electronics

MDPI

Article

FASTune: Towards Fast and Stable Database Tuning System with Reinforcement Learning

Lei Shi [1,2,3,*], Tian Li [2], Lin Wei [1], Yongcai Tao [2], Cuixia Li [1,*] and Yufei Gao [1,3]

[1] School of Cyber Science and Engineering, Zhengzhou University, Zhengzhou 450002, China; yfgao@zzu.edu.cn (Y.G.)
[2] School of Computer and Artificial Intelligence, Zhengzhou University, Zhengzhou 450001, China; 202022172013213@gs.zzu.edu.cn (T.L.)
[3] Songshan Lab, Zhengzhou 450046, China
* Correspondence: shilei@zzu.edu.cn (L.S.); lcxxcl@zzu.edu.cn (C.L.)

Abstract: Configuration tuning is vital to achieving high performance for a database management system (DBMS). Recently, automatic tuning methods using Reinforcement Learning (RL) have been explored to find better configurations compared with database administrators (DBAs) and heuristics. However, existing RL-based methods still have several limitations: (1) Excessive overhead due to reliance on cloned databases; (2) trial-and-error strategy may produce dangerous configurations that lead to database failure; (3) lack the ability to handle dynamic workload. To address the above challenges, a fast and stable RL-based database tuning system, FASTune, is proposed. A virtual environment is proposed to evaluate configurations which is an equivalent yet more efficient scheme than the cloned database. To ensure stability during tuning, FASTune adopts an environment proxy to avoid dangerous configurations. In addition, a Multi-State Soft Actor–Critic (MS-SAC) model is proposed to handle dynamic workloads, which utilizes the soft actor–critic network to tune the database according to workload and database states. The experimental results indicate that, compared with the state-of-the-art methods, FASTune can achieve improvements in performance while maintaining stability in the tuning.

Keywords: database tuning; reinforcement learning; decision making; deep learning

check for updates

Citation: Shi, L.; Li, T.; Wei, L.; Tao, Y.; Li, C.; Gao, Y. FASTune: Towards Fast and Stable Database Tuning System with Reinforcement Learning. *Electronics* **2023**, *12*, 2168. https://doi.org/10.3390/electronics12102168

Academic Editors: Franco Cicirelli and Manohar Das

Received: 10 March 2023
Revised: 2 May 2023
Accepted: 7 May 2023
Published: 10 May 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

A database has numerous tunable parameters [1], which can significantly affect performance metrics, such as latency and throughput. Appropriate configurations can improve the database performance. Database tuning is an NP-hard problem [2,3], making the search for optimal configurations a challenging task for DBAs. In recent years, some studies have focused on automatic database tuning, including rule-based methods [4–6] and learning-based methods [3,7–12]. Rule-based methods search for optimal configurations based on fixed rules, which have previously been observed to improve database throughput compared to default configurations on OLTP (Online Transaction Processing) workloads. However, the rule-based method fails to utilize experience from previous tuning processes and thus needs to restart the search process for each new tuning request. Learning-based methods, e.g., Ottertune [8], utilize a machine-learning model to select knobs, map the workload, and recommend configurations to improve database performance. However, these methods have two limitations.

Firstly, their reliance on the pipelined approach can result in sub-optimal performance, as the optimal solution for a particular stage may not guarantee the optimal solution for the subsequent stage, and different stages may not complement each other effectively. Consequently, an end-to-end optimization of overall performance becomes unfeasible.

Secondly, the models depend on large-scale, high-quality training samples, which can be difficult to access. For instance, the performance of cloud databases is influenced by

various factors, such as memory size, disk capacity, and workloads. Capturing all these conditions and accumulating high-quality samples present challenging tasks.

In this case, conventional machine learning approaches have poor adaptability and the model requires retraining to adapt to the new environment.

Another family of learning-based methods, e.g., Qtune [2], CDBTune [7] and HUNTER [3] address the tuning problem using reinforcement learning [13,14]. They consider the database an environment and use an agent to find optimal configurations through a trial-and-error strategy, which alleviates the burden of collecting a large number of samples in the initial modeling stage. However, applying these methods in the real world still has several challenges:

First, the agent updates the policy according to an evaluation of configurations, which depends on a time-consuming stress test on the database.

Second, a trial-and-error strategy is adopted in RL to exploit optimal configurations. Thus, the agent may recommend dangerous configurations that can cause performance degradation or database crashes which is unacceptable.

Third, the workload is assumed constant, so the tuning aims to improve performance on a specific workload. However, as shown in Figure 1, the real-world workload can be varied so that the tuning result may be delayed.

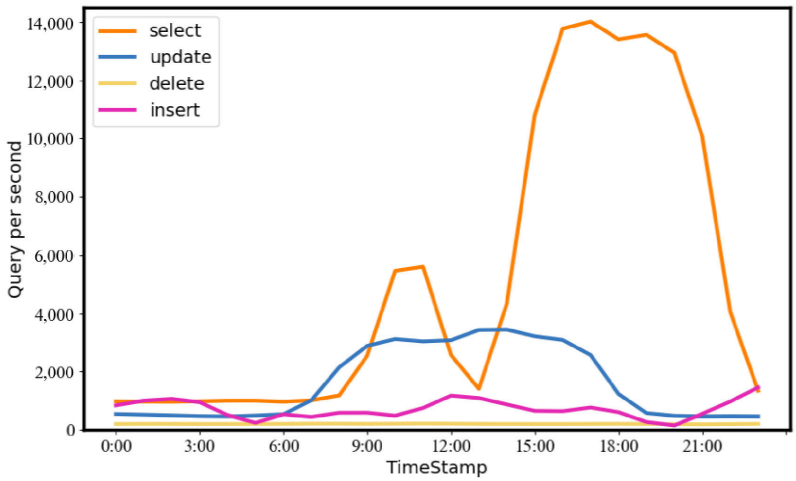

Figure 1. Statistics of different query types over time in the workload of the BusTracker application (a mobile phone application for live-tracking of the public transit bus system).

