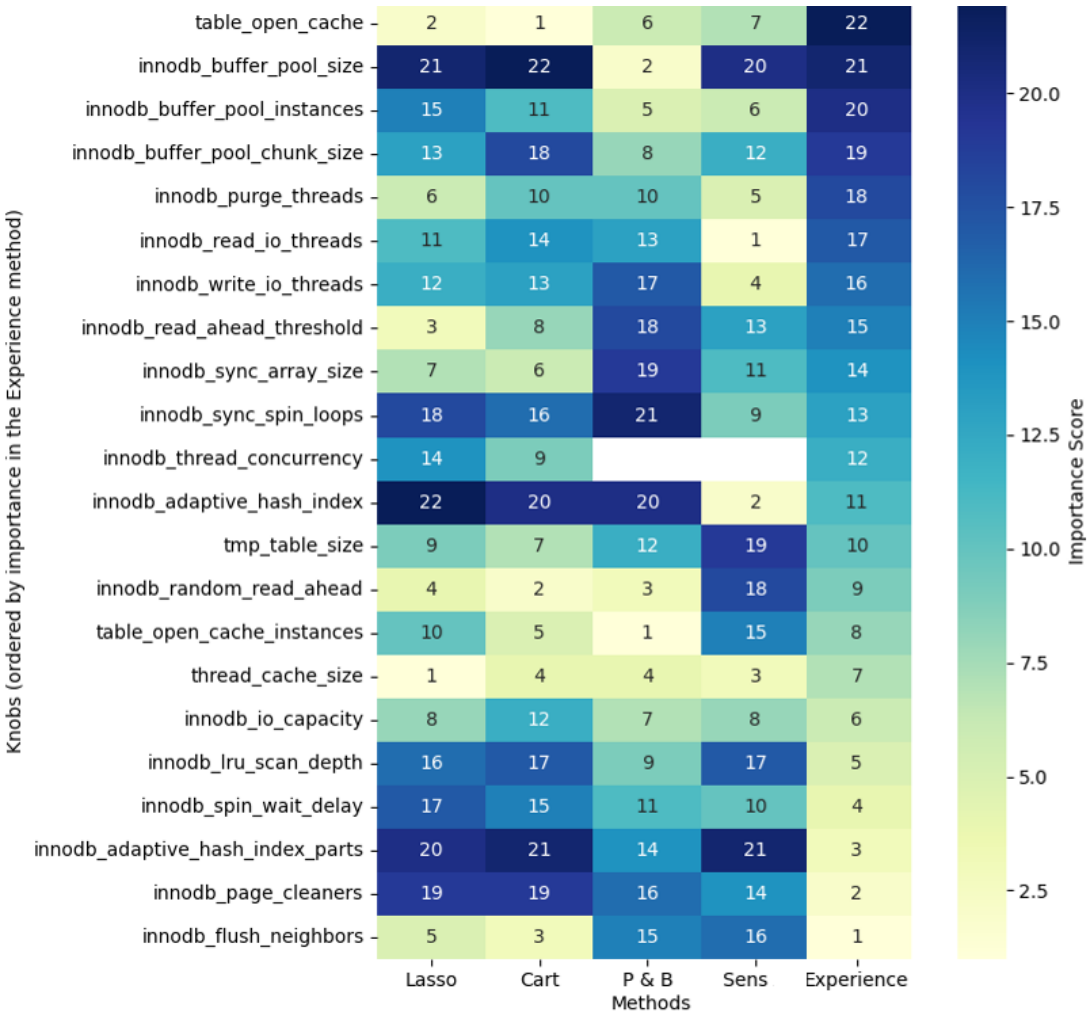


Figure 4.2: Knob importance under different methods for TPC-H

Table 4.1: SYSBENCH Tuning Results under TOP knobs (TPS)

| | TOP 1 | TOP 2 | TOP 3 | TOP 4 | TOP 5 |
|---|---|---|---|---|---|
| Sens. | 967.7 | 975.07 | 1303.4 | 1411.1 | 1411.1 |
| P & B | 709.4 | 926.8 | 926.8 | 926.8 | 939.1 |
| Lasso | 967.7 | 1236.0 | 1275.7 | 1331.4 | 1469.0 |
| CART | 709.4 | 768.8 | 1297.9 | 1415.5 | 1588.7 |
| Experience | 747.9 | 1120.1 | 1187.68 | 1238.0 | 1294.4 |

Table 4.2: TPC-H Tuning Results under TOP knobs (Total Exec. Time (s))

| | TOP 1 | TOP 2 | TOP 3 | TOP 4 | TOP 5 |
|---|---|---|---|---|---|
| Sens. | 47.85 | 47.15 | 46.59 | 46.22 | 46.22 |
| P & B | 77.38 | 76.11 | 75.51 | 74.72 | 74.35 |
| Lasso | 77.09 | 47.85 | 47.69 | 46.12 | 46.02 |
| CART | 47.85 | 47.69 | 47.69 | 46.12 | 45.76 |
| Experience | 78.15 | 46.61 | 46.25 | 46.06 | 45.72 |

2. **Tuning a few critical knobs is highly efficient.** For SYSBENCH, tuning only two knobs selected by Lasso reached 1236 TPS, nearly matching the performance of tuning five knobs from the experience-based set. For TPC-H, adjusting a single knob identified by Sensitivity Analysis or CART reduced execution time to 47.9s, close to the best result. This matters because tuning one knob requires only 5 samples, two knobs 25 samples, while five knobs require up to 3,125.

3. **Accurate knob selection yields large gains.** Choosing the right knobs improved SYSBENCH throughput by 10–20% compared to experience-based selection, and by up to 80% compared to the suboptimal P&B set. This highlights knob selection as a decisive factor in DBMS tuning effectiveness.

4. **Knob importance is workload-dependent.** The most impactful knobs differ significantly between OLTP (SYSBENCH) and OLAP (TPC-H). To confirm, we extended the study to the TPC-C workload [50]. As shown in Table 4.3, CART identified distinct sets of important knobs across workloads. The only consistently critical knob was `innodb_buffer_pool_size`, while all other top knobs varied.

These preliminary findings—derived from single exhaustive grid searches per configuration—demonstrate CART's relatively reliable knob-selection performance alongside its substantial sampling overhead, clearly illustrating the trade-off between sampling cost

Table 4.3: TOP knobs for TPC-H, TPC-C, and SYSBENCH

| Knob | TPC-H | TPC-C | SYSBENCH |
|---|---|---|---|
| innodb_adaptive_hash_index | ✓ | | |
| innodb_adaptive_hash_index_parts | ✓ | | |
| innodb_buffer_page_chunk_size | ✓ | | ✓ |
| innodb_buffer_page_cleaners | | ✓ | |
| innodb_buffer_pool_size | ✓ | ✓ | ✓ |
| innodb_lru_scan_depth | | | ✓ |
| innodb_page_cleaners | ✓ | | ✓ |
| innodb_spin_wait_delay | | ✓ | |
| innodb_sync_spin_loops | | ✓ | |
| innodb_write_io_threads | | ✓ | ✓ |

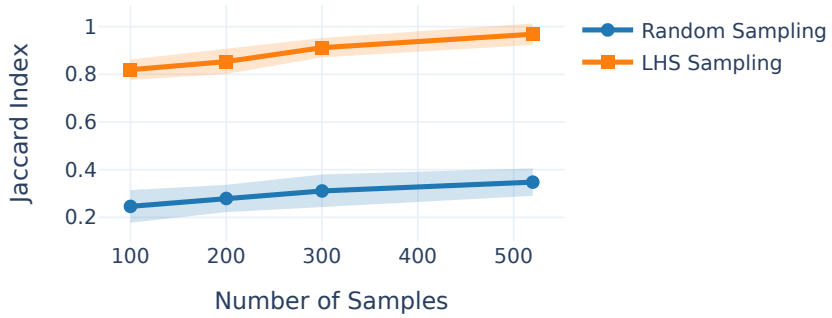


Figure 4.3: Variability of CART knob selection across varying sample sizes & sampling strategies

and performance gains. They also underscore that suboptimal knob choices directly impair tuning results. As initial guidance, these results highlight the broader challenge of balancing empirical sampling effort against tuning efficacy in automated DBMS configuration.

### 4.1.2 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 setting related knobs). **InnoDB** is MySQL's default transactional storage engine, responsible for handling disk I/O, caching, and concurrency control. Its configuration parameters—such as buffer pool size, log file size, and flush policy—have a direct and significant impact on throughput and latency. Focusing on these InnoDB-related knobs thus allows us to capture the dominant performance factors while keeping 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 [30]. 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 [30]. 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 [29] and Hunter [30]—against pure random sampling (as in CDBTune [13]). Figure 4.3 plots Jaccard indices for the Sysbench workload on MySQL when using 100, 200, 300, and 520 samples (520 aligns with ten samples per feature [35]). 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.

We then employ BO (implemented using Scikit Optimize [104]) 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 2.4, 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 4.4 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 4.4: 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 |

### 4.1.3 Workload-Sensitive Variations of Top Knobs

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, and TPC-H workloads. 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 4.5. 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 the 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.

Table 4.5: Jaccard index between knob sets.

| | SYSBENCH | TPC-C | TPC-H | EXP |
|---|---|---|---|---|
| SYSBENCH | — | $0.317 \pm 0.035$ | $0.325 \pm 0.033$ | $0.258 \pm 0.016$ |
| TPC-C | $0.317 \pm 0.035$ | — | $0.472 \pm 0.041$ | $0.243 \pm 0.029$ |
| TPC-H | $0.325 \pm 0.033$ | $0.472 \pm 0.041$ | — | $0.409 \pm 0.024$ |
| EXP | $0.258 \pm 0.016$ | $0.243 \pm 0.029$ | $0.409 \pm 0.024$ | — |

Table 4.6: SYSBENCH tuning with knob sets for different workloads ranked with CART (statistically significant at $p < 5\text{E-}2$)

| | **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) |

Yet we also uncovered a surprising degree of transferability. Table 4.6 compares SYSBENCH 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 ($1{,}725 \pm 31$ TPS), while the SYSBENCH-specific knobs converge slightly faster ($216.2 \pm 32.6$ min vs. $217.8 \pm 42.7$ min). However, their performance and tuning time are statistically equivalent according to a t-test—indicating that both represent strong, practical choices. Tuning the SYSBENCH knobs delivers excellent performance ($1{,}708 \pm 29$ 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 $1{,}653 \pm 27$ TPS and $1{,}639 \pm 35$ 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 ($1{,}677 \pm 35$ TPS). In contrast, random knob selection performs poorly, losing nearly 40% of the attainable throughput ($1{,}063 \pm 83$ TPS). For tuning time, the TPC-H knob set even outperforms others, reaching convergence in just $188.3 \pm 57.9$ minutes. However, restricting the search to shared knobs doubles the convergence time ($447.3 \pm 26.2$ min), indicating that omitting workload-specific knobs can hinder optimizer guidance. Random selection also increases tuning duration ($308.8 \pm 169.8$ 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:

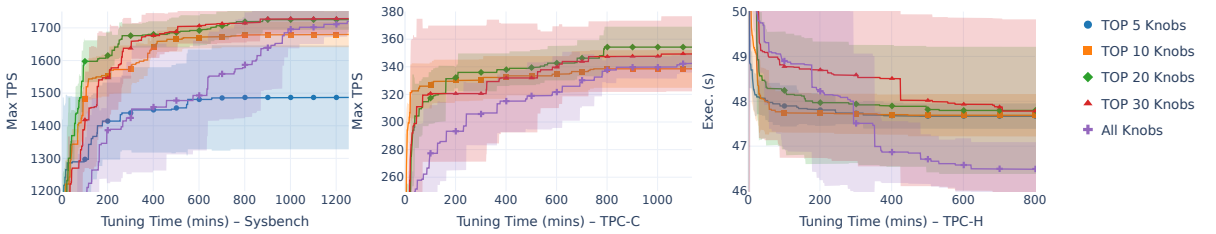


Figure 4.4: Tuning result under different numbers of knobs

- **Cross-workload reuse is viable**: top knobs from a workload can be reused in another with modest loss in throughput.
- **Common knobs matter, but are not sufficient**: restricting to universally important knobs retains much of the performance but incurs convergence delays.
- **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.

### 4.1.4 Optimal Number of Tuned Knobs

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 4.7 and figure 4.4 illustrate DBMS performance tuned with different numbers of Top knobs (expertise-based) using BO. For the SYSBENCH workload, tuning only the Top 5 knobs produces slightly lower performance than tuning the Top 10, 20, or 30 knobs (and all 52 knobs), but tuning five knobs does not achieve the fastest convergence time either. In fact, tuning the Top 20 knobs yields the best overall result: it achieves near-optimal performance (equivalent to tuning 30 knobs, both ranked at 2.0 in performance) while incurring the lowest tuning time (also ranked at 2.0). Although the p-value of the tuning time for SYSBENCH is higher than 0.05, indicating that most tuning times are statistically equivalent, we can still observe that tuning all 52 knobs requires substantially more time to reach the same performance level.

For the TPC-C workload, tuning only five knobs results in significantly worse performance compared to tuning larger knob sets; this difference is statistically significant. Here, tuning the Top 20 knobs still produces the best average performance (ranked 1.8). However, if minimizing tuning time is the priority, tuning the Top 10 knobs offers a better trade-off. As with SYSBENCH, tuning all knobs takes considerably longer to converge than tuning any smaller subset.

Under the TPC-H workload, tuning just five knobs achieves performance comparable to tuning on a larger subset of knobs, and the p-value is higher than 0.05, indicating no essential difference in the final performance. This suggests that TPC-H is relatively less sensitive to the knob set under tuning. Tuning all knobs yields the best absolute performance, but at the cost of a much longer tuning time. Therefore, selecting only the Top 5 knobs is preferable as we have comparable performance with minimal tuning overhead.

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.

### 4.1.5 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.

## 4.2 The DOT approach for dynamic knob selection

This section presents DOT, a dynamic framework for DBMS knob tuning that updates the set of tuned knobs on the fly (Figure 4.5). Instead of performing a costly, up-front selection, DOT interleaves optimization and selection: Bayesian Optimization (BO) searches the current subspace (Section 4.2.2); Recursive Feature Elimination with Cross-Validation (RFECV) identifies and retains the most influential knobs (Section 4.2.3); and a lightweight Likelihood-Ratio Test (LRT) decides whether to keep exploiting the reduced set or expand it with additional knobs (Section 4.2.4). To curb runtime overhead, we introduce adaptive benchmarking strategies tailored to OLTP and OLAP workloads (Section 4.2.5). Algorithm 1 summarizes the full loop and shows how prior samples and optional prior rankings are reused across knob-set updates. We conclude by contrasting DOT with two dynamic baselines—incremental expansion and statistical elimination—used later in our evaluation (Section 4.3).

### 4.2.1 Overview of the DOT approach

To address the above-mentioned practical challenges of knob selection in DBMS tuning, we present DOT—a dynamic and adaptive framework illustrated in Figure 4.5. 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. Given a knob list $\mathcal{K}$ (ranked or not), DOT starts tuning with the Top $k_0$ knobs using BO (Section 4.2.2). BO with surrogate model Gaussian Process (GP) 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 4.2.3). After the epoch completes, RFECV is applied to the collected samples to identify the most impactful knobs. RFECV iteratively trains a predictive model and

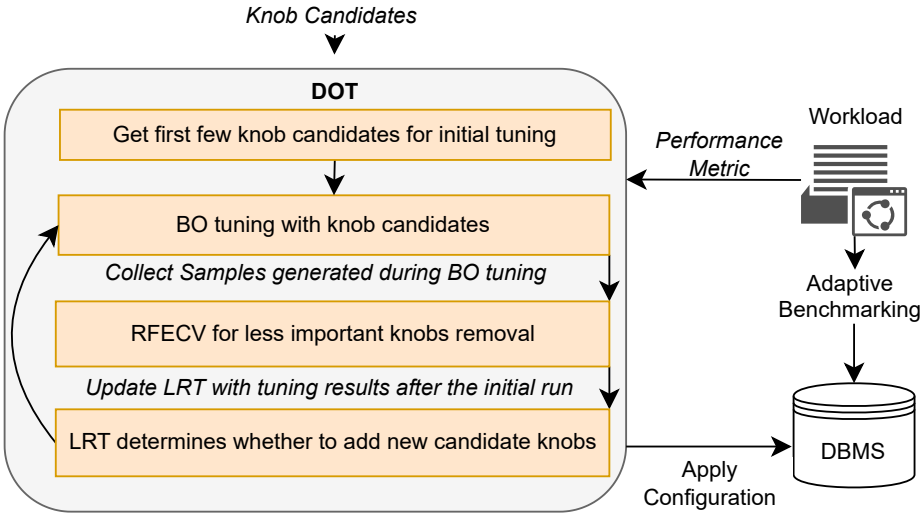


Figure 4.5: Overview of DOT

removes the least influential knobs based on their estimated importance, repeating this process with cross-validation to determine the optimal subset of knobs. Next, a likelihood-ratio test (LRT; Section 4.2.4) 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. 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 $\Delta k$ 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 maximiz-

ing 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.

### 4.2.2 Bayesian Optimization via Gaussian Process

BO with GP [105] surrogate is our tuning algorithm of choice for expensive, noisy objectives [106, 107, 108, 29, 27]. 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 4.2.5), 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.

### 4.2.3 Recursive Feature Elimination with Cross-Validation

RFECV is a widely adopted algorithm for feature selection in ML projects [109]. DOT uses it to prune tuning knobs before 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 [30]—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. 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.

### 4.2.4 Likelihood-Ratio Test for Set Expansion

In DOT, LRT [110] 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:

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

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

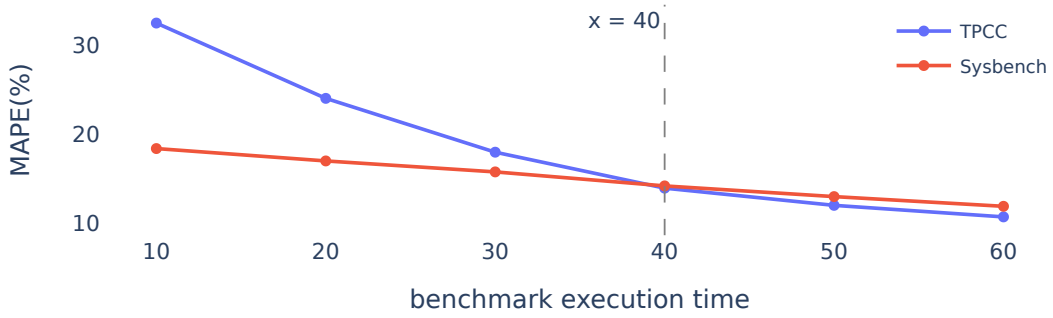


Figure 4.6: MAPE of Partial vs. Full Benchmark Results

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 in performance is strong, avoiding unnecessary expansion of the search space, which would incur additional computational overhead.

