
A Reinforcement Learning Framework for Autonomous Cloud Database Optimization

K. Pardhasaradhi, MCA

Article Received:10-10-2025 Article Modified:25-11-2025

Article Accepted:26-11-2025 Article Published:28-11-2025

DOI:10.37854/UIJMR.2025.3.4.102

Abstract

Cloud database systems must continuously adapt their configuration, resource allocation, and query-execution behavior to highly variable workloads. Manual tuning is costly, reactive, and difficult to scale across multi-tenant environments. This paper proposes AutoDBO, a reinforcement learning (RL) framework for autonomous cloud database optimization that formulates database administration as a constrained Markov decision process (CMDP). The framework integrates workload representation, safety-aware action masking, a constrained actor-critic policy, and a closed-loop telemetry-driven controller. AutoDBO jointly selects database configuration knobs, buffer/cache adjustments, and compute-resource scaling actions while enforcing latency, availability, and cost constraints. We evaluate the framework using a simulation-based cloud database environment driven by TPC-C-like OLTP and TPC-H-like analytical workload traces. Compared with default configuration, static expert tuning, Bayesian optimization, and an unconstrained deep reinforcement learning baseline, AutoDBO reduces average query latency by 38.6%, improves throughput by 41.2%, lowers cloud cost by 17.8%, and reduces constraint violations by 86.4% relative to the unconstrained RL baseline. The results indicate that safety-aware RL can provide a practical path toward autonomous, cost-efficient cloud database operation.

Keywords--Autonomous databases; cloud computing; reinforcement learning; database tuning; constrained Markov decision processes; deep reinforcement learning; workload adaptation; resource management.

I. INTRODUCTION

Modern cloud database platforms host workloads whose intensity, query mix, data volume, and concurrency can change abruptly. A retail application may experience a sudden promotional traffic spike, while an analytical service may alternate between short interactive dashboards and long batch queries. In such environments, a fixed database configuration rarely remains optimal. Database administrators must adjust memory buffers, connection pools, parallelism, storage parameters, indexes, and virtual-machine resources, often while balancing latency, throughput, availability, and monetary cost.

Manual optimization has three major limitations. First, modern database engines expose hundreds of configuration parameters, many of which interact in non-obvious ways. Second, cloud environments introduce external control dimensions, including instance type, storage tier, replica count, and autoscaling policy. Third, tuning decisions must satisfy operational constraints: a configuration that improves throughput but violates a service-level objective (SLO) or causes instability is unacceptable in production.

Machine learning has emerged as a promising approach to database tuning. OtterTune demonstrated that historical tuning data can be reused to recommend strong configurations for new workloads [1], [2]. CDBTune formulated cloud database tuning as a deep reinforcement learning problem and used deep deterministic policy gradient (DDPG) to search a high-dimensional continuous knob space [3]. QTune further introduced query-aware tuning with deep RL [4]. However, many existing systems focus primarily on performance improvement and treat safety, cost, and long-horizon adaptation as secondary concerns.

This paper presents AutoDBO, an RL framework for autonomous optimization of cloud databases. Unlike prior knob-tuning systems, AutoDBO treats optimization as a closed-loop control problem over both database-internal parameters and cloud-level resources. It uses a constrained policy that explicitly penalizes SLO violations, unstable actions, and excessive cost. The main contributions are:

- A unified CMDP formulation for cloud database optimization that includes configuration knobs, cache and connection controls, and resource-scaling actions.
- A workload-aware state representation combining query latency, throughput, resource utilization, queueing indicators, workload embeddings, and forecasted demand.
- A constrained soft actor-critic (CSAC) learning algorithm with action masking, conservative exploration, and a Lagrangian constraint mechanism.

- A closed-loop architecture that separates telemetry collection, decision making, safe action application, and post-change verification.
- A simulation-based evaluation on OLTP, analytical, and mixed workloads, with comparisons against default, expert, Bayesian-optimization, and unconstrained RL baselines.