To address the above problems, FASTune is proposed to prevent dangerous configurations, accelerate the tuning process and adapt to the dynamic workload. To boost efficiency and ensure the stability of the database during the tuning process, FASTune implements an environment proxy. Environment proxy achieves this goal through: (i) Discarding actions that could cause a dramatic drop in database performance or database failure. (ii) Reducing inefficient evaluation of the action. In contrast to the existing methods that enable direct interaction between the agent and the environment, our research incorporates a wrapped environment with a proxy, through which the agent interacts with the environment. The evaluation of action is also handled by environment proxy. Environment proxy has three key components: Filter, Virtual Environment, and Dispatcher. Filter excludes dangerous actions to avoid fluctuations and database failures. The Filter uses rule-based and learning-based methods to evaluate dangerous actions. The Filter extracts rules from the documents (e.g., the database manual), and actions that match these rules will be considered dangerous and excluded. Using rules is straightforward and effective but can not cover all situations because there are non-linear correlations between knobs and database performance. So Filter utilizes a classification model to detect a dangerous action, reduces the dangerous action significantly, and contributes to the stability of the database.

FASTune employs a virtual environment to evaluate configurations more effectively. A virtual environment is a model that mimics the behavior of the environment as closely as possible while being computationally feasible. The virtual environment estimates the evaluation of the action, which reduces unnecessary stress tests on the database. Virtual Environment can reduce tuning time more efficiently and does not require additional memory and storage compared to cloned database. However, since the estimation is based on historical data, predicting the performance of actions can be difficult when there are insufficient relevant data. To address this issue, the dispatcher is proposed, dispatcher divides the actions into two groups: "common" and "uncommon". Common actions mean that the action is numerically close to the previous action so that the performance of the action can be predicted based on historical data. Conversely, for an uncommon action, deploying it on the database and performing a stress test is necessary to acquire its performance. Therefore, the environment (i.e., the database instance) is required. The details of the Virtual Environment will be discussed in Section 5.

Since several studies have shown that different workloads are sensitive to different knobs [10,15], it is necessary for agent to consider workload characteristics during the tuning. A Multi-State Soft Actor–Critic model (MS-SAC) is proposed to handle dynamic workloads. Different from previous work, MS-SAC finds optimal configurations according to both environment state and workload state. FASTune continuously collects workload arriving at the database and constructs a model to predict future workload. The predicted results will be provided to the agent as workload state. The paper makes the following contributions:

1. Environment Proxy is proposed in Reinforcement Learning for database tuning, which improves the efficiency and stability of tuning;
2. FASTune utilizes a combined approach to exclude dangerous configurations;
3. MS-SAC model is proposed, which utilizes the soft actor–critic network to find optimal configurations based on both the environment and workload state;
4. Experimental evidence demonstrates that FASTune can considerably reduce tuning time and ensure the stability of database tuning.

2. Related Works

2.1. Database Tuning

Automatic database tuning systems have been studied for decades [2,8,16–36]. They can be broadly classified into the following categories.

2.1.1. Rule-Based Methods

Rule-based methods use rules or heuristics to find optimal configurations for the database. iTuned [12] utilizes statistical methods to select essential knobs and find correlations between these knobs and performance. Wei et al. [6] propose a method that generates rules and uses them for tuning. BestConfig [4] splits the high-dimension search space into sub-spaces and recommends configurations using a recursive bound-and-search algorithm. DB2 proposes a self-tuning memory model that uses heuristics to allocate proper memory to the components of database [37], Tran et al. [38] used linear and quadratic regression models for buffer tuning. The DBSherlock helps DBA to diagnose faults by comparing slow regions and normal regions [39]. However, rule-based methods rely on historical data or rules and fail to utilize previous tuning processes' experiences.

2.1.2. Learning-Based Methods

Ottertune [10] and ResTune [15] map the tuning problem to black-box optimization issues and uses the traditional Machine learning (ML) method to obtain optimal solutions. However, they depend on a large amount of high-quality training data. Another family of learning-based methods, e.g., CDBTune [7] and Hunter [3], utilize RL to tune database and the configurations. They train an agent that uses a trial-and-error strategy to search for optimal configurations. In the beginning, agent randomly recommends configurations and

applies them to the database, and then agent adjusts its policy based on the feedback from the database. RL can achieve high performance and does not require training data, but agent may recommend risky configurations that cause database failure. In addition, the interaction with the database is time-consuming, which prevents them from being applied in the production environment. Kanellis [40] proposed a knobs importance ranking method to automatically filter the most important knobs to reduce tuning time, which is a limited improvement. CDBTune uses cloned database instances to accelerate the tuning process. However, it brings a huge cost.

2.2. Reinforcement Learning

Reinforcement learning is proposed to solve multi-person decision problems [13,41]. RL is a model that uses an agent (learner) to execute an action (make a decision) in a specific environment (scenario) and learn from the interactions with the environment [42]. Different from supervised learning, RL does not require much training data. Instead, the agent updates its policy of outputting actions to maximize a reward function. Usually, a higher reward means a better decision(action).The essential part of reinforcement learning is learning from interactions with the environment to maximize a cumulative reward signal over a period of time. In other words, an agent learns to take actions based on the feedback received from the environment, with the goal of receiving the highest possible reward in the long run. The agent's goal is not to predict a specific output but to maximize the expected cumulative reward. This trial-and-error learning process is a key aspect of reinforcement learning. Reinforcement learning has been successfully utilized in different optimal problems [43–47], and there is a large number of published studies that use RL to address database problems. Neo [48] developed a learnable query optimizer with RL. SageDB [49] suggests a vision that learning-based models can replace some components in a DBMS. Basu et al. use RL to implement index tuning. OpenGauss [50] uses a deep reinforcement learning model to optimize the query plan.

3. System Overview

In this section, The architecture and the workflow of our database tuning system are demonstrated.

3.1. Architecture

FASTune is an database tuning system with reinforcement learning, and an overview of it is shown in Figure 2.