### 4.2.5 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 for $T_{\max} = 90\,\text{s}$, using only the final 30 s to compute the *ground-truth* TPS. This warm-up can consume 69.5% of the total tuning time. To mitigate this, we run an early-cut window of $T_{\text{cut}} = 40\,\text{s}$. If the average TPS at $T_{\text{cut}}$ 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 (Algorithm 3). Figure 4.6 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 $T_{\text{cut}} = 40\,\text{s}$, 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

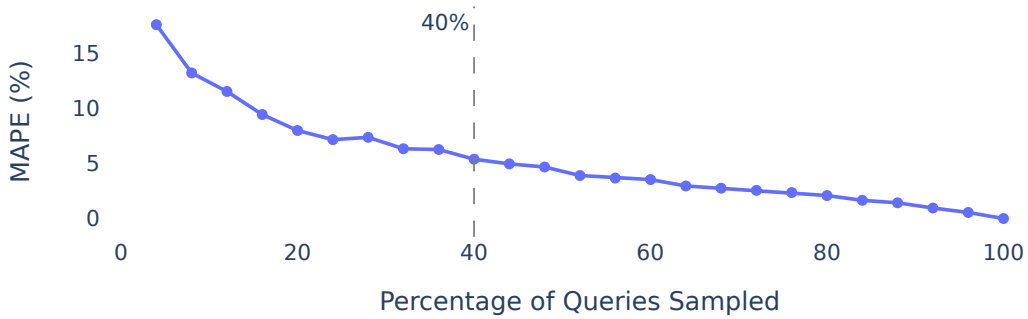


Figure 4.7: MAPE vs. Benchmark-Query Coverage

execution time of a set of analytical queries, without warm-up. Benchmarking accounts for 57% of tuning time. As shown in Algorithm 4, 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 total time, we run the full workload to obtain the execution time; otherwise, we log the partial result as a proxy. Figure 4.6 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 ($T_{\text{cut}} = 40$s and 40% target query coverage) using historical data, it can be estimated from a few pilot measurements at minimal cost—Section 4.3.8 shows that performance is insensitive to the exact budget chosen.

## 4.3 Experimental Results

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

### 4.3.1 Baseline Algorithms

**Incremental Knob Tuning.** Our first baseline adopts the progressive–expansion heuristic of OtterTune [29]. Starting with the top four most influential knobs, we let BO explore that sub-space for one epoch of 25 iterations. At the end of each epoch, we append the next two top-ranked knobs, keeping all previously collected observations so that the surrogate model is never reset. The loop stops when we have exhausted the evaluation budget. This scheme keeps early search low-dimensional, but it inherits a risk: if the initial ranking misses a truly important knob, BO may spend many iterations stuck near a sub-optimal configuration.

**Statistical Elimination.** Incremental tuning fights the curse of dimensionality by growing the search space; *Statistical Elimination* (SE) attacks it from the opposite direction. We start with all knobs, run an initial $2 \times N$ BO evaluations (where $N$ is the number of knobs), then perform a one-shot hypothesis test (Welch's two-sided $t$-test, $\alpha = 0.05$) for each knob. For numeric knobs, observations are split by median; for categorical knobs, by value. Any knob with $p > \alpha$ is deemed uninfluential, frozen at its current best setting, and removed from the search space. BO then continues on the surviving set until the budget is exhausted. By discarding inert knobs early, SE reduces dimensionality up front, yielding faster surrogate fits and more informative acquisitions.

### 4.3.2 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 4.1. 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 [27] and OtterTune [29] 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 (SE)**: Our two dynamic-search-space variants;

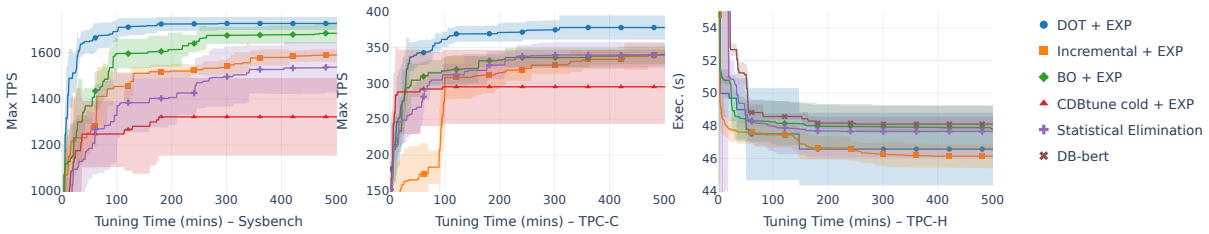


Figure 4.8: Performance vs. tuning-time of different algorithms on SYSBENCH, TPC-C, TPC-H with knob importance knowledge

- **CDBTune**: Author's public implementations of CDBTUNE [111], the implementation currently only supports OLTP;
- **DB-Bert**: Author's public implementations of DB-Bert [112],the implementation currently only supports OLAP;
- **LLM knowledge based**: An LLM-based baseline adapted from GPTUNER [34], where we modified the prompt of GPTUNER's knob selection phase to let CHATGPT decide both which knobs and how many to tune.

For the above methods, the suffix -RAN indicates that the experiment was conducted without prior knowledge, using a randomly ordered knob list, whereas the suffix -EXP denotes that the experiment was conducted with prior knowledge based on human expertise, i.e., a ranked knob list. Other methods, like HUNTER, DDPG++(optimized version of CDBTUNE proposed in [113]), and UDO, are not included in our comparison: HUNTER, DDPG++ lacks a public implementation, while UDO focuses on index tuning outside our scope.

Section 4.1 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:

1. **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.

2. **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.

### 4.3.3 Including Knob Importance Knowledge

Figure 4.8 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

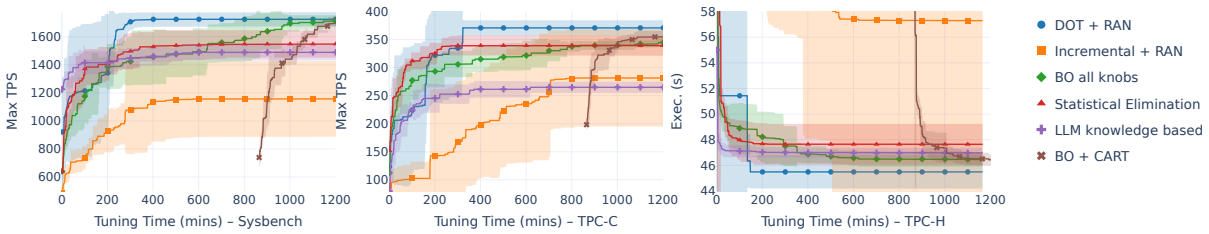


Figure 4.9: Performance vs. tuning-time of different algorithms on SYSBENCH, TPC-C, TPC-H without knob importance knowledge

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.

### 4.3.4 Excluding Knob Importance Knowledge

Figure 4.9 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+FULL 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-full eventually plateaus alongside DOT, but only after a lengthy exhaustive search. BO+full achieves similar performance as DOT yet takes a longer time. Note that Incremental+EXP follows a similar staircase pattern with a lower ceiling than 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 4.3.6).

### 4.3.5 Comparison of All Methods & Situations

Table 4.8 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.

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.

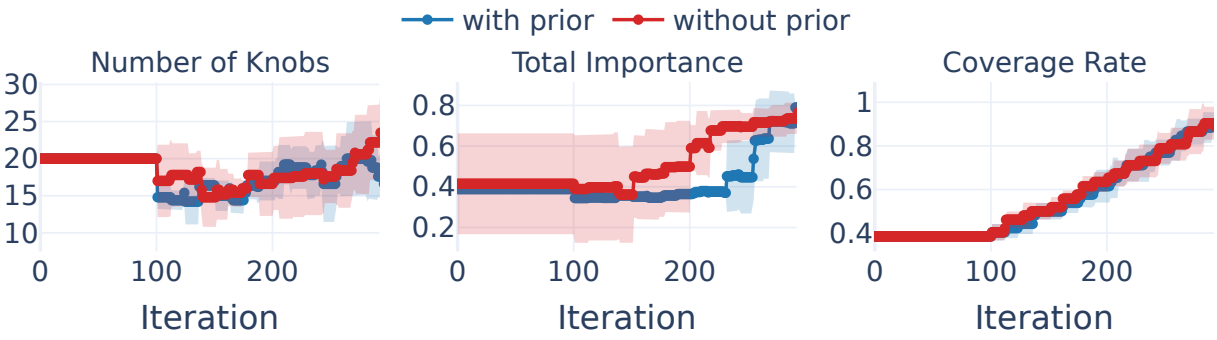


Figure 4.10: DOT behavior on SYSBENCH workload with and without knob importance prior

State-of-the-art methods, like CDBTUNE, DB-Bert, and LLM-based approaches, perform poorly here because we deliberately simulate a low-knowledge scenario. CDBTUNE requires an extended warm-up phase to gather data, DB-Bert needs more complete documentation and well-tuned hyperparameters, and LLM knowledge-based methods would perform better with more exchanges with the LLM model and complete documents. Under our "cold start" conditions, none can leverage that prior information, so their performance lags.

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.

### 4.3.6 DOT Behavior Analysis

Figure 4.10 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

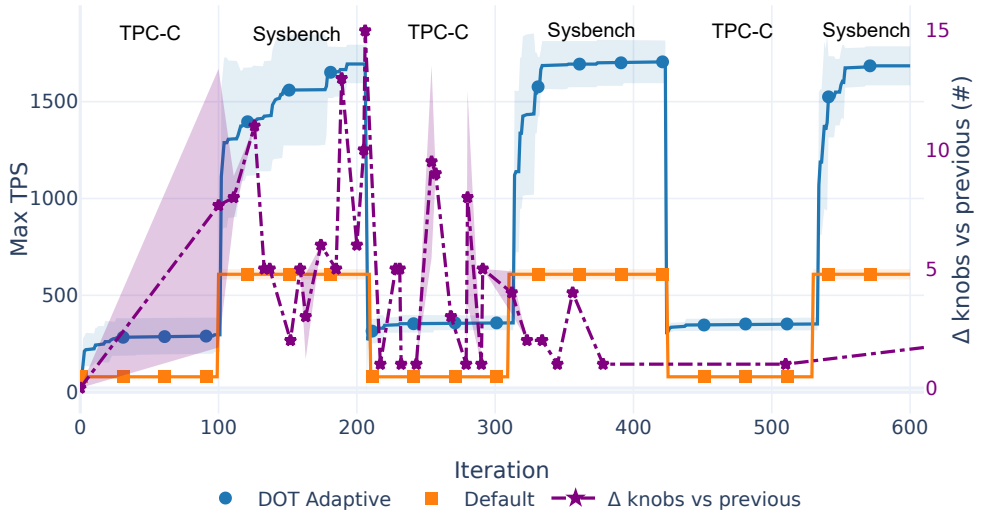


Figure 4.11: Tuning Performance of DOT on periodic workloads composed of TPC-C and Sysbench

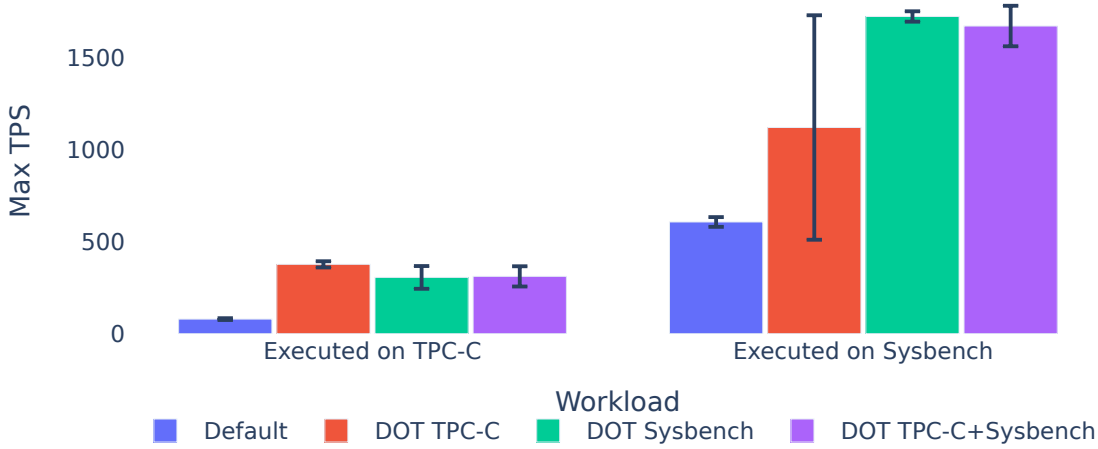


Figure 4.12: Max TPS Across Configurations and Workloads

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 achieves its design goals—dynamically balancing exploration and exploitation, maintaining a compact search space, and minimizing tuning overheads—while retaining the most important knobs even without pre-existing knob-importance priors.

### 4.3.7 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 [?]. Figure 4.11 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 workloads, which eventually stabilizes, indicating that DOT adapts effectively while converging toward a stable, efficient knob set.

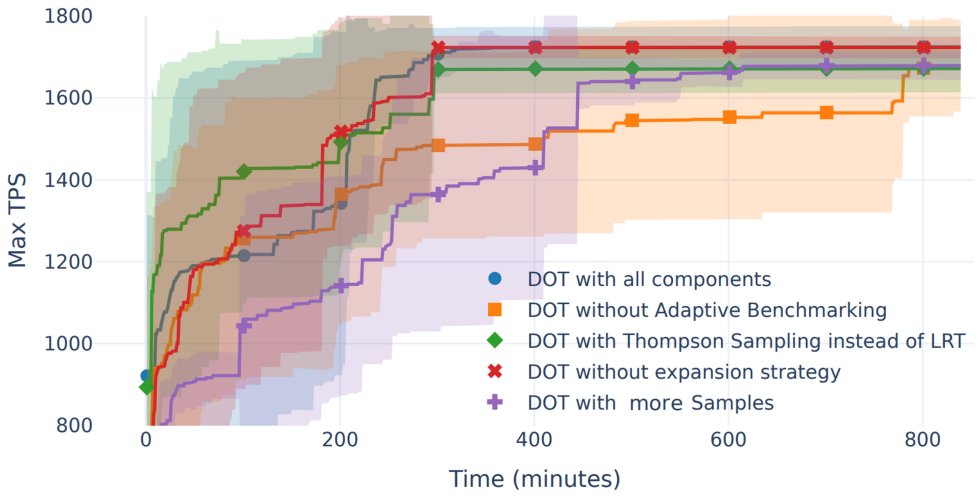


Figure 4.13: Tuning Performance of DOT on SYSBENCH for various configurations without knob importance knowledge

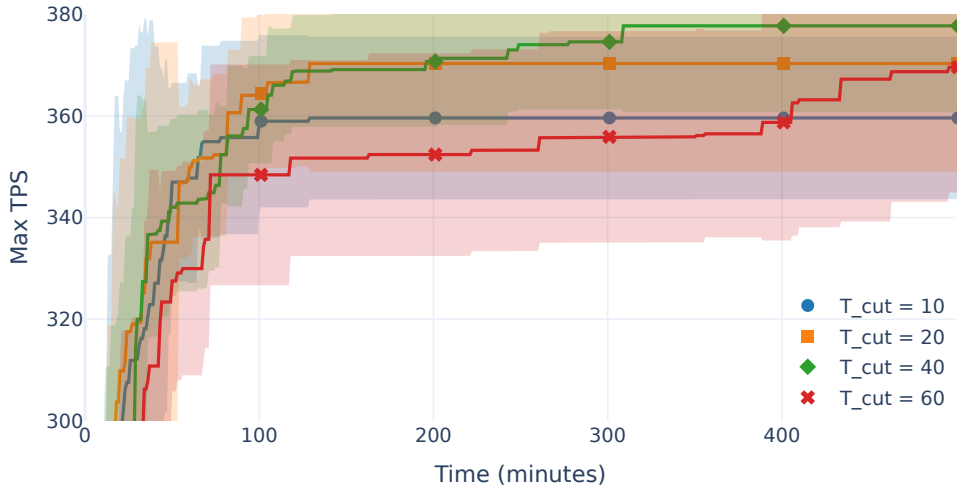


Figure 4.14: Tuning Performance of DOT on TPC-C for various $T_{\text{cut}}$ budget with knob importance knowledge

Figure 4.12 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.

### 4.3.8 Ablation Study

In this section, we evaluate the performance of DOT on SYSBENCH by ablating its key components—without any prior knob-importance knowledge (Figure 4.13):

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.

2. **DOT with Thompson sampling / without expansion strategy:** Replacing our LRT-based knob set expansion strategy with Binary Thompson Sampling [114],

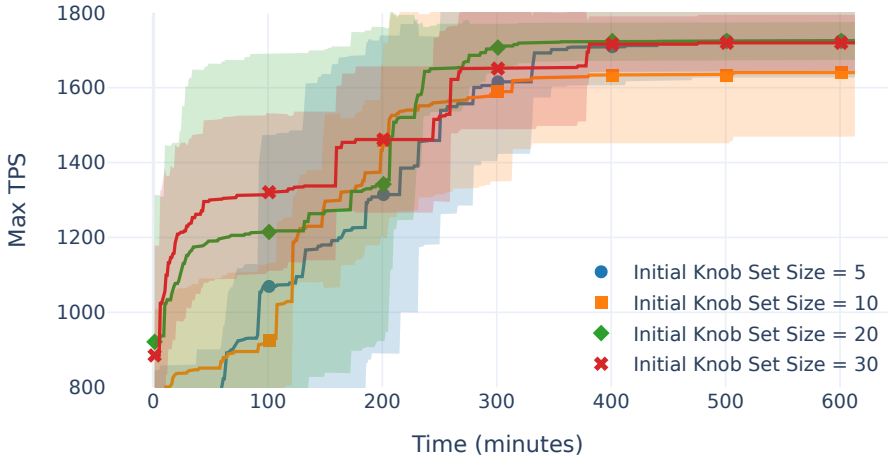


Figure 4.15: Tuning Performance of DOT on SYSBENCH for various $k_0$ without knob importance knowledge

or eliminating the 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 4.3.6). 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 number of 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 4.15 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.