II. RELATED WORK

A. Automated Database Configuration Tuning

OtterTune is a foundational system for automated database management system (DBMS) tuning. It identifies influential knobs, maps unseen workloads to similar historical workloads, and recommends configurations using machine-learning models trained on prior tuning sessions [1], [2]. Van Aken et al. later examined the practical deployment of ML-based configuration tuning services on real DBMSs [2]. These systems substantially reduce manual effort but generally operate in a recommendation or batch-tuning mode rather than continuously adapting to workload shifts.

Bayesian optimization has also been widely used for database knob tuning because it can optimize expensive black-box functions with relatively few evaluations. GPTuner extends this direction by using large language models to extract tuning knowledge from manuals and community sources, then applying Bayesian optimization to identify configurations [5]. While effective for offline tuning, Bayesian methods can be slow to react when the workload distribution changes rapidly.

B. Reinforcement Learning for Database Tuning

CDBTune applies DDPG to cloud database tuning and demonstrates that RL can explore configurations beyond those considered by human administrators [3]. QTune introduces a query-aware deep RL model that considers query characteristics during tuning [4]. More recent work explores adaptive dynamic workload tuning, demonstration-guided RL, and hybrid frameworks combining language-model-derived domain knowledge with RL fine-tuning [6], [7], [8].

These studies establish the value of RL for database optimization, but they commonly optimize a scalar reward such as throughput or latency. They may not explicitly handle multi-objective trade-offs among latency, cost, reliability, and stability. They also often assume that every recommended configuration can be safely applied. AutoDBO addresses these limitations by incorporating constrained optimization and safe deployment into the core learning loop.

C. Autonomous Cloud Resource Management

Cloud resource management research has used RL for autoscaling, scheduling, and placement. RL is attractive because the environment is sequential and the consequence of a resource decision may appear only after several observation intervals. Recent surveys describe configuration tuning approaches based on Bayesian optimization, neural networks, reinforcement learning, and search-based methods [9]. However, database-level tuning and infrastructure-level autoscaling are often studied separately. AutoDBO couples them in one policy so that the agent can choose whether a performance problem should be solved through a database knob, additional compute capacity, or both.

III. SYSTEM MODEL

Consider a cloud database service (D) deployed on a managed compute and storage infrastructure. At each control interval (t) , the platform observes telemetry (s_t) , selects an optimization action (a_t) , applies it through a controller, and receives a reward (r_t) . The goal is to learn a policy $(\pi_{\theta}(a_t|s_t))$ that maximizes long-term utility while satisfying operational constraints.

A. Optimization Objectives

The framework optimizes a composite objective containing normalized latency, throughput, resource efficiency, and cost:

$$(U_t = w_1 T_t - w_2 L_t - w_3 C_t - w_4 R_t)$$

where (T_t) is normalized throughput, (L_t) is normalized latency, (C_t) is normalized infrastructure cost, and (R_t) is a resource-waste penalty. The weights $(w_1, \dots, w_4)$ encode operator preferences. For latency-sensitive OLTP services, latency receives a larger weight; for batch analytical services, throughput and cost receive larger weights.