FASTune consists of three parts. First, Agent interacts with the Environment Proxy and recommends configurations. Agent is a Soft Actor–Critic (SAC) [51] network containing two independent neural networks: actor and critic. Actor takes multi-state as input and outputs an action (i.e., configurations), and the Proxy computes a reward based on the evaluation of action. Critic takes multi-state and action as input and outputs a Q-value (score). Critic updates according to the reward, and actor updates according to the Q-value. Details of SAC will be discussed in Section 4. Second, the Environment Proxy contains three components—(a) Filter using rules and machine learning methods to exclude dangerous action for safety tuning. (b) Virtual Environment alleviates the burden of evaluating the action in tuning. (c) Dispatcher receives the action and deploys it to environment or sends it to Virtual Environment. Third, Workload Collector records recently arrived workloads and predict the expected queries in the future. The interactions between Agent and Environment Proxy generate trajectories stored in memory; trajectories are a set of tuples $< S_w, S_d, A, r >$. S_w is the feature of predicted future workload, and S_d represents database state such as *lock_row_lock_current_waits*, etc. A is an action generated by Agent, and r is the reward of action calculated from throughput and latency. Note that, different from existing methods, S_w and S_d are combined and provided to the agent as multi-state.

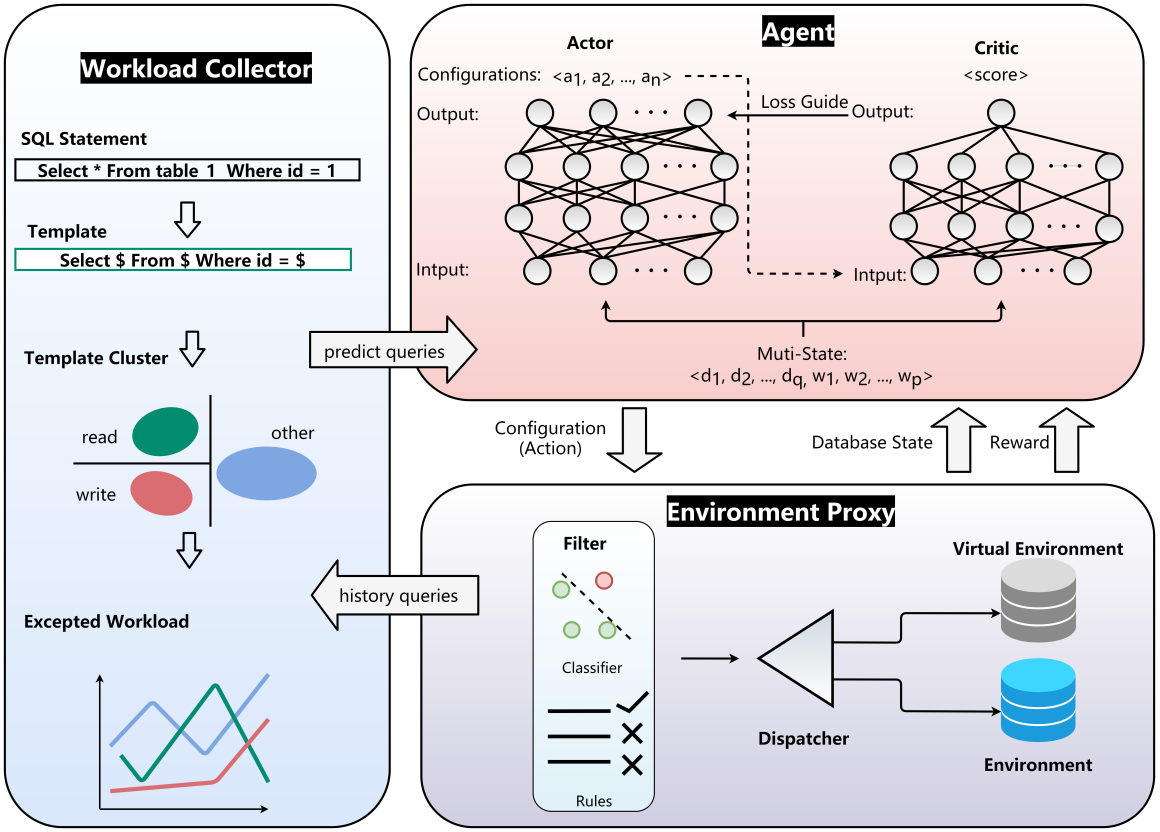

Figure 2. Overview architecture of FASTune.

3.2. Workflow

FASTune works in 4 stages:

1. Initialize. FASTune stress tests the database with default configurations to build an initial performance baseline. Then the weights of the actor and critic network are initialized by the stochastic distribution. Workload Collector starts to collect workload until the tuning process ends;
2. Recommend. As the tuning request arrives. Agent takes environment state and workload state as input and then outputs an action based on the policy. The Agent sends the action to Environment Proxy and waits for a reward (i.e., the evaluation of action);
3. Evaluate. The filter in the environment proxy excludes potentially harmful actions before the evaluation process. Once action passes through the filter, Dispatcher determines whether the action is "common" or "uncommon" based on historical data, "common" and "uncommon" action will be sent to Virtual Environment and Environment, respectively. If action is sent to Virtual Environment, Agent will get the predicted evaluation. Otherwise, the Proxy deploys the action on database and returns an evaluation based on a stress test. Every interaction between the proxy and the agent generates a sample, which is then stored in memory;
4. Update. Environment Proxy calculates the reward according to the action evaluation and returns it to Agent. Since a higher reward indicates better performance, Agent updates the policy network (i.e., actor) to earn a higher reward and updates value network (i.e., critic) to provide more accurate feedback for the actor. The detail of actor and critic are described in Section 4.

Stage 2–4 is repeated until the model converges or meets other stop conditions (e.g., database throughput reaches a given threshold). Finally, the Agent deploys configurations with ideal performance on the database instance.

4. RL for Tuning

This section introduces RL to solve the tuning problem, and then the proposed MS-SAC model is presented.

Usually, there are hundreds of tunable knobs in databases, some of which are in continuous space [1]. So it is hard for traditional machine learning methods to find optimal configurations [2]. As both the rule-based and conventional learning-based approaches have limitations, designing a more efficient tuning system is necessary. Reinforcement Learning (RL) is a learning method that can effectively operate with limited samples during initial training.Because RL makes decisions through the interaction process between an agent and its environment, relying on accumulated rewards rather than labels to perform learning. Some popular RL methods include Q-learning [52], DQN [53], and DDPG [54]. Q-learning uses Q-tables defined as $Q(s, a)$, where the rows represent the Q-value of states and the columns represent actions. The Q-value measures how beneficial it would be to take a particular action in the current state. $Q(s, a)$ is iteratively defined as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r + \gamma max_{a_t+1} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (1)$$

Q-Learning updates the Q-table based on the Bellman Equation, where a represents the learning rate, γ is a discount factor, and r is the performance at time $t + 1$. However, it is impractical to solve database tuning problems with a large state space because Q-table can hardly store so many states. Even with ten metrics discretized into ten equal bins each, it results in 10^{10} states. DQN uses neural networks to replace Q-tables but can only output discrete actions, while knob combinations in a database are high-dimensional and continuous. CDBtune employs DDPG, which utilizes two deep neural networks: an actor and critic networks to address the issue. The actor network maps states to actions, and the critic network approximates the state-action value function. The actor network learns the policy, while the critic network estimates the value function of the policy. With the actor–critic architecture, DDPG immediately acquires the value of the current action based on the current state without having to compute and store Q-values for all actions like DQN. As a result, DDPG can learn the policy with high-dimensional states and actions, making it a more suitable choice for solving database tuning problems.