We also evaluated DOT on TPC-C (which had the highest MAPE in Section 4.2.5) using prior knob-importance knowledge and varying the cutoff budget $T_{\text{cut}}$. Figure 4.14 shows that larger $T_{\text{cut}}$ 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 (Algorithms 3 and 4) concentrates precise measurements on promising configurations and uses 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 $T_{\text{cut}}$.

To summarize, these components collectively provide a robust, general-purpose solution for DBMS tuning.

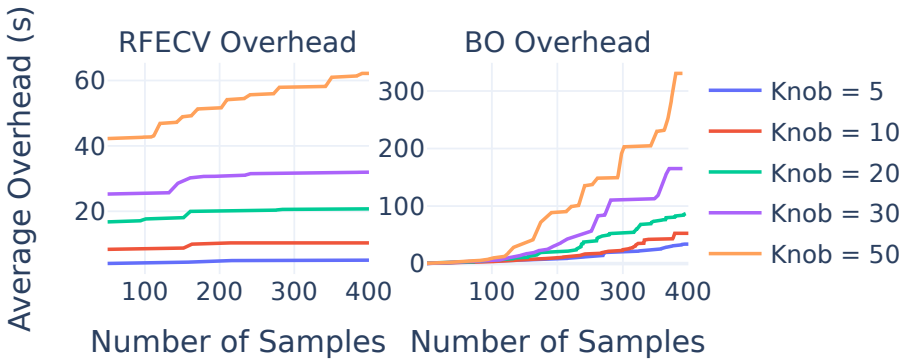


Figure 4.16: Bayesian optimization & RFECV overhead

### 4.3.9 System Overhead

The BO optimization algorithm 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 4.16). 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, and has a relatively low overhead.

Table 4.9 presents the runtime breakdown for DOT. Across all six scenarios, Benchmarking dominates the wall-clock budget, accounting for roughly 50 % to 65 %. Bayesian-optimization bookkeeping (*BO*) is the second-largest component but varies widely: it is modest for TPC-C (accurate EXP) at about 19 %, yet rises to almost 31 % for TPC-H (inaccurate EXP), reflecting the extra cost of longer or more exploratory searches. The Other bucket—comprising logging, LRT updating and inference, configuration deployment, and DBMS restarts—remains fairly stable at 13 % to 23 %. Because LRT is lightweight, its training and inference costs are negligible and therefore included in Other. RFECV pruning contributes less than 0.5 % in all runs. Taken together, the supplementary components (LRT and RFECV) are rather lightweight during the process of tuning, making them more adaptable for wider adoption.

## 4.4 Conclusion

While DOT addresses the challenge of tuning DBMS configurations by dynamically selecting and optimizing influential knobs, it primarily focuses on improving overall performance under fixed resource allocations. However, as highlighted by our industrial observations, resource provisioning itself—especially memory allocation—remains a critical and often inefficient aspect of DBMS administration. In particular, memory is frequently over-provisioned to hedge against unpredictable workload behaviors, resulting in significant waste. To tackle this complementary but equally pressing problem, the next chapter presents *MicroTune*, a lightweight reinforcement learning framework that dynamically adjusts memory buffers online. By shifting the focus from static performance tuning to adaptive resource management, *MicroTune* seeks to close the loop between DBMS performance and system resource efficiency.

```
"""
    You are an experienced DBA and your will determine which knobs are worth tuning.
    You only tune knobs that have a significant impact on DBMS performance and the target DBMS is MySQL.
    Given the following candidate knobs, score the importance for each knob between 0 and 1,
    with a higher value indicating that it is more likely to impact MySQL performance significantly for the given workload.

    The workload under tuning is a classical OLAP workload TPCH.
    Candidate knobs: {
        "table_open_cache",
        "innodb_buffer_pool_size",
        "innodb_buffer_pool_instances",
        "innodb_buffer_pool_chunk_size",
        "innodb_purge_threads",
        "innodb_read_io_threads",
        "innodb_write_io_threads",
        "innodb_read_ahead_threshold",
        "innodb_sync_array_size",
        "innodb_sync_spin_loops",
        "innodb_thread_concurrency",
        "innodb_adaptive_hash_index",
        "tmp_table_size",
        "innodb_random_read_ahead",
        "table_open_cache_instances",
        "thread_cache_size",
        "innodb_io_capacity",
        "innodb_lru_scan_depth",
        "innodb_spin_wait_delay",
        "innodb_adaptive_hash_index_parts",
        "innodb_page_cleaners",
        "innodb_flush_neighbors"
    }
    DBMS: MySQL;

    Now let us think step by step and give me your scoring of all the candidate knobs in json format:
    {
        "knob_name": {score}    // fill "score" with a number between 0 and 1
    }

    If no knobs are suggested, just fill "knob_list" with "None" and also return result in json format.
    Next, to optimize DBMS performance efficiently, we'll focus solely on the most impactful knobs-
    shrinking the search space to accelerate tuning and cut computational overhead-while preserving
    the final overall performance. How many knobs should we include to achieve this?"""
```

Listing 1: Prompt for Knob Ranking

Table 4.7: 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)<br>Test stat: 13.3<br>p-value: 1E-2 | Rank (tuning time [mins])<br>Test stat: 9.1<br>p-value: 5.8E-2 | Rank (Max TPS)<br>Test stat: 12.0<br>p-value: 1.7E-2 | Rank (tuning time [mins])<br>Test stat: 5.1<br>p-value: 2.8E-1 | Rank (Exec (s))<br>Test stat: 6.1<br>p-value: 1.9E-1 | Rank (tuning time [mins])<br>Test stat: 11.8<br>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) |

**Algorithm 1** DOT: Dynamic Knob Tuning

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

**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 $a_t \sim \text{Bernoulli}(0.5)$
    *init* $\leftarrow$ False
    **return** $a_t$
  **end if**
  $\hat{p}_0 \leftarrow \dfrac{s_0}{s_0 + f_0}, \quad \hat{p}_1 \leftarrow \dfrac{s_1}{s_1 + f_1}$
  $\Lambda \leftarrow \ln\!\left(\hat{p}_1 / \hat{p}_0\right)$
  $a_t \leftarrow \mathbb{1}[\Lambda > 0]$
  $g_t \leftarrow \dfrac{y^* - b}{b\,E}, \quad r_t \leftarrow \mathbb{1}[g_t > \delta]$
  $s_{a_t} \leftarrow s_{a_t} + r_t, \quad f_{a_t} \leftarrow f_{a_t} + (1 - r_t)$
  **return** $a_t$
**end procedure**

**Algorithm 3** Adaptive Benchmarking (OLTP)

**Require:** Workload $w$, configuration $c$; incumbent TPS $t_{\text{best}}$
**Parameter:** Early-cut window $T_{\text{cut}} = 40\,\text{s}$, full run $T_{\max} = 90\,\text{s}$
**procedure** ADAPTIVE BENCHMARKING($Q$, $t_{\text{best}}$)
    Run $w$ on $c$ for $T_{\text{cut}}$ seconds; record mean TPS $\hat{t}$
    **if** $\hat{t} > t_{\text{best}}$ **then** ▷ Promising candidate
        Continue the run until $T_{\max}$ seconds
        $t \leftarrow$ mean TPS over the last 30 s
    **else** ▷ Unpromising—save budget
        $t \leftarrow \hat{t}$
    **end if**
    **return** $t$ ▷ Performance estimate for $(w, c)$
**end procedure**

**Algorithm 4** Adaptive Benchmarking (OLAP)

**Require:** Query set $Q$, configuration $c$; best runtime $r_{\text{best}}$
**Parameter:** Sample size $k = \lfloor 0.4\,|Q| \rfloor$
**procedure** ADAPTIVE BENCHMARKING($Q$, $r_{\text{best}}$)
    Randomly pick $S \subset Q$, $|S| = k$
    $\hat{r} \leftarrow \sum_{q \in S} \text{time}(q, c)$ ▷ Run sampled queries
    $\tilde{r} \leftarrow \hat{r} \cdot \frac{|Q|}{k}$ ▷ Extrapolate
    **if** $\tilde{r} < r_{\text{best}}$ **then** ▷ Promising candidate
        Run $Q \setminus S$ and set $r \leftarrow \hat{r} + \sum_{q \in Q \setminus S} \text{time}(q, c)$
    **else** ▷ Unpromising—save budget
        $r \leftarrow \tilde{r}$
    **end if**
    **return** $r$ ▷ Performance estimate for $(k, c)$
**end procedure**

Table 4.8: 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)<br>Test stat: 37.4<br>p-value: 2.2E-05 | Rank (tuning time [mins])<br>Test stat: 32.2<br>p-value: 1.9E-04 | Rank (Max TPS)<br>Test stat: 31.4<br>p-value: 2.5e-04 | Rank (tuning time [mins])<br>Test stat: 37.9<br>p-value: 1.7E-05 | Rank (exec [s])<br>Test stat: 18.382<br>p-value: 3.1e-02 | Rank (tuning time [mins])<br>Test stat: 38.4<br>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 full | 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) |

Table 4.9: Runtime breakdown for DOT

| **Workload** | **EXP** | **Benchmarking** (%) | **BO** (%) | **RFECV** (%) | **Other** (%) |
|---|---|---|---|---|---|
| SYSBENCH | accurate | 58.75 | 26.79 | 0.40 | 14.07 |
| SYSBENCH | inaccurate | 54.56 | 32.12 | 0.40 | 12.92 |
| TPC-C | accurate | 65.22 | 18.99 | 0.27 | 15.51 |
| TPC-C | inaccurate | 58.27 | 27.57 | 0.33 | 13.83 |
| TPC-H | accurate | 50.96 | 25.86 | 0.44 | 22.74 |
| TPC-H | inaccurate | 50.34 | 30.91 | 0.46 | 18.29 |

# Chapter 5

# MicroTune: RL-based DBMS Buffer Pool Auto-Tuning for Optimal and Reduced Memory Utilization

As highlighted in the previous chapters, one challenge in DBMS resource management is the over-allocation of memory. To avoid SLA violations, administrators and static tuners often assign excessive buffer pool sizes, which wastes resources and increases operational costs. However, static configurations cannot react when workloads shift, leading either to inefficient over-provisioning or to performance degradation under sudden load changes.

Reinforcement learning offers a promising way to address this challenge because it treats DBMS tuning as a sequential decision-making problem. By continuously observing workload feedback, an RL agent can adjust configurations online, striking a balance between SLA compliance and efficient resource utilization. Unlike heuristic or rule-based methods, RL adapts its policy over time and can discover strategies that generalize across diverse workload dynamics.

In this section, we first introduce various RL algorithms most relevant to our setting. We then discuss critical design choices that strongly affect RL performance, including hyperparameter tuning, reward shaping, and training efficiency. Finally, we evaluate these algorithms both in vitro on benchmarked workloads and in vivo on a live database under varying conditions, highlighting their relative strengths and weaknesses.

This structure allows us to systematically compare RL approaches, understand the trade-offs in their design, and assess their practical viability for online DBMS tuning under realistic memory management constraints.

## 5.1 Context

DBMS like MySQL or MariaDB are composed of several memory buffers that occupy the main DRAM allocated to the database. Among these buffers, the InnoDB Buffer Pool presents the largest portion of the available DRAM [3]. It is generally recommended to allocate up to 80% of the total DRAM for the InnoDB Buffer Pool to optimize database performance according to DBMS publishers [115]. Hence, in this chapter, we focus on reducing the InnoDB Buffer Pool size.

Prior work has largely focused on one-time tuning rather than dynamic online adaptation. Search-based tuners (e.g., BestConfig [26]) sample knob configurations but incur high overhead; Bayesian optimization tuners (e.g., OtterTune [29]) leverage ML but depend on extensive training data; and *Reinforcement Learning* (RL)-based tuners (e.g., UDO [25], CDBTune [13]) explore the configuration space adaptively, yet remain limited to static settings. Rule-based methods such as iBTune [3] target buffer minimization un-

der miss-ratio constraints, while ResTune [38] broadens scope to CPU, RAM, and I/O but requires replayed workloads and slow convergence. Other studies [23, 116, 117, 118] focus on internal RAM redistribution without changing the DBMS's total allocation.

Cloud-scale systems often mitigate over-allocation through forecasting-based provisioning (e.g., Redshift [119], Intelligent Pooling [120], Oracle Cloud Infrastructure [121]), supported by proprietary infrastructures and workload traces. While enterprise-grade cloud databases include built-in resource auto-adjustment, our focus is on free open-source DBMSs, where such mechanisms are absent. MicroTune addresses this gap by introducing a lightweight RL-based agent for online buffer tuning, motivated both by vendor lock-in risks and eventual cost considerations [37].

Building on these insights, we propose MicroTune, an online tuner that dynamically adjusts RAM allocation in real time while satisfying SLA requirements. MicroTune leverages multiple RL algorithms. The use of an RL algorithm allows adjusting RAM to changes of the workload (the variation of the query type, number of connections, database size, etc.) by introducing a dependence between the observed context, which captures information about DBMS and workload, and the right decision for increasing or decreasing the needed RAM. This approach enables MicroTune to efficiently tune system parameters in an automatic, continuous manner, which optimizes DBMS resource utilization, reduces TCO, and alleviates users from the burden of fine-tuning.

To highlight the importance of buffer pool size optimization in DBMS, we analyzed metrics from 10 real production servers over three months. Using the "buffer_pool_pages_free" metric, which indicates the number of unused buffer pages, along with the "innodb_buffer_pool_size" configuration, we estimated the buffer over-allocation. This was calculated by multiplying the "innodb_buffer _pool_size" by the minimum "buffer_pool_pages_free" value observed during the three months.

Table 5.1 shows the potential DRAM savings for each database. The "Min Free ratio" is the minimum "buffer_pool_pages_free" ratio that we observed during the past 3 months, indicating the estimated over-allocation is based on the minimum DRAM space unutilized during the past 3 months. For several databases (e.g., 0, 1, 4, 6, 7), the buffer pool was fully utilized at least once, while others showed limited usage, indicating a significant opportunity for optimization. Even when "buffer_pool_pages_free" is at 0, some pages may still be dirty and unprocessed, suggesting that reducing buffer size is possible without impacting performance (i.e., latency, throughput). Thus, our estimated savings are likely conservative.

Table 5.1: Buffer Pool Usage and Estimated Overallocation

| DBMS ID | Buffer Pool Size | Over-allocation Ratio | Estimated Over-allocation |
|---|---|---|---|
| 0 | 10,240 MB | 0.00% | 0.11 MB |
| 1 | 225 MB | 1.92% | 4.33 MB |
| 2 | 12,288 MB | 79.74% | 9797.97 MB |
| 3 | 1,024 MB | 1.55% | 15.91 MB |
| 4 | 9,216 MB | 0.00% | 0.00 MB |
| 5 | 4,096 MB | 1.18% | 48.47 MB |
| 6 | 18,432 MB | 0.00% | 0.00 MB |
| 7 | 8,192 MB | 0.71% | 58.00 MB |
| 8 | 225 MB | 93.99% | 211.47 MB |
| 9 | 225 MB | 63.85% | 143.66 MB |
| **Total** | **64,163 MB** | — | **10,279.92 MB (16.02%)** |

Figure 5.1 illustrates the DRAM over-allocation for all 10 database instances mentioned in Table 5.1 under a dynamic perspective, meaning we look at the "buffer_pool_pages_free" metric and analyze the estimated DRAM over-allocation percentage of these database instances at different timestamps. Under this scenario, by continuously monitoring the

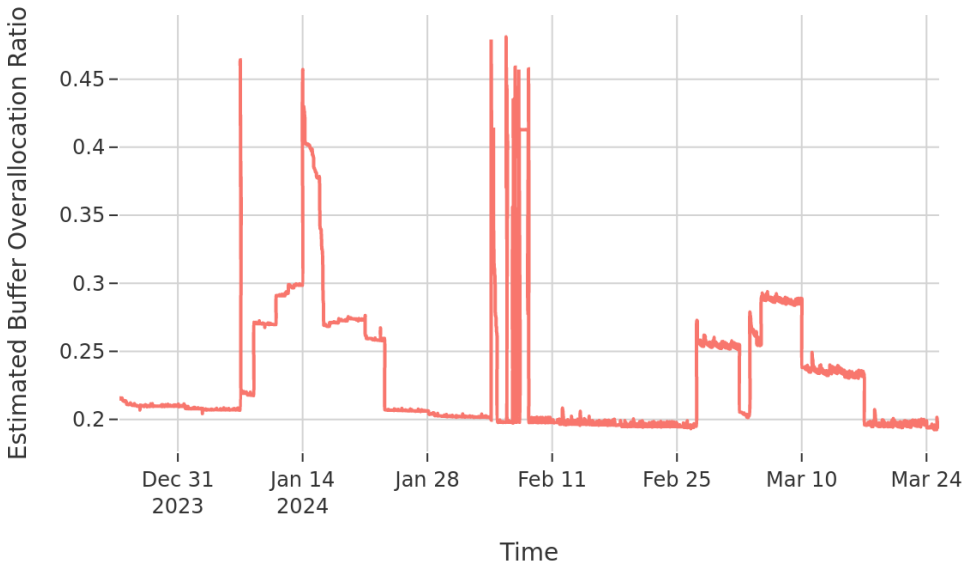


Figure 5.1: Estimated Buffer Over-allocation Ratio over Time

metric "buffer_pool_pages_free", the potential DRAM savings could range from 20% to 48%.

## 5.2 Objectives

Based on the sample analysis in the previous section, we can conclude that although a static one-time tuning can already be beneficial for the system, there is still room to better optimize the resource utilization of the DBMS. Production DBMS involves more complex SLA constraints—such as recovery time and point objectives—but here we adopt a simplified view by considering only performance-related SLA constraints. To optimize resource utilization effectively, our goal is to design a system that monitors DBMS status in real time and dynamically adjusts its buffer size based on the system's condition. In the meantime, it is important to respect the SLA constraints to not compromise the DBMS's performance. Since throughput is linked to workload and does not always reflect DBMS performance- e.g., a low number of queries results in low throughput, but this does not necessarily indicate poor performance—we use latency (the average time to process queries) as the primary performance indicator.