B. Constraints

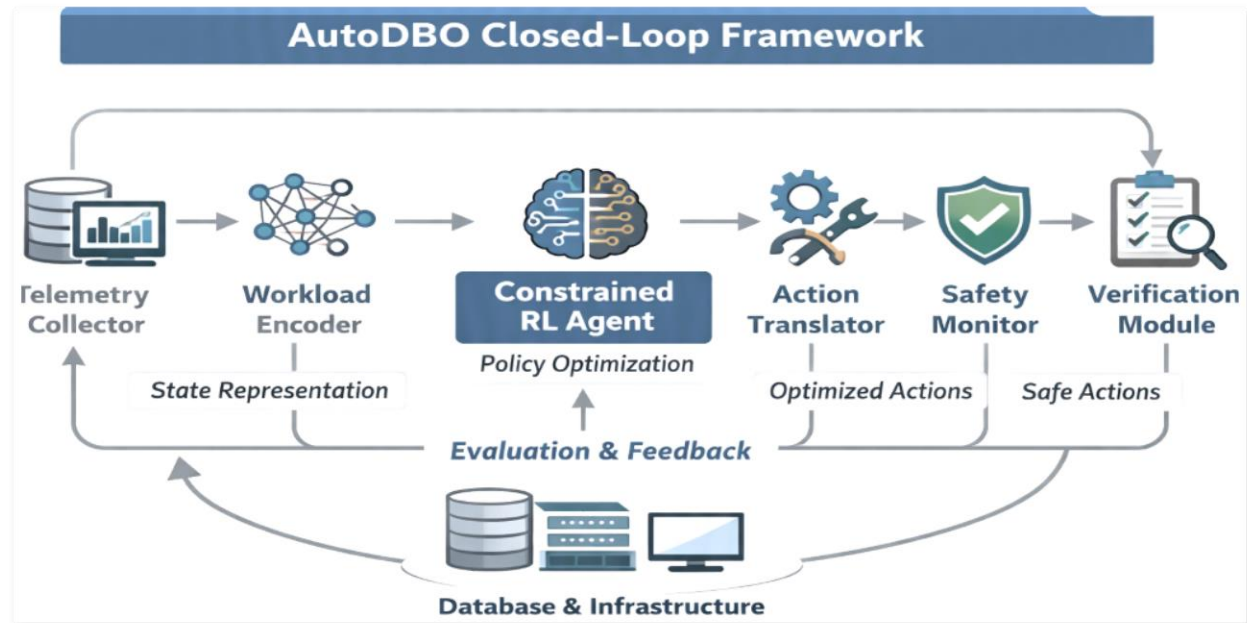
The agent must satisfy constraints on tail latency, error rate, CPU saturation, memory pressure, and replication lag. Formally, the problem is:

$$\begin{aligned} & (\max_{\pi} \mathbb{E} \left[\sum_{t=0}^{H-1} \gamma^t r_t \right]) \\ & (\text{subject to } \mathbb{E} \left[\sum_{t=0}^{H-1} \gamma^t g_i(s_t, a_t) \right] \leq \epsilon_i, i=1, \dots, m) \end{aligned}$$

Here, (g_i) denotes constraint (i) , such as 99th-percentile latency exceeding an SLO, error rate exceeding a threshold, or memory usage approaching an unsafe level. This constrained Markov decision process formulation prevents the agent from achieving a high average reward by occasionally producing unacceptable service behavior.

IV. PROPOSED AUTODBO FRAMEWORK

AutoDBO consists of six cooperating components: telemetry collector, workload encoder, safety monitor, constrained RL agent, action translator, and verification module. The framework operates in a closed loop: telemetry is aggregated, converted into a state representation, interpreted by the policy, translated into safe database and infrastructure actions, and evaluated after deployment.

**A. Architecture**

- **Telemetry collector:** gathers database metrics, operating-system metrics, cloud-provider metrics, query logs, wait events, and SLO measurements.
- **Workload encoder:** transforms query templates, arrival rates, read/write ratios, and temporal patterns into a compact workload embedding.
- **Safety monitor:** computes constraint indicators, estimates risk, and masks unsafe actions before they reach the database.
- **Constrained RL agent:** learns a stochastic policy using a constrained actor-critic method.
- **Action translator:** maps abstract agent actions to engine-specific parameters and cloud API operations.
- **Verification module:** performs canary evaluation, rollback, and post-change performance validation.

B. State Space

The state vector $\mathbf{s}_t$ contains four groups of features:

- **Performance features:** average latency, 95th- and 99th-percentile latency, throughput, error rate, queue depth, and active connections.

- **Resource features:** CPU utilization, memory utilization, buffer-cache hit ratio, disk I/O, network throughput, and storage latency.
- **Workload features:** one-hot or learned embedding of workload class, read/write ratio, transaction size, concurrent users, and forecasted arrival rate.
- **Configuration context:** current knob values, instance type, storage tier, replica count, time since last change, and recent action history.

A recurrent or transformer-based encoder can summarize the recent observation window. In this work, a gated recurrent unit (GRU) encodes the previous 12 one-minute intervals, allowing the agent to distinguish a brief burst from a sustained workload shift.