However, the interaction between the deterministic policy network (i.e., actor) and the value network (i.e., critic) makes DDPG brittle to hyper-parameter: discount factor, learning rates, and other parameters must be set carefully to achieve ideal results. Consequently, using DDPG on complex high-dimensional tasks is hard to stabilize. These issues limit the application of RL to real-world tasks. The Multi-State Soft Actor–Critic model (MS-SAC) is proposed to overcome these disadvantages.

4.1. MS-SAC Model

MS-SAC uses the soft actor–critic networks [51] to develop an agent specifically for database tuning. The SAC algorithm is a variant of the actor–critic algorithm that optimizes the actor's policy objective by introducing entropy regularization. This regularization helps to encourage exploration and prevent the policy from becoming too deterministic, which can lead to sub-optimal solutions. The concept of Multi-State refers to the agent recommending configurations based on both the environment and the workload state, unlike previous works. This approach enhances FASTune's adaptability and stability, and we plan to introduce more states (such as network states) in the future. It is worth noting that, unlike traditional RL, the environment is enveloped by our proposed proxy, which means that the agent interacts with the proxy rather than the environment itself. More information about the environment proxy can be found in Section 5. Table 1 illustrates the mapping from MS-SAC to the tuning task and clarifies the notion presented.

Table 1. Mapping from Reinforcement Learning to Database Tuning.

Variables	MS-SAC	Database to Be Tuned
E	Environment	Database to be tuned
S	Database and workload state	Metrics of database and workload
a	Action	Configurations of database
r	Reward	Performance improvement
-	Agent	The soft actor–critic network
π	Policy	-
Q^{π}	Critic	-
V^{π}	Actor	-
θ	The weights in actor	-
ω	The weights in critic	-
γ	Discount factor	set to 0.9
α	Coefficient of explore and exploit	set to 0.2
ρ	Coefficient of soft update target network	set to 0.01
L	Loss function	-
-	Environment proxy	-

1. Environment. The Environment is the tuning target. Precisely, a database instance;
2. Database state. The Database State records the metrics of database, which consist of cumulative value (e.g., *lock_deadlocks*) and state value (e.g., $file\_num\_open\_files$); both reflect the situation inside the database;
3. Workload state. Workload State represents the characteristics of the upcoming workload. FASTune combines the Workload State with and Database State as the Multi-State, which is provided to the Agent when generating an Action;
4. Action. Action is database configurations generated by Agent. From a mathematical viewpoint, Action and Multi-State are both vectors. Agent maps Multi-State to Action, given the State, Agent deterministically outputs an Action;
5. Reward. Reward is a scalar that reflects the quality of Action. FASTune calculates the reward according to performance change after the database deploys a new action. A higher reward means greater Action. Note that the MS-SAC model optimizes policies to maximize the expected entropy of the policy, so both performance change and entropy of the action are considered in the calculation of the reward;
6. Agent. FASTune utilizes the actor–critic networks as an Agent to tune knobs. Agent receives a Reward and Multi-State from Environment Proxy and updates the policy to learn how to recommend high-quality Action that can earn a higher reward.

4.2. Training

As described above, the agent is to maximize a cumulative reward signal over time which can be defined as the function:

$$\pi^* = \max_{\theta} \mathbb{E}[\sum_{t=0}^{\infty} R(s_t, a_t) + \alpha H(\pi(\cdot|s_t))] \tag{2}$$

where the policy of the agent is represented by π. $R(s_t, a_t)$ is the reward function, and α is the coefficient. $H(\pi(|s_t))$ represents the entropy of the actions. The entropy of a random variable x with probability density $p(x)$ is defined as:

$$H(P) = \mathop{\mathbb{E}}_{x \sim P}[-logP(x)] \tag{3}$$

Entropy reflects the degree of disorder in a system. In database tuning optimization, this term represents the diversity of the agent's output configurations. Entropy maximization leads to more exploration and thus prevents the model from converging to a bad local optimum.

The policy function with entropy item is defined as:

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\pi}\left[\sum_{t=0}^{\infty}\gamma^{t}\Big(R(s_t,a_t,s_{t+1}) + \alpha H(\pi(\cdot|s_t))\Big)\right] \tag{4}$$

The action-value function (Q-function) with entropy item is defined as:

$$Q^{\pi}(s,a) = \mathop{\mathbb{E}}_{\pi}\left[\sum_{t=0}^{\infty}\gamma^{t}R(s_t,a_t,s_{t+1}) + \alpha\sum_{t=1}^{\infty}\gamma^{t}H(\pi(\cdot|s_t))\right] \tag{5}$$

With the equations above, the V^{π} and Q^{π} can be connected by

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{a\sim\pi}[Q^{\pi}(s,a)] + \alpha H(\pi(\cdot|s)) \tag{6}$$

and the Bellman equation for Q^{π} is:

$$\begin{aligned} Q^{\pi}(s,a) &= \mathop{E}_{s'\sim P, a'\sim\pi} R(s,a,s') + \gamma\big(Q^{\pi}(s',a') + \alpha H\big(\pi(\cdot|s')\big)\big) &(7)\\ &= \mathop{E}_{s'\sim P} R(s,a,s') + \gamma V^{\pi}(s'). &(8)\end{aligned}$$

To alleviate the overestimation problem, SAC concurrently has two q-functions with parameters ω_1 and ω_2 and one policy function with parameter θ. SAC selects the one with a lower value between two Q-functions, and the loss functions for the Q-networks(critic) is:

$$L_Q(\omega) = \mathop{\mathrm{E}}_{(s,a,r,s',d)\sim\mathcal{D}}\Big[\big(Q_\omega(s,a) - y(r,s',d)\big)^2\Big] \tag{9}$$

the loss of the policy network(actor) is :

$$L_\pi(\theta) = \mathop{\mathbb{E}}_{s\sim R, a\sim\pi_\theta}[\alpha log(\pi_\theta(a|s)) - Q_\omega(s,a)] \tag{10}$$

The training process are summarized in pseudo-code in Algorithm 1.