For MicroTune, we primarily focus on Online transaction processing (OLTP) workloads, and the principal objectives for MicroTune to achieve are the following: **1) Dynamic Buffer Size Tuning,** develop an agent capable of adjusting the buffer size of the database dynamically. The tuning process should be continuous, responding in real time to changes in the database load. **2) Resource Efficiency,** ensure the agent operates with a high degree of memory efficiency. It should minimize the buffer cache size without compromising the performance (i.e., latency), aiming to conserve as much DRAM as possible. **3) SLA Compliance,** maintain adherence to a predefined SLA by ensuring the latency of database queries does not exceed a specified threshold. The agent should make buffer size adjustments based on this latency limit, either increasing, decreasing, or maintaining the buffer size as needed. **4) SLA-Based Operation Without Direct Latency Measurement,** operate the agent under conditions where direct latency measurements are unavailable, since in real-world scenarios, the end user's latency is not accessible from the DBMS side.

These objectives aim to create a robust, efficient, and self-regulating system that ensures optimal performance and resource utilization of the MariaDB database under varying loads, all while adhering strictly to operational constraints set by the SLA.

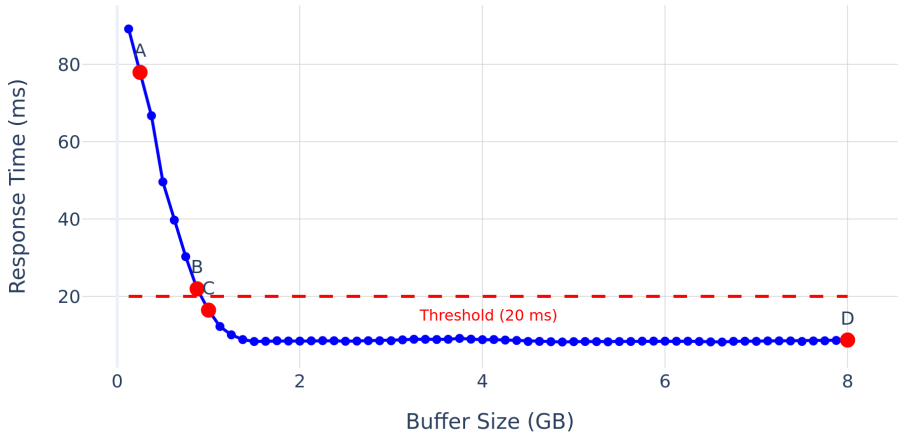


Figure 5.2: Demonstration Buffer Size vs. Mean Latency

**Example Showcase** To better illustrate the functionalities of MicroTune, an example is presented in this section. Figure 5.2 illustrates mean query latency as a function of buffer size for a MariaDB workload. As the buffer grows, latency steadily declines—but a slightly undersized pool provokes a dramatic latency spike, whereas modest over-provisioning delivers only marginal improvement. This asymmetry underscores the potential resource we can achieve with precise resource tuning without compromising performance. For this example, consider an SLA threshold of 20 ms. The graph indicates four key points:

- **Point A**: A latency much higher than the SLA threshold.
- **Point B**: A latency slightly above 20 ms.
- **Point C**: A latency slightly below 20 ms.
- **Point D**: A relatively large buffer size with latency well below 20 ms.

The goal of MicroTune is to minimize the buffer size while adhering to the SLA constraints on performance usually defined by latency and throughput objectives. For instance, if the current status of the DBMS is at Point A, MicroTune recognizes that the buffer size is insufficient and dynamically increases it until the latency satisfies the SLA constraints, stabilizing at Point C—the optimal buffer size for this workload and DBMS configuration. Conversely, if the current status is at Point D, while the SLA constraints are met, MicroTune identifies that the buffer size is unnecessarily large and reduces it to Point C, ensuring optimal resource utilization. Importantly, MicroTune avoids reducing the buffer size below Point C to prevent violating the SLA, as would occur at Point B.

This example demonstrates a typical use case for MicroTune. However, it is important to note that workloads often vary over time. As such, MicroTune must be capable of adapting to dynamic and diverse workloads to remain effective in real-world scenarios.

## 5.3 RL-based Online DBMS Memory Auto-Tuning

**Problem statement** Our objective is to find and set the smallest buffer size that satisfies SLA of DBMS under varying workloads. Here, the SLA is expressed in terms of response latency. Let $l_t \in \mathbb{R}_+$ be the observed latency at time $t$, and $l^* \in \mathbb{R}_+$ be the maximum latency corresponding to SLA. Let $w_t \in \mathcal{W}$ be the observed workload at time $t$, and $\mathcal{W}$ be the set of possible workloads. Let $(B_{min}, B_{max}) \in \mathbb{N}_+^2$ be respectively the

minimum buffer size and the maximum buffer size. Let $b_t \in [B_{min}, B_{max}]$ be the buffer size of the database at time $t$. Let $f$ be the function that given the buffer size at time $t$ $b_t$, and the workload at the next time step $w_{t+1}$ returns the latency at the next time step $l_{t+1}$:

$$f : \mathbb{R}_+ \times [B_{min}, B_{max}] \to \mathbb{R}_+, \quad l_{t+1} \leftarrow f(w_{t+1}, b_t). \tag{5.1}$$

The optimal buffer size at time $t$ of the database is defined as:

$$b_t^* = \arg \min_{b_t \in [B_{min}, B_{max}]} \{b_t \text{ such that } f(w_{t+1}, b_t) \leq l^*\}. \tag{5.2}$$

This optimal buffer size of the database is denoted as the Oracle.

$f$ is a complex function that depends on the database environment, and that in the general case is unknown. Moreover, the buffer size of the database is allocated at time $t$, while the latency at time $t+1$ depends on the workload at time $t+1$. As a consequence, even with the only knowledge of $f$, the latency at time $t+1$ cannot be evaluated. Let $w_{t+1}$ be the observed value at time $t+1$ of the random variable $w$ sampled from an unknown distribution $\upsilon_w$. Hence, the latency $l_{t+1}$ is also observed from a random variable $l$, and is sampled from an unknown distribution $\upsilon_l | w_t, b_t$.

**Algorithm 5** Optimization of buffer size

**for** $t \leftarrow 1$ **to** $T-1$ **do**
  Execute workload $w_t \sim \upsilon_w$
  Agent chooses buffer size $b_t$
  Latency $l_{t+1} \sim \upsilon_l \mid w_t, b_t$ is revealed
**end for**

Algorithm 5 summarizes the optimization problem of the buffer size of a given database. Observing the latency at time $t+1$, a reward can be evaluated for the choice of the buffer size at time $t$. Hence, this problem can be formalized as a reinforcement learning problem, where an agent interacts with an unknown environment [122]. For the sake of simplicity, in the following sections, we assume that every time step $t$ includes both the buffer size selection and the latency measurement phases.

**Environment** The environment refers to the DBMS and the workload. Specifically, the buffer size of the DBMS is the parameter being tuned. The DBMS responds to actions (changes in the buffer cache size) by reflecting changes in its states and performance metrics for the given workload, and the latency. These metrics are translated to a reward using the reward function (Section 5.3) to serve as feedback for the agent, allowing it to evaluate the impact of its actions and adapt accordingly.

**States** For choosing the buffer size given a workload, the agent observes the current state of the database. During execution, the DBMS generates various application-related metrics, which are used to describe its current state $s_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of possible states. Specifically, under the context of MariaDB, 51 external metrics from the InnoDB global status and information schema, such as "Innodb_buffer_pool_reads", "innodb_buffer_pool_read _requests" and "innodb_rows_inserted," are used to form the state vector. These metrics, automatically computed by the DBMS, offer a comprehensive view of its operational state—covering aspects such as concurrency levels, buffer-pool activity, locking behavior, and row-level insertions. Depending on the DBMS version and configuration, more than 400 distinct metrics are maintained in the Information Schema and Performance Schema. DBAs commonly rely on these indicators to monitor system

health [13], and they can be easily retrieved by querying the native DBMS information tables.

It is important to note that some of these metrics are cumulative, so we compute the difference between two consecutive measurements to capture meaningful state observations. The selection of these metrics is aimed at accurately representing the DBMS state while maintaining a manageable dimensionality. To achieve this balance, we applied a correlation-based filter—drawing on the methodology from OtterTune [29]—to remove redundant metrics and retain only the 51 most informative features.

**Actions** Rather than directly choosing the buffer size given a database state, we choose to reach the optimal buffer size by increments or decrements of 128MB. The rationale behind the fixed increments or decrements of 128MB is to avoid drastic changes to the buffer size, which could negatively impact DBMS performance. Adjusting the buffer size requires the DBMS to reorganize its buffer pool, which can introduce instability. Although variable increments would accelerate the tuning speed, it would eventually lead to a larger state space, making the optimization of the tuning more complex and less controllable. Therefore, our goal is to implement a more moderate and gradual adjustment strategy to ensure system stability while optimizing performance.

The set $\mathcal{A}$ is defined with three actions:

1. *Down*: Decrease the buffer pool size by 128MB,
2. *Stay*: Maintain the current buffer pool size,
3. *Up*: Increase the buffer pool size by 128MB.

**Policy** A policy $\pi \in \Pi : \mathcal{S} \rightarrow \mathcal{A}$ returns the action to play (*Down*, *Stay*, *Up*) given the current state of the database, where $\Pi$ is the set of policies that depends on the used RL algorithm. When the reward is received, the policy is updated. Any RL algorithms (probabilistic or deterministic) can be used for updating the policy, which aims to handle the optimization of buffer size (Algorithm 5).

**Reward Function** The reward function allows the optimization of the policy, hence defining the behavior of the policy; it is a key element in reinforcement learning algorithms. In contrast to other DBMS auto-tuning projects [13, 25, 32, 123], which adopt reward functions based only on the performance metric for evaluating the reward, whatever the action, we design a reward function that is dependent on the chosen action (cf. Figure 5.3). The dashed black line marks the target latency threshold. The blue curve overlaps with the red curve for high latencies, indicating convergence to similar rewards.

Our reward function is designed to impose a severe penalty when $l_t > l^*$ (an SLA violation), while cases where $l_t < l^*$ but $b_t > b_t^*$ are considered suboptimal yet acceptable regarding SLA compliance. Moreover, using different functions for each action allows for favoring different behaviors: for instance, rewarding more the action *Up* leads to more conservative policies minimizing SLA violations at the expense of minimizing the buffer size, while rewarding more the action *Down* leads to more risky policies in terms of SLA violations for minimizing the buffer size.

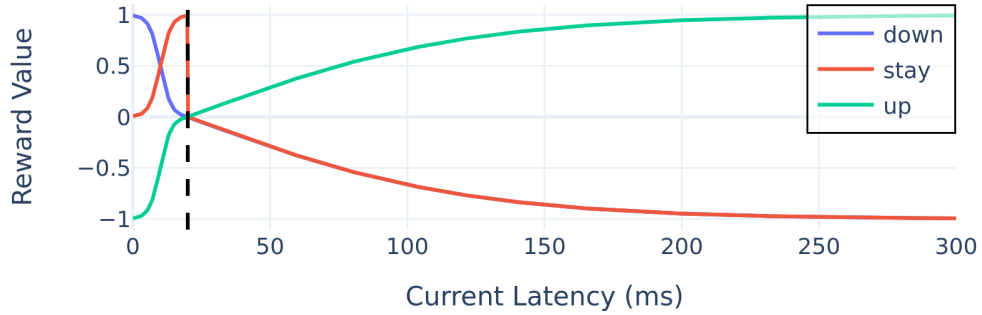


Figure 5.3: Shape of the reward function for each action

Let $d_+$ and $d_-$ be functions $\mathbb{R}_+ \to [0, 1]$:

$$d_+(l_t) = \frac{1}{1 + \exp\left(-10\left(\frac{l^* - l_t}{l^*} - 0.5\right)\right)}, \tag{5.3}$$

$$d_-(l_t) = 2\left(\frac{1}{1 + \exp\left(\frac{l^* - l_t}{-50}\right)} - 0.5\right). \tag{5.4}$$

The slopes of these scaled sigmoid functions are graphically selected (cf. Figure 5.3). The reward for actions (*Down*, *Stay*, *Up*) is defined as:

$$r_t = \begin{cases} \Big((1-\alpha)d_+(l_t),\ \beta(1 - d_+(l_t)),\ -\alpha d_+(l_t)\Big) & \text{if } l_t < l^*, \\ \Big((1-\alpha)d_-(l_t),\ \beta d_-(l_t),\ -\alpha d_-(l_t)\Big) & \text{if } l_t \geq l^*, \end{cases} \tag{5.5}$$

where $\alpha, \beta \in [0, 1]^2$ are parameters that have to be optimized (see Section 5.5.4).

The reward function and its slope parameters are designed to reward for increasing the buffer size when actual performance does not meet the SLA requirements; punish the increase of buffer size when the SLA requirements are already met; reward the agent for staying at a certain buffer size that satisfies the SLA requirements; severely penalize deviations that fail to meet with SLA requirements. This reward structure encourages the system to self-correct and maintain performance within the optimal range.

## 5.4 System Overview

In this section, we present our RL-based database tuning system using (cf. Figure 5.4). It uses three phases: data collection, Exploration (model training), and Exploitation (real-time tuning), which we discuss in detail in the following sections. The use of these three phases is because it is time-consuming and resource-intensive to train the RL algorithm on the real database environment by collecting the metrics in real-time. Indeed, after changing the buffer size, a waiting time is necessary for system stabilization before evaluating the state and the latency. Therefore, it is found more efficient to first collect training data that covers all possible states and then use it for model training. This allows for testing different algorithms without the measurement costs.

### 5.4.1 Data Collection Phase

The data collection phase is designed to gather data for the subsequent model training and testing (Algorithm 6). The data consists of a series of sextuplets $\langle w, b_t, s_t, l_t, l_{t^*}, b_{t^*} \rangle_{w \in \mathcal{W}, t \in \{1,\ldots,T\}}$, where:

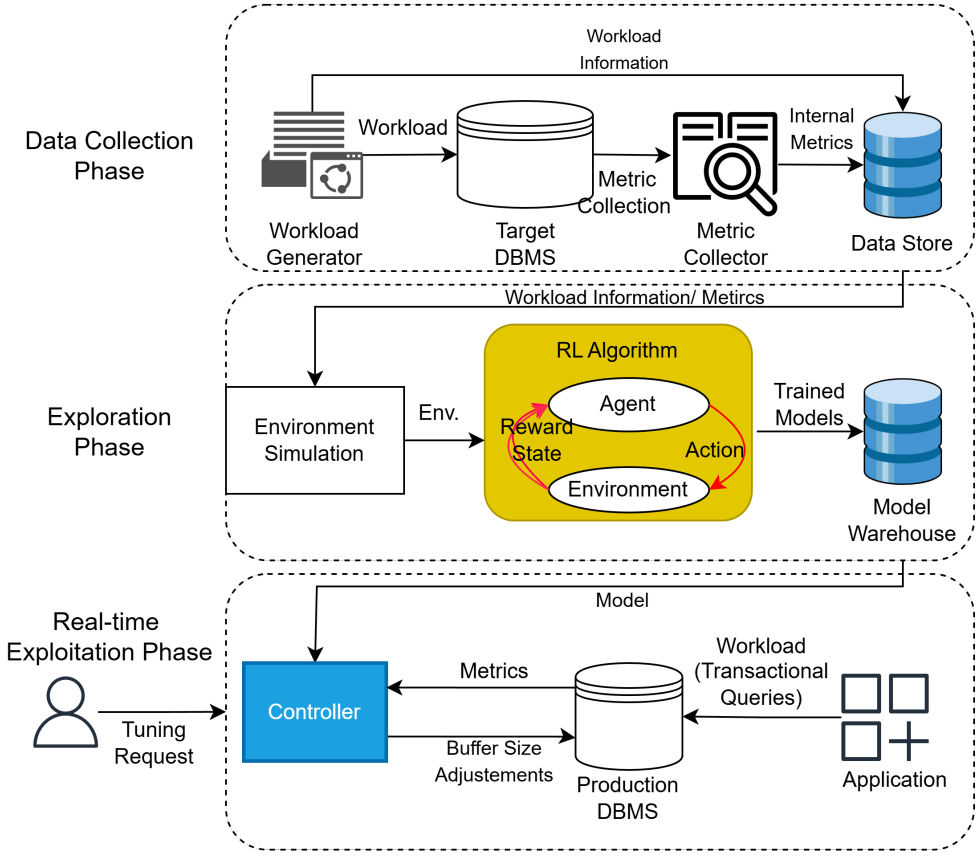


Figure 5.4: System Architecture

- $w \in \mathcal{W}$ represents the workload,
- $b_t \in [B_{min}, B_{max}]$ denotes the buffer size,
- $s_t \in \mathcal{S}$ is the database state, which is reflected by the internal metrics as discussed in Section 5.3,
- $l_t \in \mathbb{R}_+$ is the latency,
- $l_w^*$ is the optimal latency for workload $w$,
- $b_w^*$ is the optimal buffer size for workload $w$.

After the data generation, the data store is split into three datasets: one for training the models, one for validating the parameters, and one for testing.

Algorithm 6 uses SYSBENCH [49] as a workload generator to inject SQL queries on the DBMS. During the execution of the workload, at each timestep $t$, the metric collector retrieves the metrics relevant to DBMS state $s_t$ and the latency $l_t$. Although using instant latency as the primary metric for optimizing buffer size is desirable, it is impractical. When changing the buffer size on a busy database, measuring instant latency is a very noisy process. To address this, a warm-up time is implemented to ensure metric stability before collecting. The 95th percentile of latency is also used as it provides a more stable and representative measure of system performance. Under typical OLTP conditions and depending on the workload, the DBMS may process from hundreds to tens of thousands of queries and data modification operations within a single second. Hence, the 95th percentile latency is computed from the instantaneous latencies of all queries executed during a one-second interval.

**Algorithm 6** Data Collection

**for** $w \in \mathcal{W}$ **do**
  $t \leftarrow 0$
  **for** $b_t \leftarrow B_{\max}$ **down to** $B_{\min}$ **do**
    Set buffer size to $b_t$
    Execute $w$
    Measure $s_t$, $l_t$
    $t \leftarrow t + 1$
  **end for**
  $b_w^* \leftarrow \min_{i \in \{0,\ldots,t\} \,:\, l_i \le l^*} b_i$
  $l_w^* \leftarrow \max_{i \in \{0,\ldots,t\} \,:\, l_i \le l^*} l_i$
  **for** $i \in \{0, \ldots, t\}$ **do**
    **write** $(w, b_i, s_i, l_i, l_w^*, b_w^*)$ to datastore
  **end for**
**end for**
Split the datastore by workload $w$ into training, validation, and test sets.