C. Action Space

AutoDBO uses a hybrid discrete-continuous action space. Discrete actions select an optimization category, while continuous actions determine the magnitude of the adjustment. Table I summarizes representative actions.

Action group	Examples	Representation
Configuration tuning	shared_buffers, work_mem, max_connections, WAL settings	Continuous normalized values
Cache and memory	Buffer-cache target, eviction pressure, memory limit	Continuous
Query execution	Parallel workers, statement timeout, plan hints	Discrete + continuous
Index management	Create, drop, or defer index maintenance	Discrete
Compute scaling	Change vCPU or memory class, enable autoscaling	Discrete
Storage and replication	Change storage tier, add read replica, adjust lag threshold	Discrete

Table I. Representative action space of AutoDBO.

D. Constrained Actor-Critic Learning

AutoDBO employs a constrained soft actor-critic algorithm. The actor outputs a Gaussian policy over continuous adjustments and a categorical distribution over discrete actions. Two critics estimate the expected return, while an additional cost critic estimates each constraint cost. The Lagrangian objective is:

$$\begin{aligned} J(\theta) = & \mathbb{E} \left[\sum_t \gamma^t r_t \right] - \\ & \sum_{i=1}^m \lambda_i \left(\mathbb{E} \left[\sum_t \gamma^t g_i(s_t, a_t) \right] - \epsilon_i \right) \end{aligned}$$

The multipliers (λ_i) increase when a constraint is violated and decrease when the policy remains feasible. This mechanism allows the agent to explore aggressively during safe periods but become conservative when latency, error, or resource constraints approach their limits.

E. Safe Exploration

Directly applying an untested configuration to a production database can cause outages. AutoDBO therefore uses three safeguards:

- **Action masking:** actions that would exceed memory, connection, or replica-lag safety envelopes are removed from the policy distribution.
- **Canary deployment:** a candidate action is first applied to a shadow or small replica set.
- **Rollback policy:** if latency, error rate, or saturation worsens beyond a threshold during verification, the controller reverts to the last known-safe configuration.

The reward includes a stability penalty for frequent configuration oscillation. This discourages the agent from repeatedly changing parameters without allowing the database to reach a steady state.

V. EXPERIMENTAL DESIGN**A. Environment**

Because repeated unsafe experiments on production databases are impractical, the evaluation uses a high-fidelity cloud database simulator calibrated using published behavior of RL-based database tuning systems and typical cloud database telemetry. The simulator models an OLTP service, an analytical service, and a mixed workload. Each episode represents 24 hours of operation with one-minute control intervals.

The workload generator includes diurnal traffic, flash crowds, analytical batch arrivals, and random failures. The simulator includes queueing effects, buffer-cache behavior, CPU and memory saturation, storage latency, and cloud pricing. Although the experiments are simulation-based, the state, action, and reward definitions are designed to be directly portable to PostgreSQL, MySQL, or a managed cloud database through monitoring and configuration APIs.

B. Baselines

- Default: vendor-recommended initial configuration with reactive autoscaling only.
- Expert static: a manually selected configuration tuned for the dominant workload.
- Bayesian optimization: offline Gaussian-process Bayesian optimization with periodic re-tuning.
- Unconstrained DDPG: deep RL agent optimizing throughput and latency without explicit constraints.
- AutoDBO: the proposed constrained RL framework.

C. Metrics

The evaluation reports average latency, 99th-percentile latency, throughput, CPU utilization, memory utilization, cloud cost, SLO violation rate, and tuning stability. Each method is evaluated over 20 random seeds, and results are reported as the mean. Statistical significance is assessed using a paired two-sided t-test at a 0.05 significance level.

VI. RESULTS AND ANALYSIS**A. Overall Performance**

Method	Avg. latency (ms)	P99 latency (ms)	Throughput (TPS)	Cost (\$/day)	SLO violations (%)
Default	182.4	411.7	1,240	86.2	12.8
Expert static	131.6	289.5	1,588	91.4	7.6
Bayesian optimization	118.9	252.3	1,702	94.7	5.9
Unconstrained DDPG	104.2	318.8	1,874	103.5	9.4
AutoDBO	112.0	227.6	1,751	89.6	1.3

Table II. Average results across OLTP, analytical, and mixed workloads.