The network structure and parameter of agent have been described in Table 2 to make it easier to understand and implementation.

During the training process, the MS-SAC model learns a stochastic policy that maximizes the expected reward while also maximizing entropy, leading to a more diverse set of actions and better exploration. The model also learns a Q-function that estimates the state-action value and can be used to guide the policy towards more optimal actions. The use of a replay buffer and target networks helps stabilize the learning process and prevent overfitting to recent experiences. With proper design and training, MS-SAC performs well with high-dimension data. Since there are many tunable knobs in a database with continuous space, MS-SAC is suitable for database tuning problems.

Table 2. Network and parameters of agent.

Actor Layer	Actor Param	Critic Layer	Critic Param
Input	33	Input	(Number of knobs) + 33
Full connection	128	Full connection	128
Activation function	ReLU	Activation function	ReLU
Full connection	128	Full connection	128
Activation function	ReLU	Activation function	ReLU
Full connection	64	Full connection	64
Output	Number of knobs	Data	1

Algorithm 1 Train Agent

Input: initial actor parameters θ, critic parameters ω_1, ω_2, empty replay buffer $\mathcal{D}$.
let target parameters equal to main parameters $\omega_{\text{target},1} \leftarrow \phi_1$, $\omega_{\text{target},2} \leftarrow \omega_2$.
while *!converged* **do**
observe the initial state of the environment s_1.
execute a_t and get reward r_t
the state of environment change to s_{t+1}
store (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
for time = 1 to ∞ **do**
select an action based on policy: $a_t = \pi(\theta)_{s_t}$.
if *ready to update* **then**
for k = 1 to K **do**
get N samples from $\mathcal{D}$:$\{(s_i, a_i, r_i, s_{i+1})\}_{i=1,2,\ldots,N}$
for each sample, compute target for the Q-functions:

$$y_i(r, s', d) = r_i + \gamma\left(\min_{i=1,2} Q_{\omega_{\text{targ},i}}(s_{i+1}, a_{i+1}) - \alpha \log \pi_\theta(a_{i+1}|s_{i+1})\right), \quad a_{i+1} \sim \pi_\theta(\cdot|s_{i+1})$$

update critic to minimizing the loss function:

$$\nabla\omega_i \frac{1}{N}\sum_{i=1}^{N}(y_i - Q_\omega(s_i, a_i))^2, \ for\ i = 1, 2$$

update actor by gradient ascent using:

$$\nabla\theta = \frac{1}{N}\sum_{i=1}^{N}\left(\alpha log \pi_\theta(\tilde{a_i}|s_i) - \min_{j=1,2} Q_{\omega_{targ,j}(s_i,\tilde{a_i})}\right)$$

update target network:

$$\omega_{\text{targ},i} \leftarrow \rho\omega_{\text{targ},i} + (1-\rho)\omega_i, \ for\ i = 1, 2$$

end for
end if
end for
end while

5. Environment Proxy

Earlier studies on reinforcement learning have perceived databases as an environment [2,3,10,15] and agent update according to the feedback from environment. However, in database tuning problems, the agent requires a non-trivial amount of time for action evaluation. Moreover, RL employs trial-and-error to devise a solution to the tuning problem, which means actions from the agent may lead to performance degradation or database crash. To address these problems, Environment Proxy has been suggested to act as an interface between the agent and the environment, adding stability and safety. The environment proxy contains the Filter, Virtual Environment, and Dispatcher. The environment (i.e., the database that needs to be tuned) is wrapped with an environment proxy. Instead of deploying configurations to the environment directly, a combined approach is used to check configurations. Filter drops dangerous configurations to bring stability to the database. Proxy provides a more efficient evaluation of configurations using a Dispatcher and Virtual Environment. Dispatcher categorizes the configurations as "uncommon action" if the evaluation of the action is hard to estimate. On the contrary, the "common action" means the Virtual Environment can estimate an evaluation according to historical data. The proxy replaces the original position of environment in RL, and the interior of proxy is a black box for agent.

5.1. Filter

The Filter adopts a rule-based and learning-based method to exclude dangerous configurations. If the configurations are considered dangerous, Filter notices environment proxy to discard it and return a negative reward to agent as punishment. To judge the action, FASTune first extracts fixed rules from documents (e.g., the manual) and adds them to the rules library. The action matches any rule in the library is considered dangerous. For example, in MySQL, if $com\_stmt\_prepare - com\_stmt\_close$ is greater than $max\_prepared\_stmt\_count$, it may cause database failure because database cannot create more than $max\_prepared\_stmt\_count$ statements. Extracting rules from documents can be burdensome. Fortunately, DB-BERT [11] brings some light to this problem.

DB-BERT utilizes the BERT for document analysis, BERT (Bidirectional Encoder Representations from Transformers) is a transformer-based deep learning model. It is one of the most popular and powerful models for natural language processing tasks, including text classification, question answering, and text generation. BERT is pre-trained on a large amount of text data, using two unsupervised tasks: masked language modeling (MLM) and next sentence prediction (NSP). This pre-training approach allows BERT to learn various language tasks without requiring task-specific training data. That makes BERT a highly flexible and versatile NLP model that can be fine-tuned for a wide range of NLP tasks with state-of-the-art performance. DB-BERT is an extension of the BERT model that specializes in extracting explicit and implicit parameter references from the text. It does so by comparing the BERT encoding of the text with those of the DBMS parameter names, selecting the parameter with the smallest cosine distance. That allows DB-BERT to pair extracted values with parameters that are explicitly mentioned or are similar to the text. Additionally, DB-BERT can translate tuning hints into arithmetic formulas using a sequence of decisions and reinforcement learning.FASTune uses DB-BERT to exploit rules from a document and feed these rules to filter.