### 5.4.2 Exploration Phase

In the exploration phase (Figure 5.4), we construct a virtual DBMS tuning environment by replaying a dataset of sextuplets collected with Algorithm 6: $\langle w,\, b_t,\, s_t,\, l_t,\, l_w^*,\, b_w^* \rangle$, where each metric was gathered from extensive experiments on a real DBMS and covers all possible system states. At each step, the agent's action sets the virtual DBMS buffer size $b_t$. We then query the collected dataset with the full context tuple $\langle w, b_t, l_w^*, b_w^* \rangle$ to retrieve the corresponding state–metric pair $\langle s_t, l_t \rangle$. This "memorize-and-replay" approach accelerates training and guarantees full reproducibility. The goal of this phase is to optimize the policy parameters by training the selected algorithm on our dataset and rigorously evaluating the resulting policy's performance.

**Training** For each workload in the training set, at the beginning of the exploration phase (Algorithm 7 line 3). The starting point is initialized by selecting a buffer size from a truncated Gaussian distribution $\mathcal{N}_{\mathcal{T}}(\mu, \sigma, a, b)$ where parameters $\mu = b_w^*$ is the mean, $\sigma = \left\lfloor \frac{\max\{\, b_w^*, B_{\max} - b_w^* \}}{2} \right\rfloor$ is the standard deviation $a = \frac{B_{\min} - b_w^*}{\sigma}$ is the lower bound, and $b = \frac{B_{\max} - b_w^*}{\sigma}$ is the upper bound.

This approach encourages the agent to interact frequently with values near $b_w^*$, helping it learn to adjust both in extreme situations and in more delicate scenarios close to $b_w^*$.

Algorithm 7 outlines the exploration process. The $T$ is chosen to be sufficiently large so that the agent can thoroughly examine different states $s_t$ under a given workload $w$. For instance, if the valid buffer pool size varies from 128 MB to 8 GB, a minimum of 64 steps ensures the agent can fully explore the range with an increment of 128 MB per step.

**Evaluation** To optimize our policy, we use the dataset of sextuplets $\langle w,\, b_t,\, s_t,\, l_t,\, l_w^*,\, b_w^* \rangle$ where: $l_t$ yields the immediate reward, $b_t$ identifies the buffer-size decision that corresponds to the agent's discrete action, and $s_t$ describes the DBMS state under workload $w$.

Because we have exhaustively recorded every possible state for each workload, an **Oracle** can compute the optimal targets $\left(l_w^*, b_w^*\right)$ for each one. From this, we derive a **Optimal Policy** $\pi^*$ that, at each step, picks the action in $\mathcal{A} = \{\mathit{Down}, \mathit{Stay}, \mathit{Up}\}$ which moves $b_t$ closest to the oracle's $b_w^*$. Although $\pi^*$ represents the best achievable tuning behavior under our discrete action set, its performance is inherently capped by

**Algorithm 7** RL Exploration Phase

Initialize policy $\pi_0$ randomly
**for** $w \in \mathcal{W}$ **do**
    Set initial buffer size $b_0 \sim \mathcal{N}_{\mathcal{T}}(\mu, \sigma, a, b)$
    **for** $t \leftarrow 1$ **to** $T$ **do**
        Observe state $s_t$
        Select action $a_t \in \{Down, Stay, Up\}$ using $\pi_t$
        Apply $a_t$ to update buffer size $b_t$
        Environment transitions from $s_t$ to $s'_t$
        Observe $s'_t$, $l_t$ and compute reward $r_t$ using Eq. 5.5
        Update $\pi_t$ with transition $(s_t, a_t, r_t, s'_t)$
    **end for**
**end for**
Evaluate and store final policy $\pi_T$

that coarseness (cf. Section 5.3).

All learned policies $\pi$ (e.g. PPO, DQN, A2C) are then evaluated by their proximity to $\pi_o$ using the normalized-distance metric (Alg 8), thereby quantifying how closely each agent approximates the oracle-derived reference.

**Algorithm 8** Optimal Policy $\pi^*$

Load optimal buffer size $b^*_w$ for workload $w$ from datastore
**for** $t \leftarrow 1$ **to** $T$ **do**
    **if** $b_t < b^*_w$ **then**
        **action** $\leftarrow$ *Up*
    **else if** $b_t = b^*_w$ **then**
        **action** $\leftarrow$ *Stay*
    **else**
        **action** $\leftarrow$ *Down*
    **end if**
**end for**

It is important to highlight that with the set of actions $\mathcal{A}$, the optimal policy can make SLA violations (when $l_t > l^*$) since it could not reach the optimal buffer size in one time step. SLA violation does not necessarily imply a mistake by the policy. For instance, during high workload scenarios combined with low buffer sizes, even if the policy attempts to increase the buffer size, it might not immediately satisfy the SLA constraints. Consequently, even the optimal policy would accumulate instances of SLA violations.

The normalized distance of a given policy $\pi$ to the optimal policy $\pi^*$ on the dataset of length $T$ is defined as following:

$$d_T(\pi, \pi^*) = \frac{\sum_{t=1}^{T} |\mathbb{1}(l_t > l^*) - \mathbb{1}(l^*_t > l^*)|}{\sum_{t=1}^{T} \mathbb{1}(l^*_t > l^*)} + \frac{\sum_{t=1}^{T} |b_t - b^*_t|}{\sum_{t=1}^{T} b^*_t}, \tag{5.6}$$

where $b^*_t$ is buffer size chosen by the optimal policy (Algorithm 8), and $l^*_t$ the corresponding latency. This normalized distance serves as a general metric for evaluating and comparing policies by measuring how closely each one approximates the optimal policy.

### 5.4.3 Exploitation Phase

After the policy has been trained and evaluated, the policy is stored in a model warehouse. The model warehouse stores policies trained under different scenarios and SLA

targets, yielding policies with different behaviors. The model warehouse provides different solutions for different needs. In the real-time exploitation phase (Figure 5.4), when the DBMS administrator makes a tuning request, they have to choose a model corresponding to their DBMS configuration in the model warehouse. The controller loads the selected model and starts to collect the metrics (DBMS state and latency) from the DBMS, which is typically under the production workload from an application. The RL algorithm tunes the buffer size based on the collected metrics (state and latency) by applying modifications to the production DBMS. This process is repeated to optimize DRAM utilization while preventing SLA violations.

## 5.5 Experimental Results

In this section, the performance and the efficiency of MicroTune are evaluated under different core algorithms and baselines. We first show their efficiency by comparing their training and inference time costs. Then, we demonstrate their effectiveness by evaluating their capability to reduce DRAM utilization while respecting SLA constraints compared to baseline. To evaluate the performance of different RL algorithms, perform hyperparameter tuning of these algorithms (cf. Section 5.5.3), and perform reward shaping (cf. Section 5.5.4), the following metrics are used:

**1. Total number of SLA Violations** Let $V_{SLA}$ denote the total number of SLA violations observed. This metric evaluates the number of times where $l_t$ under $b_t$ adjusted by the policy fails to meet the SLA requirement $l^*$. It serves as an indicator of the agent's capability to satisfy SLA constraints. The $V_{SLA}$ is defined as follows:

$$V_{SLA_T}(\pi) = \sum_{t=1}^{T} \mathbb{1}(l_t > l^*). \tag{5.7}$$

**2. Cumulative DRAM Utilization** The cumulative DRAM Utilization metric, denoted $C_{DRAM}$, calculates the total buffer size utilized across all time steps. It is used to assess the agent's ability to minimize buffer size while managing workload demands effectively. This metric is defined as follows:

$$C_{DRAM_T}(\pi) = \sum_{t=1}^{T} b_t. \tag{5.8}$$

**3. Normalized Distance to optimal policy $d_T(\pi, \pi^*)$ (Equation 5.6)** The main metric we use to evaluate the overall effectiveness of a given policy as it leverages both the $V_{SLA}$ and the $C_{DRAM}$ compared to the optimal policy (cf. Algorithm 8).

### 5.5.1 Workloads & Datasets

**Description** We used MariaDB 11.1.3 and the Sysbench benchmark to test and evaluate our work. MariaDB is deployed on Flexible Engine [103] using machines with 4 vCPU cores and 16 GB of DRAM (model c6.xlarge.4). We collected states and performances of a total of 592 different workloads generated by Sysbench executed on MariaDB. It takes approximately 37.5 minutes to collect the data points for one workload. The setup relies on 3 machines, one for the benchmarking tool as a database client, one for the DBMS server, and one to pilot operations and collect data. The whole run continued for 16 days to collect data for all 592 workloads.

A workload $w$ within our database tuning context is characterized by the following factors:

- The database size evolves from 128MB up to 8GB with several tables (5, 15, 22, 30, 40, 50) and number of rows per table (1000000, 300000, 100000).
- The number of simultaneous clients connected to the database (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12).
- The distribution of data access, with four types currently available: special, Gaussian, uniform, and Pareto. The uniform distribution allows for more extensive and equal access to all data during a load, while Pareto focuses on accessing a smaller subset of the dataset more intensively.

We collect 95th percentile latency for every second from the benchmark tool SYSBENCH. Due to system fluctuations and network issues, we may observe outliers of the measured Latency in our datasets even with the 95th percentile. These outliers may significantly skew the training process and lead to inaccurate learned policies of the agent. To address this issue, the technique known as smoothing is applied, which adjusts the values of certain data points to mitigate the impact of these outliers. Smoothing involves replacing a data point $y_i$ with a weighted average of its neighboring points. We adopt a simple moving average smoothing method, each point $y_i$ in the dataset might be replaced by the average of itself and its two nearest neighbors if it exceeds a certain variation:

$$y'_i = \frac{1}{3}(y_{i-1} + y_i + y_{i+1}). \tag{5.9}$$

This approach helps to reduce the influence of any single outlier by distributing its impact across several points.

Following Algorithm 6, we initialize the SYSBENCH benchmark with the DBMS set to a buffer size of 8 GB. Then, we decrease the DBMS buffer size after the collection of each tuple $\langle w, b_t, s_t, l_t, l^*_w, b^*_w \rangle$, with a granularity of 128 MB and repeat this process until it reaches the minimum value, which is 128 MB. Then, we change the workload and repeat the process. During this process, a total of $37,888$ tuples $\langle w, b_t, s_t, l_t, l^*_w, b^*_w \rangle$ are generated. The total steps for each workload are set at $T = 64$. With $b_{max} = 8G$ and $b_{min} = 128MB$, $T = 64$ steps with a decrement of 128 MB is sufficient to reach the optimal buffer size.

The collected dataset is partitioned into three disjoint subsets: 60% for training (used in the exploration-phase model fitting), 20% for validation (for RL hyperparameter tuning and reward shaping), and 20% for testing (for the final evaluation of each algorithm). By reserving a dedicated validation set, we avoid optimizing hyperparameters and reward functions on the test data, thereby reducing overfitting. Each subset is then used to instantiate its own virtual DBMS tuning environment (see Section 5.4.2). Importantly, data from each workload remains strictly separate—no workload data appears in more than one of the training, validation, or test sets. Ensuring no overlap between workloads across the datasets not only reduces evaluation bias but also enables us to assess the model's transferability and generalizability on unseen workloads.

### 5.5.2 Algorithms Under Study

In this section, we present the algorithms evaluated in our study. To ensure comprehensive coverage, the evaluation includes several deep RL algorithms with Stable-Baselines3 [124] (SB3) implementation, alongside contextual bandit algorithms. Horizontal Pod Autoscaler (HPA) [125] is assessed as a baseline, and some other rule-based baselines are also proposed.

Rule-based baselines do not require a training phase but always rely on their specific inputs: for example, the *Basic* and *HPA* baselines require $l_t$, the *Miss Ratio* baseline depends on the miss ratio distribution, and the Optimal Policy is contingent on $b_w^*$. RL baselines require a training phase with their specific inputs, but once trained, in the exploitation phase, they can be used without the latency $l_t$ as reward computation is not needed anymore. Baselines that do not rely on the latency in the exploitation phase are more reflective of most industrial scenarios, where collecting real-time latency data is either impractical or prohibitively expensive.

**Baseline: Optimal Policy** The algorithm that makes perfect actions as described in Algorithm 8. This baseline cannot be used in real-time in the exploitation phase; note that this is calculated offline and not deployable online.

**Baseline: Basic** A straightforward method to manage buffer sizes is to mimic the intuitive actions a human operator might take in real time. Algorithm 9 illustrates this reactive, threshold-based mechanism. Given a workload, one could compare at each interval, $l_t$ against $l^*$ with a tolerance $\epsilon$ and adjust the buffer size accordingly. In our experimentation, the tolerance is set to $\epsilon = 0.1$.

**Algorithm 9** Basic Tuning Baseline

```
for t ← 1 to T do
    Observe latency l_t
    if l_t > l* then
        action ← Up
    else if l_t < l* · (1 − ε) then
        action ← Down
    else
        action ← Stay
    end if
    Apply action and update buffer size b_{t+1}
end for
```

**Baseline: Strategy HPA** HPA [125] is an algorithm used in Kubernetes to automatically adjust the number of *Pods*—the smallest deployable units that encapsulate one or more application containers and their shared resources—for an application facing a fluctuating workload, similar to how RAM resources are dynamically allocated in a DBMS. HPA determines the number of Pods based on the ratio between the desired and the current metric values, as follows:

$$\text{Optimal Replicas} = \lceil \text{current Replicas} \times \left( \frac{\text{current Metric Value}}{\text{desired Metric Value}} \right) \rceil. \tag{5.10}$$

Inspired by HPA, in the context of memory allocation for MicroTune, Equation 5.10 has been adapted to compute, at each time step $t$, the optimal buffer size denoted $b^*_{HPA_t}$, as

$$b^*_{HPA_t} = b_t \times \left( \frac{l_t}{l^*} \right). \tag{5.11}$$

Algorithm 10 describes the process: if $b_t$ is less than $b^*_{HPA_t}$, then the action *Up* is selected, if $b_t$ is within the range of $[b^*_{HPA_t}, b^*_{HPA_t} + 128\,\text{MB}]$, then the action *Stay* is chosen, else the action *Down* is applied. This strategy allows us to adjust $b_t$ step by step to $b^*_{HPA_t}$

while updating the $b^*_{HPA_t}$ for every time steps $t$. The range of $[b^*_{HPA_t}, b^*_{HPA_t} + 128\,\text{MB}]$ allows the HPA baseline to over-allocate a little DRAM space to ensure that SLA is met.

As shown in Equation 5.11, it is important to note that this algorithm assumes a strong linear relationship between the latency and the buffer size. The collected data suggests that this assumption may hold in certain cases, but is limited.

**Algorithm 10** Baseline HPA Strategy

**for** $t \leftarrow 1$ **to** $T$ **do**
    Observe latency $l_t$
    Compute target buffer size:

$$b^*_{\text{HPA},t} \leftarrow b_t \times \frac{l_t}{l^*}$$

    **if** $b_t < b^*_{\text{HPA},t}$ **then**
        **action** $\leftarrow$ Up
    **else if** $b_t \in [\, b^*_{\text{HPA},t},\ b^*_{\text{HPA},t} + 128\,\text{MB}\,]$ **then**
        **action** $\leftarrow$ *Stay*
    **else**
        **action** $\leftarrow$ *Down*
    **end if**
    Apply action and update buffer size $b_{t+1}$
**end for**

**Baseline: Miss Ratio** Similar to iBTune [3], a rule-based tuning baseline that leverages the miss ratio is implemented. The miss ratio is defined as the proportion of data read requests that are not served from the buffer pool (i.e., a cache miss) and consequently require disk access. In InnoDB engine, this is calculated as the ratio of `innodb_buffer_pool_reads` (the number of physical reads from disk) to `innodb_buffer_pool_read_requests` (the total number of read requests), both metrics are observed in state metrics $s_t$. A lower miss ratio indicates that the buffer pool is effectively caching data, thereby reducing disk I/O and enhancing performance, while a higher miss ratio suggests that the current buffer pool size may be insufficient for the workload.

While iBTune focuses on setting the buffer pool size to achieve a desired miss ratio, our objective is to attain a specific latency level that complies with the SLA. Although there is no universally generalizable relationship between latency and miss ratio, we empirically identify a miss ratio threshold corresponding to the target SLA range. Note that this threshold depends on both workload and hardware characteristics, so our approach serves only as a comparison baseline rather than an optimal solution.

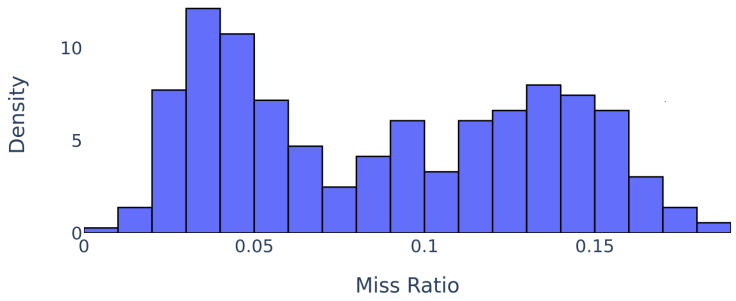


Figure 5.5: Distribution of Miss Ratio under given SLA at 20ms

The difficulty in defining a single adaptive threshold for all workloads is evident, as the distribution of the miss ratio for the given latency at 20 ms varies according to different

workloads and does not show a specific distribution pattern (Figure 5.5). Under the example with target latency at 20 ms, we use a target miss ratio range of 0.03–0.04 that most frequently achieves the desired latency. If the miss ratio lies in this interval, we *Stay* at the current setting; if it exceeds 0.04, we *Up* the buffer pool size; if it falls below 0.03, we *Down* the buffer pool size (cf. Algorithm 11). The rationale behind this is that a larger buffer can store more data, thereby reducing the miss ratio. *Miss Ratio* does not necessitate evaluating the latency and hence can be used in the exploitation phase.