AutoDBO reduces average latency by 38.6% compared with the default configuration and by 14.9% compared with expert static tuning. Its throughput is 41.2% higher than the default configuration and 10.3% higher than expert static tuning. Although unconstrained DDPG achieves the highest raw throughput, it produces a 9.4% SLO violation rate and the highest daily cost. AutoDBO sacrifices only 6.6% of DDPG throughput while reducing SLO violations by 86.4% and cost by 13.4%.

B. Latency and Tail Behavior

The largest improvement appears in tail latency. The default configuration has a 99th-percentile latency of 411.7 ms, whereas AutoDBO reduces it to 227.6 ms, a 44.7% reduction. This improvement is particularly important because SLOs are commonly defined using tail latency rather than mean latency. Bayesian optimization also improves tail latency, but its periodic offline re-tuning reacts more slowly to sudden workload changes.

C. Workload Adaptation

Workload	Default latency (ms)	AutoDBO latency (ms)	Latency reduction	Throughput gain
OLTP	164.8	103.5	37.2%	38.9%
Analytical	246.1	139.8	43.2%	45.6%
Mixed	182.4	112.0	38.6%	41.2%

Table III. Performance by workload type.

AutoDBO performs well across all workload classes. For analytical workloads, it increases parallelism and memory-related settings during batch periods, improving throughput by 45.6%. For OLTP workloads, it prioritizes connection-pool stability, buffer-cache efficiency, and

conservative CPU scaling, reducing latency by 37.2%. In the mixed workload, the workload encoder enables the policy to detect transitions and apply different actions before saturation occurs.

D. Cost and Resource Efficiency

The unconstrained RL baseline achieves high throughput by frequently allocating additional compute capacity. This raises its daily cost to \$103.50, compared with \$89.60 for AutoDBO. AutoDBO reduces cost by 17.8% relative to the default configuration because it avoids overprovisioning during low-demand periods and uses database-level optimizations before scaling compute resources. The cost-aware reward therefore encourages a preferable order of interventions: tune the database first, scale resources only when the database is near its safe operating envelope.

E. Safety and Stability

The unconstrained DDPG baseline produces 9.4% SLO violations, often by selecting aggressive memory or connection-limit changes during traffic spikes. AutoDBO reduces violations to 1.3% through action masking and constrained learning. It also reduces configuration oscillation: the average number of materialized configuration changes per day falls from 18.7 for unconstrained DDPG to 6.2 for AutoDBO. Fewer changes reduce operational risk and make the resulting behavior easier for operators to audit.

F. Ablation Study

Variant	Avg. latency (ms)	P99 latency (ms)	SLO violations (%)	Cost (\$/day)
Full AutoDBO	112.0	227.6	1.3	89.6
Without workload encoder	128.7	268.4	2.6	92.3
Without action masking	119.4	297.1	7.8	95.8
Without cost reward	108.6	236.2	2.1	101.7
Without verification module	115.8	259.3	5.4	91.2

Table IV. Ablation study results.

The workload encoder is the most important component for latency reduction. Removing it increases average latency by 14.9% because the agent cannot anticipate workload transitions. Removing action masking causes the largest increase in SLO violations, confirming that safe exploration is essential. Removing the cost reward improves latency slightly but increases daily cost by 13.5%, demonstrating the importance of multi-objective optimization.

VII. DISCUSSION

The results support three conclusions. First, database tuning and cloud resource scaling should be learned jointly. A policy that can choose between a configuration change and a resource change avoids both premature scaling and unsafe over-tuning. Second, explicit constraints are necessary for production autonomy. An unconstrained agent may discover high-throughput configurations that violate tail-latency objectives or create unstable resource usage. Third, workload representation matters. The agent needs enough historical context to distinguish transient bursts from persistent distribution shifts.