Using rules to exclude dangerous action is effective, but rules can not explore potential relations between knobs and the database performance. Thus, FASTune utilizes a classifier based on Support Vector Machines (SVM) [55,56]. SVM is one of the most robust binary classifier models with a wide range of applications in several fields. SVMs aim to find the best possible decision boundary that separates two or more classes of data(e.g., dangerous and safe). SVM maximize the margin, or the distance between the hyperplane and the closest data points from both classes, and use the closest data points, called support vectors, to determine the hyperplane parameters. Once the hyperplane is determined, the SVM can classify new data points based on which side of the hyperplane they fall. A newly collected data point is classified as belonging to one or the other class, based on which side of the hyperplane it falls into. SVMs can handle complex data sets with high dimensions and nonlinear boundaries using kernel functions.

FASTune adopts SVM to divide the actions into dangerous and safe groups. FASTune maps each action to a vector, representing a point in the high-dimension coordinate system. Then FASTune uses SVM to find a hyperplane to split these points into two groups. Since these points are not linearly separable, we use Radial Basis Function (RBF) kernel [57,58] to map them to higher dimensional spaces.The RBF kernel function is a powerful tool for SVMs because it can handle complex, nonlinear relationships between input variables. By transforming the input data into a higher-dimensional feature space using the RBF kernel, SVMs can find decision boundaries that are highly flexible and can accurately classify data points that are not linearly separable in the original feature space.

There are two important parameters in RBF: γ and c. Parameter c makes a trade-off between the misclassification of examples and the simplicity of hyperplane. A higher value of c increases the correctness of classification, while a lower c makes the hyperplane smooth. The γ determines how influential an example can be. The greater the value of γ is, the more it affects the neighboring data points. Choosing the right γ and c is vital for the performance of SVM. To address this problem, FASTune adopts the exhaustive grid search method that tries all combinations of c and γ from a grid of them. Then, each c and

γ combination is evaluated by k-fold cross-validation. Finally, we get the best combination of c and γ.

Note that the classifier we introduced above needs labeled training data. To collect training data, configurations are generated using uniform random distribution, and each set of configurations is deployed to the database. The configurations are labeled as dangerous if database performance drops sharply or fails. Otherwise, FASTune marks it as safe. Collecting training data can be a hard task. In future work, We would like to explore more efficient methods to generate training data and open the source code and dataset.

5.2. Dispatcher

Intuitively, if the action is similar to the previous (i.e., common action), it is possible to predict the performance changes in a database; thus, it is not necessary to deploy the action to the database and run a time-consuming stress test. On the contrary, If an action is rarely seen or never seen before (i.e., uncommon action), it will be hard to predict its impact on database performance, so the Dispatcher will send the action to the real database (i.e., environment) to see what happens. The main challenge for Dispatcher is to define what is uncommon action. Fortunately, ODT (outlier detection technique) can address the challenge effectively. Outlier detection has been studied for decades and is widely used in various fields. Outlier detection is used to identify data points or observations that are significantly different from the other data points in a dataset. There are several methods for outlier detection, including statistical methods, distance-based methods, and machine learning methods. In the context of FASTune, uncommon actions are considered outliers, and a distance-based method is used to identify them. Outlier detection relies on the idea that uncommon actions are likely to be located far away from the other data points in a dataset. The Mahalanobis distance is used to measure the distance between each data point. Data points significantly far away from most data points are identified as outliers. To determine where most data points are located, gaussian distribution is introduced, where the majority of data points are located close to the mean, and the frequency of data points decreases as they move away from the mean.Therefore, data points significantly far away from the mean are likelier to be outliers.

FASTune performs outlier detection by assuming that the regular data come from a gaussian distribution. From this assumption, our work defines outliers that stand far enough from the gaussian distribution. For gaussian distribution, the distance of a sample x_i to the distribution can be computed using Mahalanobis distance. Mahalanobis distance measures the distance between a point and a distribution [59]. It performs well in the multivariate outlier detection task. The classical Mahalanobis distance of sample x is defined as:

$$d(x, \mu, \Sigma) = \sqrt{(x - \mu)' \Sigma^{-1} (x - \mu)} \tag{11}$$

Σ and μ are covariance and location of the gaussian distribution, respectively, and they must be estimated among the specific data. FASTune uses The Minimum Covariance Determinant (MCD) [60,61] estimator to estimate Σ and μ for its simplicity and ease of computing. The MCD was introduced by P.J.Rousseuw [61]. MCD is a robust estimator of covariance. The basic idea of MCD is to identify a subset of observations in a dataset with the smallest determinant of their covariance matrix, which is less sensitive to the influence of outliers. The MCD estimator is useful for producing reliable estimates of the parameters of the distribution in datasets. According to the definition of MCD-based Mahalanobis distance, the uncommon action detection is described in Algorithm 2.

Algorithm 2 Uncommon Action Detection

```
Execute Initialization to collect enough samples.
Initialize P: an empty points collection.
while Agent recommends configurations do
    Map configurations to a point p.
    Estimate corresponding σ and μ of gaussian distribution using MCD.
    Compute tolerance ellipse with Mahalanobis distance.
    if the position of the sample is within tolerance ellipse then
        Lable the new sample as an outlier and send it to Environment.
    else
        Sent it to Virtual Environment.
    end if
    Add the point to the collection: P = P ∪ p.
end while
```

The Detection process can be summarized as follows:

5.2.1. Collect Samples

To estimate the Σ and μ of the gaussian distribution of the samples, FASTune first collects actions generated by an agent as initial samples. Let L represent the number of samples. The agent interacts with a database L times, producing L number of samples. All actions are considered inliers at this stage because the number of samples has not reached the given threshold, and detection is unavailable. Without enough samples, it is hard to distinguish between an inlier and an outlier. Once L reaches the manually set threshold, this stage ends. Note that the threshold is a hyper-parameter.

5.2.2. Estimate the Distribution

Map the action to an n-dimension multivariate $A = (a_1, a_2, a_3 \dots a_n)$, where a_i represents the value of the i-th configurations of the database. Then estimate the location of the gaussian distribution based on existing samples, specifically, estimates the Σ and μ of gaussian distribution using samples collected in the last stage.