**Algorithm 11** Miss Ratio Baseline

```
for t ← 1 to T do
    Observe state s_t and compute miss ratio m_t
    if m_t < m_min then
        action ← Down
    else if m_t > m_max then
        action ← Up
    else
        action ← Stay
    end if
    Apply action and update buffer size b_{t+1}
end for
```

**LinUCB** LinUCB [126] (*Linear Upper Confidence Bound*) is a contextual bandit algorithm used for online learning and recommendation systems. It has been applied in similar contexts, such as cloud horizontal scaling by [127], and demonstrates good performance. It models the reward as a linear function of the context (features) and selects actions (arms) based on an upper confidence bound strategy. Specifically, LinUCB maintains a linear model for each arm, updating its parameters as it receives feedback. At each step, it chooses the arm with the highest upper confidence bound, balancing exploration (trying new actions) and exploitation (choosing the best-known action). This approach efficiently handles high-dimensional context spaces and provides strong theoretical guarantees for cumulative regret minimization. Here, we use the implementation extracted from LinUCB disjoint arms [128].

**SAC** *Soft Actor-Critic* (SAC) [129] is an off-policy actor-critic algorithm that integrates the maximum entropy framework into reinforcement learning. It maximizes both expected rewards and policy entropy to encourage exploration. SAC uses separate networks for policy and value estimation, often employing two Q-value networks to mitigate overestimation bias. A temperature parameter balances reward maximization with entropy, making SAC particularly effective in continuous action spaces.

**PPO** PPO [130] (*Proximal Policy Optimization*) is a RL algorithm that aims to improve policy updates stably and efficiently. It strikes a balance between exploration and exploitation by using a clipped objective function, which prevents large updates that could destabilize training. PPO optimizes a surrogate objective using stochastic gradient descent, ensuring that new policies do not deviate too far from old ones by penalizing excessive changes. This makes PPO robust and relatively easy to implement, leading to strong performance across various environments.

**DDPG** *Deep Deterministic Policy Gradient* [131] (DDPG) is a RL algorithm designed for continuous action spaces. It combines the benefits of DQN (*Deep Q-Network*) and

policy gradient methods. DDPG employs an actor-critic architecture: the actor network proposes actions given states, while the critic network evaluates these actions by estimating the Q-value. The algorithm uses experience replay to store and sample experiences, which stabilizes training. Additionally, DDPG employs target networks to further stabilize learning by slowly updating the target values. This approach allows DDPG to learn complex policies in high-dimensional continuous action spaces efficiently and is shown to be effective in similar DBMS tuning problems in CDBTune [13]. In the SB3 implementation of DDPG, only a continuous action space is supported. This space is defined as the interval $[-1, 1]$. Consequently, the continuous action value returned by the model is rounded to yield a discrete action: $-1$ for *down*, 0 for *Stay*, and 1 for *up*.

**DQN** *Deep Q-Networks* (DQN) [132] was introduced as a breakthrough method for learning control policies in high-dimensional state spaces (e.g., Atari games) using deep neural networks. The algorithm combines Q-learning with two key components to stabilize training: an experience replay buffer, which stores past transitions and samples them uniformly to break correlation, and a target network, which is updated slowly to reduce divergence. DQN approximates the optimal action-value function $Q(s, a)$ with a deep neural network, allowing it to handle large state spaces in discrete action settings effectively.

**A2C** *Advantage Actor-Critic* (A2C) is a synchronous variant of the Asynchronous Advantage Actor-Critic algorithm [133]. It employs two networks: an actor network that outputs a stochastic policy (i.e., action probabilities) and a critic network that estimates the state-value function. The key idea is to use the advantage function, which measures how much better or worse an action is compared to an average action in a given state, thereby reducing the variance in policy gradient updates. By synchronizing multiple parallel workers (as opposed to the asynchronous setup in A3C), A2C offers a more stable and reproducible training process while retaining the benefits of actor-critic methods.

### 5.5.3 RL Hyperparameters Tuning

RL performance depends critically on hyperparameters such as learning rate (which controls the step size during parameter updates), batch size (the number of samples processed in one iteration), and discount factor gamma (the discount factor that balances immediate and future rewards)—to avoid unstable training or poor convergence [134]. We used Optuna [135] with its *Tree-structured Parzen Estimator* (TPE) [136], which models separate distributions for promising and poor values and selects new samples by maximizing expected improvement. To ensure a diverse and well-informed search, the optimization process is initialized with random sampling for the first few trials before applying TPE.

We use the Normalized Distance to optimal policy (see Equation 5.6) as the performance metric. We optimize key hyperparameters using Optuna. First, the model is trained on the training dataset and subsequently evaluated on a separate validation dataset. The optimal hyperparameter configuration is selected by identifying the one that minimizes the Normalized Distance to optimal policy, averaged over multiple evaluation runs—each initialized with a different random seed—to ensure robustness and mitigate overfitting.

### 5.5.4 Reward Shaping

Most RL algorithms require extensive reward engineering to effectively solve challenging tasks [137]. To refine the reward function, we constrain the parameters $\alpha$ and $\beta$ to lie

within the interval [0, 1]. This calibration allows us to strike an appropriate balance between minimizing SLA violations and optimizing DRAM utilization.

In Equation 5.5, $\alpha$ governs the balance between the *Up* and *Down* actions. A smaller value of $\alpha$ implies a stronger penalty for executing a *Down* action when SLA requirements are not met, while simultaneously rewarding a *Down* action when they are satisfied. Conversely, a larger value of $\alpha$ favors rewarding an *Up* action in cases where SLA requirements are unmet and penalizes the *Up* action when they are met. Meanwhile, the coefficient $\beta$ modulates the impact of the *Stay* action relative to the other two actions. An elevated $\beta$ indicates that the agent gains more benefit from choosing to *Stay*, whereas a lower $\beta$ suggests that the agent should place less emphasis on remaining in the same state.

We performed a grid search over the coefficients $\alpha$ and $\beta$, training on the 60% training set and evaluating on the 20% validation set. Our objective was to minimize both the mean and standard deviation of the normalized distance to the optimal policy (see Equation 5.6). The search identified $\alpha = 0.2$ and $\beta = 0$ as the optimal values. Consequently, the reward function reduces to:

$$r_t = \begin{cases} \big(0.8 \cdot d_+(l_t),\ 0,\ -0.2 \cdot d_+(l_t)\big) & \text{if } l_t < l^*, \\ \big(0.8 \cdot d_-(l_t),\ 0,\ -0.2 \cdot d_-(l_t)\big) & \text{if } l_t \geq l^*. \end{cases} \tag{5.12}$$

Figure 5.6 illustrates the reward function using the optimized values of $\alpha$ and $\beta$ when $l^* = 20$ ms. The visualization reveals that, with the selected $\beta$, the *Stay* action no longer yields the highest reward in every scenario; instead, either the '*Up*' or '*Down*' action produces a relatively larger reward. This suggests that the agent learns most effectively when the *Stay* action has minimal impact—allowing it to remain in its current state without significant reward or penalty. Although consistently choosing *Stay* would not maximize the immediate reward, it helps maintain system stability when the agent is uncertain about which action to take, ultimately leading to a lower long-term error. To verify this finding, we conducted additional experiments by varying $\beta$ while keeping $\alpha$ fixed at its optimal value. As shown in Figure 5.7, the minimum distance across all three algorithms is achieved when $\beta = 0$. This ensures the reward function strongly discourages growing the buffer through the larger weight on *Down* while not excessively penalizing efforts to lower the buffer *Up*. Simultaneously, it permits the agent to *Stay* at the current state when no decisive action is warranted, ultimately yielding the best overall performance.

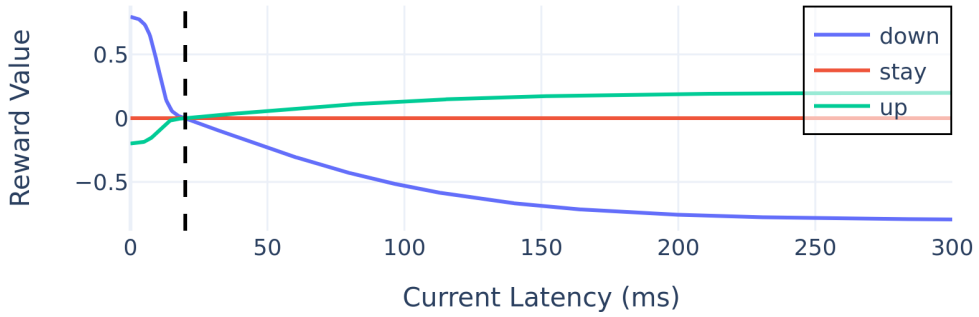


Figure 5.6: Visualization of Reward Function with coefficients

### 5.5.5 Efficiency

The algorithms under evaluation adopt diverse structures, ranging from deep learning-based approaches to simple heuristic strategies, with varying inference and training times.

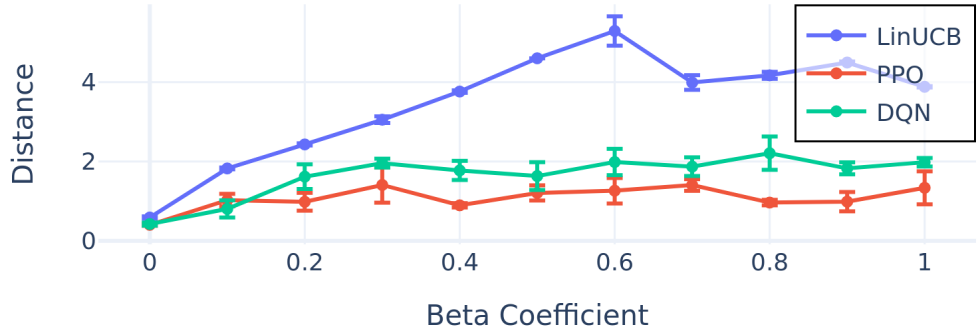


Figure 5.7: Mean & Std for Normalized Distance to the optimal policy under different Beta Coefficient

In this section, we analyze and discuss the efficiency of each algorithm in detail. All experiments were performed under identical hardware configurations to keep computational overhead consistent across all tests.

Table 5.2: Average training & inference time

| Metric | LinUCB | SAC | PPO | DDPG | A2C | DQN |
|---|---|---|---|---|---|---|
| Training Time | 26.16 s | 2046.78 s | 87.41 s | 2258.65 s | 90.70 s | 53.41 s |
| Inference Time | 0.59 ms | 0.66 ms | 0.59 ms | 0.66 ms | 0.70 ms | 0.46 ms |

As shown in Table 5.2, training times vary considerably—ranging from relatively low (e.g., 26.16 s for LinUCB) to very high (2258.65 s for DDPG). This is due to differences in algorithmic architecture and complexity. For example, LinUCB models only a linear relationship and require only a few iterations for training, whereas more complex algorithms such as DDPG involve training multiple deep neural networks and require significantly more iterations to converge. This leads to the observed variation in training times across different algorithms. However, all methods exhibit inference times below 1 ms, as inference primarily involves simple forward computations rather than extensive iterative processes. As a result, even computationally demanding deep RL methods can be deployed online with minimal overhead once trained.

### 5.5.6 In Vitro Effectiveness

Table 5.3 presents the performance of the studied algorithms on the test dataset under their best hyperparameter configurations, each tested with multiple random seeds to ensure robustness. Algorithms are first trained on the training dataset and then evaluated on a test dataset with entirely disjoint workloads. Since the test workloads are unseen during training, this evaluation measures both the model's ability to learn a policy under the training conditions and its capacity to generalize to novel workloads as they shift over time in real-world production scenarios. The $l^*$ is set at 20 ms, as DBA experts consider this value a reasonable compromise between resource utilization and system performance. The starting buffer pool size for each algorithm and each random seed is set to the same value to ensure a uniform testing environment and a fair comparison. Note that in practice one may choose a conservative threshold $l^*$ below the SLA limit. For example, with a 20 ms SLA, setting $l^* = 18$ ms prevents occasional SLA violation at the cost of higher memory usage. The trade-off depends on the application's specific requirements.

**Summary:** The baselines are clearly outperformed by RL algorithms. Among the RL algorithms, LinUCB is the worst, while PPO, DQN, and A2C show the smallest deviation from the optimal policy in terms of the Normalized Distance to optimal policy, achieving SLA violations and DRAM utilization closest to optimal policy (cf. Table 5.3).

Table 5.3: Performance of algorithms on test set

| | SLA Violations | | Cumulative DRAM Utilization (MB) | | Normalized Distance to Optimal Policy | |
|---|---|---|---|---|---|---|
| | Train | Test | Train | Test | Train | Test |
| LinUCB | 2651.3 ± 226.6 | 2195.8 ± 171.5 | 86.6 ± 1.0 | 78.7 ± 1.2 | 0.6 ± 0.1 | 1.4 ± 0.2 |
| DQN | 2054.9 ± 86.5 | 1288.3 ± 186.5 | 92.1 ± 2.6 | 82.9 ± 2.2 | 0.3 ± 0.0 | 0.6 ± 0.2 |
| A2C | 2040.3 ± 38.3 | 1335.5 ± 154.6 | 93.5 ± 2.4 | 83.6 ± 2.4 | 0.3 ± 0.0 | 0.6 ± 0.1 |
| PPO | 1946.3 ± 34.8 | 1249.7 ± 68.0 | 94.8 ± 2.0 | 86.3 ± 1.6 | 0.3 ± 0.0 | 0.6 ± 0.1 |
| DDPG | 2275.7 ± 615.1 | 1702.8 ± 720.7 | 92.0 ± 4.9 | 82.6 ± 4.4 | 0.4 ± 0.3 | 1.0 ± 0.7 |
| Basic | 4293.0 | 3312.0 | 75.4 | 66.4 | 1.3 | 2.4 |
| HPA | 3808.0 | 3366.0 | 74.8 | 63.4 | 1.0 | 2.4 |
| Miss Ratio | 5914.0 | 5907.0 | 91.7 | 80.8 | 2.3 | 5.2 |
| Optimal Policy | 1914.0 | 984.0 | 73.8 | 65.1 | 0.0 | 0.0 |

PPO, DQN, and A2C thrive because they harness sequential, non-linear feedback—PPO's clipped updates stabilize learning, DQN's replay buffer and target network smooth out Q-values, and A2C's advantage baseline cuts gradient variance. DDPG falters in our discrete, bounded action space and is hyper-sensitive; LinUCB's linear bandit can't model long-horizon non-linear DBMS dynamics, and simple baselines have no learning mechanism.

**Optimal Policy:** As the optimal policy is constrained to adjust the buffer size in fixed increments of 128 MB, when its buffer size starts too low compared to the optimal one, the optimal policy makes SLA violations: 984 during testing. The optimal policy achieves the lowest possible $V_{SLA}$ and the $C_{DRAM}$.

**Basic:** The Basic baseline exhibits a relatively high $V_{\text{SLA}}$ (3,312 on test). This is primarily due to the tolerance $\epsilon = 0.1$ defined in Algorithm 9. In many cases, no $b_t$ satisfies the range $[0.9\,l^*, l^*]$, leaving the Basic Baseline without a clear reference point for "staying" and causing it to oscillate. This underscores the challenges of designing a rule-based tuner capable of robust online tuning across varying workloads. Although its $C_{DRAM}$ (66.38 MB on test) is slightly better than some other methods. The high Normalized Distance to optimal policy (2.38 on test) indicates that the basic baseline may be too negligent in SLA compliance despite being resource efficient.

**HPA:** HPA reports a $V_{SLA}$ of 3,366 on test with low $C_{DRAM}$ (63.37 MB on test). However, the distance value (2.44 on the test) shows a significant deviation from the optimal policy, meaning that despite efficient memory usage, SLA adherence is not optimal.

**Miss Ratio:** This method reports a high $V_{SLA}$ (5,907 on test), and $C_{DRAM}$ is also high compared to other algorithms. Its Normalized Distance to optimal policy (5.24 on test) is the highest among the compared methods. This highlights the difficulty of solely relying on a Rule-Based approach to optimize Buffer Pool Size in a real-time manner with varying workloads.

**LinUCB:** LinUCB is the least efficient of the tested RL algorithms: Its distance to the optimal policy on the training set is almost double that of the other RL algorithms, which means that the linear model is not sufficient to achieve performance close to the optimal policy.

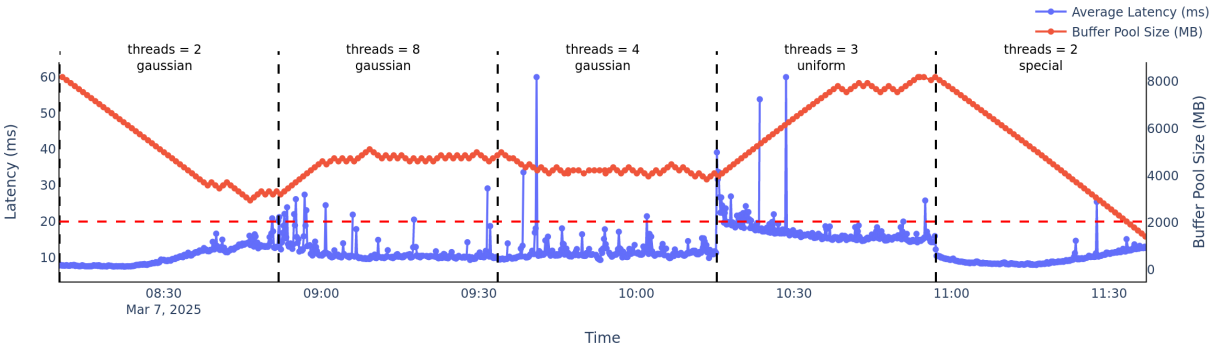


Figure 5.8: MicroTune Online Tuning with varying workloads with A2C agent

**DQN:** DQN shows a $V_{SLA}$ of 1,288.30 on the test, which is the closest to the optimal policy. Its $C_{DRAM}$ is slightly higher (82.91 MB on test) than optimal policy, and the normalized distance to optimal policy is low (0.3 on train, 0.6 on test). DQN achieves performance nearly matching the optimal policy in both SLA compliance and resource utilization.

**A2C:** Similar to DQN, A2C records an $V_{SLA}$ of 1,335.50 on test, with slightly higher $C_{DRAM}$ (83.62 MB on test). Its normalized Distance to optimal policy remains low (0.3 on train, 0.6 on test), demonstrating that A2C closely approximates the optimal policy's performance in balancing SLA adherence and memory usage.

**PPO:** PPO has an $V_{SLA}$ of 1,249.70 on test, and a $C_{DRAM}$ of 86.34 MB on test.

PPO achieves the best result when it comes to $V_{SLA}$ with a little overhead in resource utilization. As A2C and DQN, the distance values (0.3 on train, 0.6 on test) for PPO indicate a very small deviation from the optimal policy.