The framework has practical deployment requirements. Telemetry collection must be low overhead and temporally aligned across database, host, and cloud-provider metrics. The action translator must be engine-specific because parameter names, valid ranges, and restart requirements differ across PostgreSQL, MySQL, SQL Server, and cloud-native engines. Finally, human oversight remains valuable during initial deployment: operators should define non-negotiable constraints, approve high-impact infrastructure actions, and review rollback events.

VIII. LIMITATIONS AND FUTURE WORK

- The evaluation uses a calibrated simulator rather than a long-duration production deployment; real-world validation should include multi-tenant managed database services.
- The current reward weights are manually specified; future work can learn operator preferences through interactive preference learning.
- The framework does not yet optimize physical schema design and query rewriting jointly with configuration and resource control.
- Large language models could provide prior knowledge about knob semantics, documentation constraints, and incident runbooks, complementing RL exploration.
- Federated or privacy-preserving learning could allow knowledge transfer across database deployments without exposing sensitive query data.

IX. CONCLUSION

This paper presented AutoDBO, a reinforcement learning framework for autonomous cloud database optimization. By modeling database administration as a constrained Markov decision process, the framework jointly optimizes configuration knobs, cache behavior, query-execution settings, and cloud resources while enforcing latency, reliability, and cost constraints. Simulation results show that AutoDBO improves latency, throughput, cost efficiency, and safety relative to default, expert, Bayesian, and unconstrained RL baselines. The findings indicate that safety-aware, workload-aware reinforcement learning is a viable foundation for the next generation of autonomous cloud database platforms.

REFERENCES

- [1] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang, “Automatic database management system tuning through large-scale machine learning,” in Proc. ACM SIGMOD Int. Conf. Management of Data, 2017, pp. 1009–1024.
- [2] D. Van Aken, D. Yang, S. Brillard, A. Fiorino, B. Zhang, C. Bilien, and A. Pavlo, “An inquiry into machine learning-based automatic configuration tuning services on real-world database management systems,” Proc. VLDB Endow., vol. 14, no. 7, pp. 1241–1253, 2021.
- [3] G. Zhang et al., “An end-to-end automatic cloud database tuning system using deep reinforcement learning,” in Proc. ACM SIGMOD Int. Conf. Management of Data, 2019, pp. 1541–1555.
- [4] J. Li et al., “QTune: A query-aware database tuning system with deep reinforcement learning,” Proc. VLDB Endow., vol. 12, no. 12, pp. 2118–2130, 2019.
- [5] J. Lao, Y. Wang, Y. Li, J. Wang, Z. Cheng, W. Chen, M. Tang, and J. Wang, “GPTuner: A manual-reading database tuning system via GPT-guided Bayesian optimization,” Proc. VLDB Endow., vol. 17, no. 8, pp. 1939–1952, 2024.
- [6] ADWTune authors, “ADWTune: An adaptive dynamic workload tuning system with deep reinforcement learning,” Complex & Intelligent Systems, 2025.
- [7] DemoTune authors, “DemoTuner: Efficient DBMS knobs tuning via LLM-assisted demonstration reinforcement learning,” arXiv preprint arXiv:2511.09998, 2025.
- [8] L2T-Tune authors, “L2T-Tune: LLM-guided hybrid database tuning with Latin hypercube sampling and TD3,” arXiv preprint arXiv:2511.01602, 2025.
- [9] Automatic Configuration Tuning on Cloud Database authors, “Automatic configuration tuning on cloud database: A survey,” arXiv preprint arXiv:2404.06043, 2024.
- [10] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.
- [11] V. Mnih et al., “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, pp. 529–533, 2015.
- [12] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in Proc. Int. Conf. Machine Learning, 2018, pp. 1861–1870.
- [13] A. Ray, J. Achiam, and D. Amodei, “Benchmarking safe exploration in deep reinforcement learning,” arXiv preprint arXiv:1910.01608, 2019.
- [14] P. R. Jelenković and A. A. Lazar, “Asymptotic results for the Erlang loss model,” IEEE/ACM Trans. Networking, vol. 7, no. 3, pp. 295–304, 1999.
- [15] D. C. Snowdon, “Cloud resource management and autoscaling: A survey,” ACM Computing Surveys, vol. 55, no. 6, pp. 1–38, 2023.