5.2.3. Set a Threshold

A threshold is set to identify data points that are significantly far away from the other data points. FASTune computes the tolerance ellipse with a 97.5 percentile point. The 97.5% tolerance ellipse is defined as:

$$RD(x) = \sqrt{(x - \hat{\mu}_{MCD})^t \hat{\Sigma}_{MCD}^{-1} (x - \hat{\mu})} = \sqrt{\chi^2_{\rho,0.975}} \tag{12}$$

$[\chi^2_\rho, \alpha]$ stands for the α-quantile of the $[\chi^2_\rho]$ distribution. Here, variable $\hat{\Sigma}_{MCD}^{-1}$ is the MCD estimated location, and the point outside the ellipse is considered an outlier.

5.2.4. Identify

Data points that are above the threshold (i.e., outside the ellipse) are identified as outliers and are considered to be significantly different from the other data points in the dataset. Note that, The Accurate MCD estimator is hard to compute since it requires the evaluation of all subsets [61]. The FAST-MCD [62] algorithm is borrowed to improve the estimator's speed. Figure 3 shows the outlier detection results. There are 71 dimensions in action and we choose two for visualization.

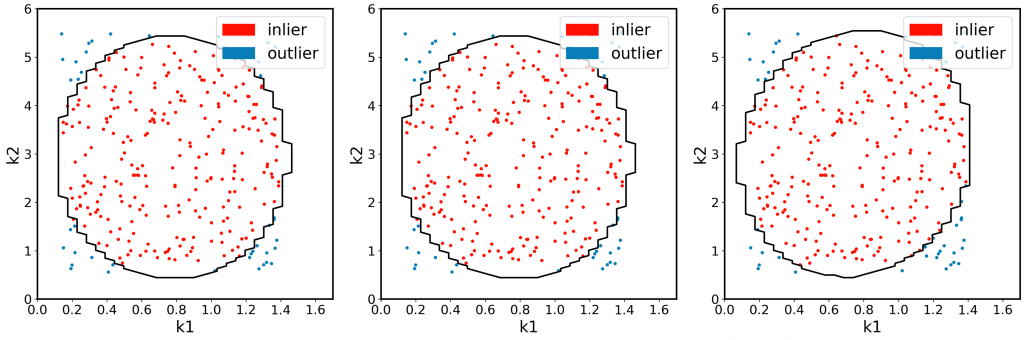

Figure 3. Results of outlier detection on two dimensions, $k1$ denotes *table_open_cache* and $k2$ denotes *innodb_adaptive_max_sleep_delay*.

The results show that the actions generated by the agent tend to be gaussian distributed when the number of iterations is large enough. Actions inside the tolerance ellipse (the red points) are considered common actions dominating the majority. Note that the Dispatcher does not split the action; Dispatcher sends the whole action to the environment or the virtual environment for each interaction.

5.2.5. Update

To increase the accuracy of detection, the new action from an agent in each iteration will be added to the samples collection, so the estimate of Σ and μ are also updated as the collection is updated. In summary, outlier detection enables Dispatcher to distinguish between common and uncommon actions, and the Dispatcher distributes common actions to the virtual environment and uncommon actions to the environment. The sample in the collection can also be used to train the virtual environment, and more details will be discussed below.

This Distance-based method can be computationally expensive for calculating the distance. However, these methods can be effective at identifying outliers that are located far away from the other data points.

5.3. Virtual Environment

In the reinforcement learning framework, the update of agent depends on the reward. Specifically, in database tuning problems, the existing methods calculate the reward by running a stress test on database, which is rather time-consuming. The tuning time of the state-of-the-art approach for optimal configurations can take hours, which can be the bottleneck for tuning efficiency. CDBTune and Hunter [3,7] use cloned databases to make the evaluation parallelization to reduce tuning time efficiently; however, it also brings a heavy burden because each database instance requires additional threads, disk space, and memory.

To address this challenge, we proposed a virtual environment which is an equivalent but faster approach. The virtual environment and the environment are essentially different; virtual environment is a neural network that estimates the evaluation of action based on historical data. It does not have the functionality of a real database instance (i.e., execute queries). The environment is the real database to be tuned, the environment runs a benchmark to evaluate the action, and the reward is calculated from the evaluation. If the database performance is improved after deploying an action, the reward will be positive; otherwise, it will be negative depending on the extent of performance improvement or reduction. For the reinforcement learning framework, virtual environment and environment play the same role in the training process and have the same input and output. In FASTune, both virtual environment and environment are wrapped with a proxy, and proxy providing an interface for the agent to interact. We now describe how to train the virtual environment.

Training the Virtual Environment

Training data of the virtual environment is a collection of tuples $Tv = < S_w, S_d, A, r >$, where S_w is a vector that represents the states of workload, S_d is the state of database, A is the action from agent, and r is the reward. For each $< S_w, S_d, A >$, the virtual environment aims to output a value $r\prime$ which is close to r. The training data can be obtained between the environment and the agent in each trajectory. Given S_d and S_w, then the agent output A (action) and sent to environment proxy. Proxy returns calculated r (reward) based on an evaluation of A, and the r will be recorded.The virtual environment is a multi-layer neural network model consisting of four fully connected layers.The input layer receives a vector that combines S_w, S_d, and A and output a higher dimension tensor to the two hidden layers, using a non-linear function to transform data. The output is a vector representing throughput and latency. The neural network can be viewed as a chain of functions that convert the input into an output pattern [63,64]. To prevent the network from solely learning linear transformations, Tanh, a commonly used activation function in neural networks, is introduced into the hidden layers to capture more complex patterns. The network's weights are initialized using a standard normal distribution. Given a training dataset $U = < W, S, A, r >$, The objective of training is to minimize the loss function, which is defined as follows:

$$L = \frac{1}{2}\sum_{i=1}^{U}|r\prime - r|^2 \tag{13}$$

The output value $r\prime$ produced by the Virtual Environment for a given $< W, S, A >$. To train the Virtual Environment, The adam optimization algorithm [65] is introduced, which is a stochastic optimization algorithm that updates the network weights by computing the first and second moments of the gradients using a stochastic objective function. The training process is terminated when the model has converged or has reached a specified number of steps. The training process spend about 200 s on average and the details on virtual environment are list in Table 3.

Table 3. Details of Virtual Environment.