**DDPG:** DDPG records a $V_{SLA}$ of $1,702.8$ on test, with $C_{DRAM}$ of 82.55 MB on test. Its distance values (0.4 on train, 1.0 on test) are similar to PPO's. However, the high variability in the $V_{SLA}$ metric suggests greater instability in performance, potentially making DDPG less reliable than others.

### 5.5.7 In Vivo Effectiveness

In this section, we demonstrate how our trained MicroTune agent, using the A2C algorithm (one of our top-performing methods), tunes a live database in real-time to minimize $C_{DRAM}$ and $V_{SLA}$. Compared with a simulated environment, real-time tuning introduces additional noise, making it more challenging for the agent to take appropriate actions to meet the latency target.

Figure 5.8 illustrates the online tuning process for a Sysbench workload (50 tables $\times$ 1M rows) with a 20 ms latency goal. We change thread counts and access patterns periodically to mimic production environments. The red line depicts the buffer-pool size; the blue line is the 10-second moving average of latency; black dotted lines indicate workload shifts. We run this on the same machine used for training, and for demonstration set MicroTune's buffer-size update interval to 30s, production deployments would use a lower frequency for stability.

**Phase 1: Low Workload (2 threads, Gaussian distribution)** Initially, the workload runs with 2 threads under a Gaussian distribution, and the buffer pool is initialized at 8 GB. The resulting latency is far below the 20 ms target, indicating that the buffer pool is over-allocated. The agent recognizes this and gradually downsizes the buffer until it stabilizes around 4 GB. Although some outliers briefly exceed 20 ms during the transition, these short spikes are normal in a live system subject to background noise.

**Phase 2: Higher Workload (8 threads)** Next, the number of threads is increased to 8 to stress the database further. The latency begins to exceed the target more frequently, prompting the agent to increase the buffer pool size. As the buffer grows, the latency stabilizes again below 20 ms.

**Phase 3: Moderate Workload (4 threads)** Then, the workload concurrency is reduced to 4 threads. The agent detects that the system can operate with a smaller buffer and reduces the size from approximately 4.8 GB to 4 GB without incurring SLA violations.

**Phase 4: Different Access Pattern (3 threads, Uniform distribution)** The workload is then switched to 3 threads with a uniform distribution, which is more intensive than the Gaussian pattern. Latency immediately spikes above the threshold, and the agent responds by increasing the buffer pool to 8 GB. This brings latency back under 20 ms, demonstrating the agent's ability to react to heavier workloads.

**Final Phase: Lighter Workload (2 threads, Special distribution)** Finally, the workload is lowered to 2 threads with a "special" distribution, resulting in a relatively light load. The agent observes decreased demand and reduces the buffer size accordingly.

Overall, this live demonstration confirms the agent's capability to handle dynamic changes in workload while keeping latency within acceptable bounds and reducing the buffer pool size when possible. Despite occasional short-lived spikes, the system remains under 20 ms for the vast majority of operations, underscoring the practicality of our real-time tuning approach.

## 5.6 Conclusion

In this chapter, we introduced MicroTune, an online tuner that continuously monitors database metrics and optimizes memory usage while respecting SLA constraints. We formally stated the problem of optimizing the buffer size under SLA constraints and an Oracle is deduced from it. Our goal is to use RL to reach the buffer size of the Oracle for each workload. Rather than learning the policy online, which is expensive in time and may raise safety issues, we start with data collection for workloads with various characteristics, which enables us to evaluate the optimal buffer size for each workload. Then, we defined an optimal policy (target policy) that can be computed offline on the data, which allows us to reach Oracle's performance step by step. The data allows us to train the RL algorithms, while the optimal policy allows us to perform hyperparameter tuning to select the best-trained policy. Then, the best-trained policy can be used in real time to tune the buffer size of a DBMS under SLA constraints. We defined some baselines, which could be used as an alternative to MicroTune, and we tested the state-of-the-art reinforcement learning algorithms. The results coming from our experiments are clear: while RL algorithms do not directly use latency, they dominate all baselines, even those that use it. Finally, we selected the best-trained policy, based on A2C, and we successfully tested it in real-time using realistic workloads that change over time.

While these initial results are promising, they can be improved in many directions. One of the main limitations of our approach relies on the target policy. Indeed, increasing the size of the action set will mechanically lead to faster reaching the buffer size chosen by the Oracle, and hence decreasing the SLA violations of the optimal policy. Increasing the set of actions also increases the policy space and hence necessitates more data to learn robust policies. Data augmentation could be reached using workloads of different origins, from synthetic benchmarks such as TPC-C [50] to real-world production workloads. For collecting real-world production data, we plan to deploy MicroTune on some production platforms for logging data that allows performing counterfactual learning of policies [138].

# Chapter 6

# Conclusion & Future Work

This final chapter reflects on the work presented in this thesis and outlines possible future work. Having examined both the practical barriers to DBMS automation and the algorithmic opportunities for more adaptive tuning, we now step back to consolidate the contributions and highlight their broader implications.

We begin by summarizing the thesis's contributions: a qualitative study that reveals why adoption of automation remains limited in practice, and two algorithmic systems, DOT for dynamic knob selection and MicroTune for reinforcement-learning-based resource control, which tackle pressing challenges in DBMS tuning and resource allocation. Building on this foundation, we then turn to future perspectives. In the short term, we discuss concrete steps to make our systems more robust, portable, and production-ready. In the longer term, we outline directions toward unified, interpretable, and generalizable autonomic database management.

This chapter serves both as a synthesis of what has been achieved and as a blueprint for future research into trustworthy, adaptive, and efficient database systems.

## 6.1 Summary of Contributions

Our qualitative study, based on in-depth interviews with experienced DBAs from industry, revealed that while DBMS automation is viewed positively in principle, its adoption is hindered by several technical and organizational concerns. These include the opacity of tuning algorithms, lack of workload generalizability, risk of performance regressions, and the significant manual effort still required to supervise automated tools. These findings offer actionable guidelines for tool developers, including the need for transparent decision processes, workload-specific adaptability, and mechanisms that preserve human oversight.

In response to these findings, we proposed two complementary systems. First, DOT, a dynamic DBMS auto-tuning framework that eliminates the need for expensive pre-tuning phases. DOT leverages online sampling, *Recursive Feature Elimination with Cross Validation*, *Bayesian Optimization*, and a *Likelihood Ratio Test* policy to iteratively refine its search space, enabling it to converge rapidly on effective configurations even under limited prior knowledge. Our experiments demonstrate that DOT achieves state-of-the-art tuning quality while reducing overhead by up to 73.6% compared to traditional BO-based methods.

Second, MicroTune addresses the common issue of resource over-provisioning in DBMS deployments, particularly RAM adjustment in real-time. Industrial observations showed that 20–48% of allocated memory is often unused, suggesting massive optimization potential. MicroTune uses offline-trained *Reinforcement Learning* policies to dynamically adjust buffer sizes in real-time, ensuring SLA compliance while reducing memory waste. Extensive experimentation across synthetic and realistic workloads validated its

performance and robustness against conventional baselines, including static configurations and latency-aware heuristics.

Together, these contributions advance the vision of more autonomous, efficient, and trustworthy database systems.

## 6.2 Perspectives

### 6.2.1 Short-Term Goals

Several short-term directions can build directly upon this work. These steps are primarily aimed at increasing robustness, portability, and immediate practical applicability of DOT and MicroTune.

- **Broader Workload Coverage**: A key limitation of our current evaluation is that it is restricted to a small set of synthetic and benchmarked workloads. Extending the evaluation to industry-standard benchmarks such as YCSB [139] and TATP [140] would provide a more diverse set of access patterns and performance characteristics. Beyond benchmarks, evaluation on production-like traces—such as enterprise customer relationship management queries, financial transaction logs, or telecom call-data workloads—would stress-test robustness under skew, diurnal patterns, and long-tail queries. Furthermore, porting our systems beyond MySQL to other popular DBMSs such as PostgreSQL, MariaDB, and CockroachDB would validate their portability and generality. This cross-system evaluation would also allow us to examine how DBMS-specific architectures and query optimizers interact with tuning strategies.

- **Toward Deployment in Production Pipelines**: To move beyond controlled lab settings, both DOT and MicroTune should be tested in cloud-native deployments where elasticity, failures, and workload variability are the norm. For example, embedding these tools into Kubernetes-based pipelines would allow us to study their responsiveness to container restarts, resource contention, and scaling events. Integrating with observability stacks (e.g., Prometheus [141], Grafana [82]) would further enable automatic correlation between tuning actions and application-level KPIs such as query latency percentiles or SLA compliance. Such integration would provide insights into fault tolerance, responsiveness under load spikes, and compatibility with DevOps workflows.

- **More Adaptive Knob Expansion Policies in DOT**: Currently, DOT relies on a likelihood-ratio test based knob expansion strategy. This could be made more adaptive by incorporating budget-aware heuristics (e.g., stopping exploration once marginal utility falls below a threshold) or workload-sensitive policies (e.g., adjusting exploration rate based on workload). More advanced methods such as contextual multi-armed bandits could dynamically allocate exploration effort to knobs most likely to impact SLA-critical metrics. By embedding these adaptive heuristics, DOT can strike a better balance between exploration cost and tuning efficiency.

- **Counterfactual Learning for MicroTune**: Our current RL policies are trained with controlled workloads and limited real-world traces. A promising next step is to leverage logged traces from production systems and apply counterfactual reasoning to simulate alternative actions. This would allow us to train and validate policies safely offline, reducing the risks of SLA violations during deployment. Techniques such as inverse propensity scoring or doubly robust estimators could provide

statistically sound offline evaluation, while also enabling "what-if" analysis where DBAs can explore hypothetical tuning actions without incurring live performance penalties.

- **Smarter Action Granularity in MicroTune**: At present, buffer pool resizing actions are coarse-grained, which may lead to abrupt changes in resource allocation and occasional SLA violations. Increasing the resolution of the action space (e.g., smaller buffer size increments) could yield smoother adjustments. However, finer granularity must be carefully balanced with data availability and policy complexity to avoid overfitting. One promising avenue is hierarchical reinforcement learning, where a high-level agent decides when to adjust resources, and a low-level agent determines the precise increment. This could achieve both robustness and precision in adaptive memory management.

These short-term tasks aim to enhance the usability, reliability, and adaptability of our systems, bridging the gap between controlled research prototypes and tools ready for production deployment.

### 6.2.2 Long-Term Goals

In the long run, this thesis contributes toward the broader vision of self-managing database systems. Several strategic directions emerge:

- **Unified Autonomic DBMS Architecture**: A natural extension of this work is to integrate DOT and MicroTune into a holistic, closed-loop autonomic controller that jointly manages configuration tuning and resource allocation. Such a unified framework could continuously monitor workloads, predict performance bottlenecks, and trigger both knob adjustments and memory reallocation in tandem. By coordinating across CPU, memory, and I/O resources, this architecture would enable true end-to-end optimization, moving closer to the long-standing goal of fully self-managing DBMSs.

- **Multi-Agent Reinforcement Learning for DBMS**: Current RL-based approaches optimize a single resource dimension, but real-world tuning is inherently multi-objective and involves trade-offs across several resources. Extending MicroTune to a *multi-agent reinforcement learning* (MARL) framework would allow different agents, one for memory, one for CPU, one for I/O, to coordinate their policies while respecting shared system constraints. This would capture the complex interdependencies across resources and enable more balanced, efficient optimization strategies, potentially reducing both cost and latency simultaneously.

- **Trustworthy and Interpretable Automation**: For automated tuning to be widely adopted in practice, it must earn the trust of DBAs and system operators. Inspired by feedback from our user study, future work should focus on developing interpretable models and transparent decision-making. For example, reinforcement learning agents could be augmented with feature attribution methods to explain why a particular knob adjustment was chosen, and visual dashboards or natural language rationales could present these explanations in an accessible way. Such mechanisms would bridge the gap between black-box automation and human oversight, fostering confidence in deploying these systems at scale.

- **Cross-Instance and Cross-Workload Generalization**: A major barrier to practical auto-tuning is that models trained in one environment often fail to transfer to

another, requiring costly retraining. Techniques from meta-learning and domain adaptation offer a promising path forward, enabling systems like DOT and MicroTune to quickly adapt to new hardware profiles, workloads, or DBMS engines with minimal additional sampling. Achieving such generalization would significantly reduce deployment costs and improve portability, allowing autotuners to be applied across heterogeneous cloud and on-premise environments with little manual intervention.

- **Standardized Benchmarking for Adaptive DBMS**: Finally, as adaptive tuning techniques proliferate, the community lacks common benchmarks and evaluation protocols. Establishing standardized datasets and workloads, particularly ones that capture non-stationary behavior such as workload shifts, hardware heterogeneity, and failure scenarios, would allow fairer comparisons and reproducibility. Such benchmarks would not only accelerate research progress but also provide practitioners with clearer guidance on which methods are reliable under realistic deployment conditions.

These directions lie at the intersection of systems, machine learning, and human-computer interaction, and offer a compelling blueprint for future research into intelligent, adaptive data management.

## 6.3 Closing Remarks

DBMSs remain a foundational component of modern data infrastructure, yet their administration continues to demand extensive expertise and manual intervention. This thesis represents a step toward reducing that burden by pairing qualitative insight with intelligent automation. By aligning algorithmic design with real-world constraints and DBAs expectations, we hope to contribute not only to more efficient systems but also to more usable and trustworthy tools that support the evolving role of the database administrator in an increasingly automated world.

# Bibliography


[1] Statista. Worldwide data created. https://www.statista.com/statistics/871513/worldwide-data-created/, 2024. [Accessed: 12-03-2025].

[2] SAP AG. SAP MaxDB – Low TCO for SAP Applications. In *Expert Sessions, SAP TechEd 2008*, 2008.

[3] Jian Tan, Tieying Zhang, Feifei Li, Jie Chen, Qixing Zheng, Ping Zhang, Honglin Qiao, Yue Shi, Wei Cao, and Rui Zhang. ibtune: individualized buffer tuning for large-scale cloud databases. *Proc. VLDB Endow.*, 12(10):1221–1234, June 2019.

[4] Oracle autonomous database. https://www.oracle.com/autonomous-database/, 2024. [Accessed 12-03-2024].

[5] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd C Mowry, Matthew Perron, Ian Quah, et al. Self-driving database management systems. In *CIDR*, volume 4, page 1, 2017.

[6] Tim Kraska, Mohammad Alizadeh, Alex Beutel, Ed H. Chi, Jialin Ding, Ani Kristo, Guillaume Leclerc, Samuel Madden, Hongzi Mao, and Vikram Nathan. Sagedb: A learned database system. 2019.

[7] Crimsondb: A self-designing key-value store. https://demosubmitter.github.io/, 2017. [Accessed 12-01-2024].

[8] Adam Storm, Christian Garcia-Arellano, Sam Lightstone, Yixin Diao, and Maheswaran Surendra. Adaptive self-tuning memory in DB2. pages 1081–1092, 01 2006.

[9] Wenhu Tian, Patrick Martin, and Wendy Powley. Techniques for automatically sizing multiple buffer pools in DB2. pages 294–302, 01 2003.

[10] Surajit Chaudhuri and Vivek Narasayya. Self-tuning database systems: A decade of progress. pages 3–14, 01 2007.

[11] D. Narayanan, E. Thereska, and Anastasia Ailamaki. Continuous resource monitoring for self-predicting DBMS. volume 2005, pages 239– 248, 10 2005.

[12] Junxiong Wang, Immanuel Trummer, and Debabrota Basu. UDO: universal database optimization using reinforcement learning. *CoRR*, abs/2104.01744, 2021.

[13] Ji Zhang, Ke Zhou, Guoliang Li, Yu Liu, Ming Xie, Bin Cheng, and Jiashu Xing. CDBTune+: An efficient deep reinforcement learning-based automatic cloud database tuning system. *The VLDB Journal*, 30, 11 2021.

[14] Orange S.A. Orange: Global telecommunications operator. https://www.orange.com/, 2025. [Accessed: 2025-07-09].

[15] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd Mowry, Matthew Perron, Ian Quah, Siddharth Santurkar, Anthony Tomasic, Skye Toor, Dana Van Aken, Ziqi Wang, Yingjun Wu, Ran Xian, and Tieying Zhang. Self-driving database management systems. In *CIDR 2017, Conference on Innovative Data Systems Research*, 2017.

[16] Carnegie Mellon University Database Group. Cmu database group. https://db.cs.cmu.edu/, 2025. [Accessed: 2025-07-15].

[17] Noisepage. https://noise.page/about/, 2020. [Accessed: 2023-01-28].

[18] Crimsondb. https://demosubmitter.github.io/, 2018. [Accessed: 2023-01-28].

[19] Azure sql database. https://azure.microsoft.com/en-us/products/azure-sql/database/. [Accessed: 2025-08-18].

[20] Amazon aurora. https://aws.amazon.com/rds/aurora/. [Accessed: 2025-08-18].

[21] Amazon Web Services. Amazon devops guru for rds. https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/devops-guru-for-rds.html, 2025. [Accessed: 2025-09-07].

[22] Ibm db2 design advisor. https://www.ibm.com/docs/en/db2/11.5?topic=tools-design-advisor. [Accessed: 2025-08-18].

[23] Adam J. Storm, Christian Garcia-Arellano, Sam S. Lightstone, Yixin Diao, and M. Surendra. Adaptive self-tuning memory in DB2. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, VLDB '06, page 1081–1092. VLDB Endowment, 2006.

[24] Oracle. Autonomous database pricing. https://www.oracle.com/autonomous-database/pricing/, 2025. [Accessed: 2025-07-16].

[25] Junxiong Wang, Immanuel Trummer, and Debabrota Basu. Udo: universal database optimization using reinforcement learning. *Proceedings of the VLDB Endowment*, 14:3402–3414, 09 2021.

[26] Yuqing Zhu, Jianxun Liu, Mengying Guo, Yungang Bao, Wenlong Ma, Zhuoyue Liu, Kunpeng Song, and Yingchun Yang. Bestconfig: tapping the performance potential of systems via automatic configuration tuning. In *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17. ACM, September 2017.

[27] Vamsidhar Thummala and Shivnath Babu. ituned: a tool for configuring and visualizing database parameters. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, page 1231–1234, New York, NY, USA, 2010. Association for Computing Machinery.

[28] Biplob Debnath, David Lilja, and Mohamed Mokbel. Sard: A statistical approach for ranking database tuning parameters. pages 11–18, 05 2008.

[29] Dana Van Aken, Andrew Pavlo, Geoffrey Gordon, and Bohan Zhang. Automatic database management system tuning through large-scale machine learning. pages 1009–1024, 05 2017.

[30] Baoqing Cai, Yu Liu, Ce Zhang, Guangyu Zhang, Ke Zhou, Li Liu, Chunhua Li, Bin Cheng, Jie Yang, and Jiashu Xing. Hunter: An online cloud database hybrid tuning system for personalized requirements. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 646–659, New York, NY, USA, 2022. Association for Computing Machinery.

[31] L. Breiman, J. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Chapman and Hall/CRC, 1st edition, 1984.

[32] Immanuel Trummer. Db-bert: A database tuning tool that "reads the manual". In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 190–203, New York, NY, USA, 2022. Association for Computing Machinery.

[33] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 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)*, pages 4171–4186, 2019.

[34] Jiale Lao, Yibo Wang, Yufei Li, Jianping Wang, Yunjia Zhang, Zhiyuan Cheng, Wanghu Chen, Mingjie Tang, and Jianguo Wang. Gptuner: A manual-reading database tuning system via gpt-guided bayesian optimization. *Proceedings of the VLDB Endowment*, 17(8):1939–1952, April 2024.

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

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

[37] Yifan Wang, Pierre Bourhis, Romain Rouvoy, and Patrick Royer. Challenges & opportunities in automating DBMS: A qualitative study. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ASE '24, page 2013–2023, New York, NY, USA, 2024. Association for Computing Machinery.

[38] Xinyi Zhang, Hong Wu, Zhuo Chang, Shuowei Jin, Jian Tan, Feifei Li, Tieying Zhang, and Bin Cui. Restune: Resource oriented tuning boosted by meta-learning for cloud databases. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, page 2102–2114, New York, NY, USA, 2021. Association for Computing Machinery.

[39] Rukai Wei, Yu Liu, Yufeng Hou, Heng Cui, Yongqiang Zhang, and Ke Zhou. Toptune: Tailored optimization for categorical and continuous knobs towards accelerated and improved database performance tuning. In *IEEE 41st International Conference on Data Engineering (ICDE)*, pages 613–626, 2025.

[40] Jiahui Li, Junhao Ye, Yuren Mao, Yunjun Gao, and Lu Chen. Loftune: A low-overhead and flexible approach for Spark sql configuration tuning. *IEEE Transactions on Knowledge and Data Engineering*, 37(6):3528–3542, 2025.

[41] Alexander Bianchi, Andrew Chai, Vincent Corvinelli, Parke Godfrey, Jarek Szlichta, and Calisto Zuzarte. Db2une: Tuning under pressure via deep learning. *Proceedings of the VLDB Endowment*, 17(12):3855–3868, 2024.

[42] Taiyi Wang, Liang Liang, Guang Yang, Thomas Heinis, and Eiko Yoneki. A new paradigm in tuning learned indexes: A reinforcement learning enhanced approach. *Proceedings of the ACM on Management of Data (SIGMOD)*, 3(3):Article 120, 1–26, 2025.

[43] Wenwen Sun, Zhicheng Pan, Zirui Hu, Yu Liu, Chengcheng Yang, Rong Zhang, and Xuan Zhou. Rabbit: Retrieval-augmented generation enables better automatic database knob tuning. In *IEEE 41st International Conference on Data Engineering (ICDE)*, pages 3807–3820, 2025.

[44] Chongjiong Fan, Zhicheng Pan, Wenwen Sun, Chengcheng Yang, and Wei-Neng Chen. Latuner: An llm-enhanced database tuning system based on adaptive surrogate model. In *Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2024)*, volume 14945 of *Lecture Notes in Artificial Intelligence*, pages 372–388. Springer, 2024.

[45] Theresa Eimer, Marius Lindauer, and Roberta Raileanu. Hyperparameters in reinforcement learning and how to tune them. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 9104–9149. PMLR, 23–29 Jul 2023.

[46] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Commun. ACM*, 56(2):74–80, February 2013.

[47] Theophilus A Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Understanding data center traffic characteristics. In *ACM SIGCOMM Workshop: Research on Enterprise Networking*. Association for Computing Machinery, Inc., January 2009.

[48] Younggyun Koh, Rob Knauerhase, Paul Brett, Mic Bowman, Zhihua Wen, and Calton Pu. An analysis of performance interference effects in virtual environments. In *2007 IEEE International Symposium on Performance Analysis of Systems & Software*, pages 200–209, 2007.

[49] Alexey Kopytov. Sysbench: A system performance benchmark. https://github.com/akopytov/sysbench, 2024. [Accessed: 2024-07-15].

[50] TPC Professional Affiliates. TPC-C is an on-line transaction processing benchmark. https://www.tpc.org/tpcc/, 1992.

[51] Bilge Acun, Phil Miller, and Laxmikant V. Kale. Variation among processors under turbo boost in hpc systems. In *Proceedings of the 2016 International Conference on Supercomputing*, ICS '16, New York, NY, USA, 2016. Association for Computing Machinery.

[52] Zakaria Ournani, Mohammed Chakib Belgaid, Romain Rouvoy, Pierre Rust, Joel Penhoat, and Lionel Seinturier. Taming energy consumption variations in systems benchmarking. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, ICPE '20, page 36–47, New York, NY, USA, 2020. Association for Computing Machinery.

[53] Sasalak Tongkaw and Aumnat Tongkaw. A comparison of database performance of MariaDB and MySQL with OLTP workload. In *2016 IEEE Conference on Open Systems (ICOS)*, pages 117–119, 2016.

[54] X. Kong. Research on performance optimization of OLTP systems based on innoDB. *Journal of Theoretical and Applied Information Technology*, 46:638–642, 12 2012.

[55] Transaction Processing Performance Council. Tpc-h benchmark. http://www.tpc.org/tpch/, 2016. [Accessed: 24-03-2025].

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

[57] B. L. Welch. The generalization of 'student's' problem when several different population variances are involved. *Biometrika*, 34(1/2):28–35, 1947.

[58] Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937.

[59] Katarina Grolinger and Miriam A. M. Capretz. Autonomic database management: State of the art and future trends. In *Proceedings of the ISCA 27th International Conference on Computers and Their Applications (CATA 2012)*, pages 190–195, March 2012.

[60] Justin Smith, Brittany Johnson, Emerson Murphy-Hill, Bill Chu, and Heather Richter Lipford. How developers diagnose potential security vulnerabilities with a static analysis tool. *IEEE Transactions on Software Engineering*, 45(9):877–897, 2019.

[61] Zakaria Ournani, Romain Rouvoy, Pierre Rust, and Joel Penhoat. On reducing the energy consumption of software: From hurdles to requirements. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–12, 2020.

[62] Sarra Habchi, Xavier Blanc, and Romain Rouvoy. On adopting linters to deal with performance concerns in android apps. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 6–16, 2018.

[63] B.G. Glaser and A.L. Strauss. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Observations (Chicago, Ill.). Aldine Transaction, 1967.

[64] Kathy Charmaz. *Constructing grounded theory: A practical guide through qualitative analysis*. sage, 2006.

[65] Steve Adolph, Wendy Hall, and Philippe Kruchten. Using grounded theory to study the experience of software development. *Empirical Software Engineering*, 16:487–513, 08 2011.

[66] Klaas-Jan Stol, Paul Ralph, and Brian Fitzgerald. Grounded theory in software engineering research: A critical review and guidelines. 05 2016.

[67] Titus Barik, Brittany Johnson, and Emerson Murphy-Hill. I heart hacker news: expanding qualitative research findings by analyzing social news websites. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, page 882–885, New York, NY, USA, 2015. Association for Computing Machinery.

[68] Percona xtradb cluster. https://www.percona.com/software/mysql-database/percona-xtradb-cluster, 2024. [Accessed 13-02-2024].

[69] Redis. https://redis.io/, 2024. [Accessed 13-02-2024].

[70] Mongodb. https://www.mongodb.com/, 2024. [Accessed 13-02-2024].

[71] Oracle database. https://www.oracle.com, 2024. [Accessed 13-02-2024].

[72] Mariadb. https://mariadb.org/, 2024. [Accessed 13-02-2024].

[73] Postgresql. https://www.postgresql.org/, 2024. [Accessed 13-02-2024].

[74] Cassandra. https://cassandra.apache.org/, 2024. [Accessed 13-02-2024].

[75] Elasticsearch. https://www.elastic.co/, 2024. [Accessed 13-02-2024].

[76] S.E. Hove and Bente Anda. Experiences from conducting semi-structured interviews in empirical software engineering research. volume 2005, pages 10 pp.–, 10 2005.

[77] Thomas Fritz and Gail Murphy. Determining relevancy: How software developers determine relevant information in feeds. pages 1827–1830, 05 2011.

[78] Zakaria Ournani, Romain Rouvoy, Pierre Rust, and Joel Penhoat. On reducing the energy consumption of software: From hurdles to requirements. In *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ESEM '20, New York, NY, USA, 2020. Association for Computing Machinery.

[79] Konstantina Vasileiou, Julie Barnett, Susan Thorpe, and Terry Young. Characterising and justifying sample size sufficiency in interview-based studies: systematic analysis of qualitative health research over a 15-year period. *BMC medical research methodology*, 18:1–18, 2018.

[80] Daniel Oliver, Julianne Serovich, and Tina Mason. Constraints and opportunities with interview transcription: Towards reflection in qualitative research. *Social forces; a scientific medium of social study and interpretation*, 84:1273–1289, 01 2006.

[81] Kubernetes. https://kubernetes.io/, 2024. [Accessed 12-11-2023].

[82] Grafana. https://grafana.com/, 2024. [Accessed: 8-4-2024].

[83] Monolog. https://github.com/waza-ari/monolog-mysql, 2020. [Accessed 12-6-2024].

[84] Terraform. https://www.terraform.io/, 2024. [Accessed 8-4-2024].

[85] Ansible. https://www.ansible.com/, 2024. [Accessed 8-4-2024].

[86] Hashicorp. https://www.hashicorp.com/, 2024. [Accessed 8-4-2024].

[87] Optimizer with oracle database 12c release 2. https://www.oracle.com/technetwork/database/bi-datawarehousing/twp-optimizer-with-oracledb-12c-1963236.pdf, 2017. [Accessed 8-8-2023].

[88] Asmaa Sallam, Elisa Bertino, Syed Rafiul Hussain, David Landers, R. Michael Lefler, and Donald Steiner. Dbsafe—an anomaly detection system to protect databases from exfiltration attempts. *IEEE Systems Journal*, 11(2):483–493, 2017.

[89] Sainan Li, Qilei Yin, Guoliang Li, Qi Li, Zhuotao Liu, and Jinwei Zhu. Unsupervised contextual anomaly detection for database systems. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 788–802, New York, NY, USA, 2022. Association for Computing Machinery.

[90] Yi Hu and B. Panda. Identification of malicious transactions in database systems. In *Seventh International Database Engineering and Applications Symposium, 2003. Proceedings.*, pages 329–335, 2003.

[91] Doyup Lee. Anomaly detection in multivariate non-stationary time series for automatic DBMS diagnosis. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 412–419, 2017.

[92] Surajit Chaudhuri and Vivek Narasayya. Autoadmin "what-if" index analysis utility. *SIGMOD Rec.*, 27(2):367–378, jun 1998.

[93] Lin Ma, Dana Van Aken, Ahmed Hefny, Gustavo Mezerhane, Andrew Pavlo, and Geoffrey Gordon. Query-based workload forecasting for self-driving database management systems. pages 631–645, 05 2018.

[94] Francisco J. Baldan, Sergio Ramirez-Gallego, Christoph Bergmeir, Francisco Herrera, and Jose M. Benitez. A forecasting methodology for workload forecasting in cloud systems. *IEEE Transactions on Cloud Computing*, 6(4):929–941, 2018.

[95] Archana Ganapathi, Harumi Kuno, Umeshwar Dayal, Janet L. Wiener, Armando Fox, Michael Jordan, and David Patterson. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In *2009 IEEE 25th International Conference on Data Engineering*, pages 592–603, 2009.

[96] Aws amazon autoscaling. https://aws.amazon.com/en/autoscaling/, 2023. [Accessed 8-5-2024].

[97] Overview of autoscale with azure virtual machine scale sets. https://learn.microsoft.com/en-us/azure/virtual-machine-scale-sets/virtual-machine-scale-sets-autoscale-overview, 2024. [Accessed 8-5-2024].

[98] Oracle cloud infrastructure documentation autoscaling. https://docs.oracle.com/en-us/iaas/Content/Compute/Tasks/autoscalinginstancepools.htm, 2024. [Accessed 8-5-2024].

[99] Guoliang Li, Xuanhe Zhou, Shifu Li, and Bo Gao. Qtune: A query-aware database tuning system with deep reinforcement learning. *Proc. VLDB Endow.*, 12(12):2118–2130, aug 2019.

[100] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, et al. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In *Proceedings of the 2019 International Conference on Management of Data*, pages 415–432, 2019.

[101] Percona-Lab. tpcc-mysql. https://github.com/Percona-Lab/tpcc-mysql. [Accessed: 2025-05-30].

[102] Electrum. tpch-dbgen. https://github.com/electrum/tpch-dbgen. Accessed: 2025-05-30.

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

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

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

[106] 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. Bayesian hyperparameter optimization for deep neural network-based network intrusion detection. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 5413–5419. IEEE, 2021.

[107] Tong Yu and Hong Zhu. Hyper-parameter optimization: A review of algorithms and applications. *CoRR*, abs/2003.05689, 2020.

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

[109] Adi Zaenul Mustaqim, Sumarni Adi, Yoga Pristyanto, and Yuli Astuti. 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)*, pages 270–275. IEEE, 2021.

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

[111] HustAIsGroup. CDBTune: An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning. GitHub repository. [Accessed: 2025-06-20].

[112] Immanuel Trummer. dbbert: DB-BERT Database Tuning Tool Implementation. GitHub repository. [Accessed: 2025-06-20].

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

[114] Shipra Agrawal and Navin Goyal. Analysis of Thompson sampling for the multi-armed bandit problem. In Shie Mannor, Nathan Srebro, and Robert C. Williamson, editors, *Proceedings of the 25th Annual Conference on Learning Theory*, volume 23 of *Proceedings of Machine Learning Research*, pages 39.1–39.26, Edinburgh, Scotland, 25–27 Jun 2012. PMLR.

[115] Oracle. InnoDB Buffer Pool, MySQL 8.4. https://dev.mysql.com/doc/refman/8.4/en/innodb-buffer-pool.html. [Accessed: 2025-03-04].

[116] Robert Lasch, Thomas Legler, Norman May, Bernhard Scheirle, and Kai-Uwe Sattler. Cooperative memory management for table and temporary data. In *Proceedings of the 1st Workshop on Simplicity in Management of Data*, SiMoD '23, New York, NY, USA, 2023. Association for Computing Machinery.

[117] Laurens Kuiper, Peter Boncz, and Hannes Muhleisen. Robust External Hash Aggregation in the Solid State Age . In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 3753–3766, Los Alamitos, CA, USA, May 2024. IEEE Computer Society.

[118] Riki Otaki, Jun Hyuk Chang, Charles Benello, Aaron J. Elmore, and Goetz Graefe. Resource-adaptive query execution with paged memory management. In *Proceedings of the 15th International Conference on Very Large Data Bases (VLDB)*, 2025.

[119] Vikram Nathan, Vikramank Y. Singh, Zhengchun Liu, Mohammad Rahman, Andreas Kipf, Dominik Horn, Davide Pagano, Gaurav Saxena, Balakrishnan (Murali) Narayanaswamy, and Tim Kraska. Intelligent scaling in amazon redshift. 2024.

[120] Deepak Ravikumar, Alex Yeo, Yiwen Zhu, Aditya Lakra, Harsha Nagulapalli, Santhosh Kumar Ravindran, Steve Suh, Niharika Dutta, Andrew Fogarty, Yoonjae Park, Sumeet Khushalani, Arijit Tarafdar, Kunal Parekh, and Subru Krishnan. Intelligent pooling: Proactive resource provisioning in large-scale cloud service. *PVLDB*.

[121] Oracle Corporation. *Oracle Autonomous Database on Oracle Cloud Infrastructure*. Oracle Corporation, 2025. [Accessed: 01-09-2025].

[122] Richard S Sutton, Andrew G Barto, et al. Reinforcement learning. *Journal of Cognitive Neuroscience*, 11(1):126–134, 1999.

[123] Yaniv Gur, Dongsheng Yang, Frederik Stalschus, and Berthold Reinwald. Adaptive multi-model reinforcement learning for online database tuning. In *EDBT*, pages 439–444, 2021.

[124] German Aerospace Center (DLR) Institute of Robotics and Mechatronics (RM) open source projects. Stable-baselines3 docs - reliable reinforcement learning implementations, 2025.

[125] Kubernetes Documentation. Horizontal pod autoscaler, 2025. [Accessed: 2025-01-07].

[126] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670, 2010.

[127] David Delande, Patricia Stolf, Raphaël Feraud, Jean-Marc Pierson, and André Bottaro. Horizontal scaling in cloud using contextual bandits. In *European Conference on Parallel Processing*, pages 285–300. Springer, 2021.

[128] Kenneth Foo and Fangwei. Contextual bandit: Linear upper confidence bound disjoint (linucb disjoint) algorithm, 2020.

[129] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018.

[130] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

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

[132] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[133] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[134] Dennis Shasha, Philippe Bonnet, and Nancy Hartline Bercich. Database tuning principles, experiments, and troubleshooting techniques. *SIGMOD Rec.*, 33(2):115–116, jun 2004.

[135] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. *CoRR*, abs/1907.10902, 2019.

[136] Shuhei Watanabe. Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance, 2023.

[137] Abhishek Gupta, Aldo Pacchiano, Yuexiang Zhai, Sham Kakade, and Sergey Levine. Unpacking reward shaping: Understanding the benefits of reward engineering on sample complexity. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 15281–15295. Curran Associates, Inc., 2022.

[138] Houssam Zenati, Eustache Diemert, Matthieu Martin, Julien Mairal, and Pierre Gaillard. Sequential counterfactual risk minimization. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23, 2023.

[139] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC)*, pages 143–154. ACM, 2010.

[140] Patrick O'Neil, Elisabeth O'Neil, and X. Chen. The telecommunication application transaction processing benchmark (TATP). http://tatpbenchmark.sourceforge.net/, 2009. Specification and benchmark kit.

[141] Prometheus. https://prometheus.io/. [Accessed: 22-07-2024].