Item	Description
Dimensions of layers	$33 \rightarrow 32 \rightarrow 16 \rightarrow 1$
Train data size	4094
Test data size	1000
Valid ratio	0.2
Number of epochs	3000
Batch size	32
Learning rate	1×10^{-5}
Early stop	300

6. Workload State

Several studies have shown that workloads are sensitive to various knobs. As shown in Figure 4, the performance of different workloads using the same configurations is varied, and performance does not change linearly in any direction because knobs have non-linear correlations. Qtune encodes queries to capture the workload information to provide a query-aware tuning system. Hunter bounces back quickly from throughput plummet when workload drifts by learning from historical data. They perform well on given workloads but fail to handle the dynamic workload. It can also cause dramatic fluctuations in performance if the database tries to apply configurations when the workload drifts. It is well-known that workload may shift over time in a production environment, which poses a challenge to the stability of the system. To address this challenge, we extract features from the predicted future workload that comes in a short time (e.g., one minute). Furthermore, feed these features as workload state to an agent. Agent not only considers the state of database but also the state of the workload when generating configurations.

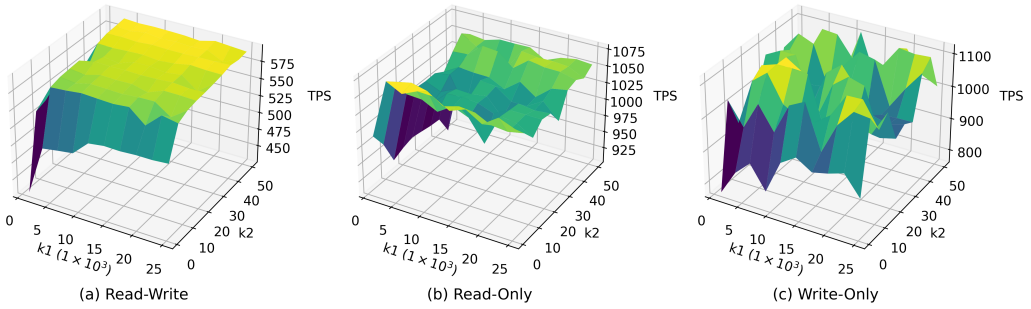

Figure 4. Throughput on different workload types using consistent configurations, $k1$ denotes *innodb_change_buffer_max_size*, and $k2$ denotes *table_open_cache*.

Our work builds a forecasting model to predict the types of queries and how many of them will arrive in the database. The predicted results are used as workload state, which will be sent to the agent as part of the multi-state. The agent can then use this data to find optimal configurations for the dynamic workload. Our approach first encodes each query into a vector and aggregates a batch of queries together to approximate the workload pattern. Then similar patterns will be combined into several groups using a clustering algorithm. Finally, models predict how many quires will arrive at each group. Note that predicting exact SQL (Structured Query Language) statements may lead to expensive computing, so our method forecasts the number of queries in each group.

6.1. Encoder

In general, a SQL statement can be divided into four types: insert, delete, select, and update, different queries may involve different tables. Both query type and tables related to the query significantly impact the database performance. For example, OLAP (Online Analytical Processing) usually involves large numbers of records, while OLTP only involves a few records and executes simple updates, insertions, and deletions in databases. Tables with different structures and sizes also affect the database performance. FASTune capture that information in the vector. The Encoder first extracts the template (i.e., prepared statements) from queries. FASTune uses DBMS's SQL parser (e.g., PostgreSQL-parser) to map SQL statements to an abstract syntax tree to get a standard query template. Encoder counts the number of queries in an established time interval and saves the final result at the end of each time interval. The time interval can be a hyper-parameter. Too short an interval can lead to expensive calculations and fewer performance gains. Conversely, If the time interval is too long, it will be difficult to promptly detect workload drift. FASTune makes a trade-off between speed and accuracy and manually sets the time interval to 10 s. We will leave choosing the time interval automatically for future work.

6.2. Cluster

Although Query Encoder decreases queries by converting queries to templates, it is still a heavy burden to predict how many and what kind of templates will arrive in the future. Thus, FASTune further clusters similar templates into a group to reduce the number of those templates. After gaining the template of queries, many algorithms can cluster the templates. We chose DBSCAN [66] and made some improvements to it. Compared to the original DBSCAN algorithm, our approach made a trade-off between accurate and computational costs by setting a threshold t. The threshold t determined how similar the templates must be to be in the same group. Higher t means more templates will be clustered together so the result can be more precise, yet it will lead to longer computational time. We map each template to a point and use DBSCAN to group these points close to nearby neighbors according to the distance measurement.

6.3. Forecaster

The Encoder and Cluster convert SQL statements to templates and cluster templates into groups. The forecaster predicts the arrival rate of each group's queries. The forecaster aims to predict queries in a near-term (e.g., 10 s) so that the agent can take future workload into account when recommending configurations. Linear models earn our trust since they consume fewer computing resources, require fewer samples, and usually perform well in the near term predicting [67].

7. Evaluation

7.1. Evaluation of Efficiency

In this section, we compare FASTune with state-of-the-art methods [2,3,7,15] on OpenGauss and MySQL, and the method we compared are listed below:

1. QTune is a query-aware tuning system that supports three database tuning granularities [12]. The evaluation uses its workload-level tuning;
2. CDBTune adopts a reinforcement learning model to tune the database [9]. The agent inputs internal metrics of the database and outputs optimal configurations;
3. ResTune uses Bayesian Optimization to optimize resource utilization with constraints [15]. It uses a meta-learning approach to extract useful knowledge from historical experiences;
4. HUNTER designed a hybrid tuning model that uses samples from a genetic algorithm to accelerate the exploration of deep reinforcement learning [10]. Meanwhile, HUNTER uses Random Forest, Principal Component Analysis, and Fast Exploration Strategy to reduce the action space.

Details of the hardware are listed in Table 4.

Table 4. Details of hardware used in the evaluation.

DBMS	CPU	RAM	Disk Speed	Disk Latency
OpenGauss	4 core	16 GB	220 Mb/s	12.3 ms
MySQL	4 core	16 GB	220 Mb/s	12.3 ms

Both MySQL and OpenGauss run on a server with a 4 core CPU, 16 GB RAM, and a 200 GB disk. A virtual machine (VM) is deployed to keep the experimental environment consistent. We first create a snapshot that records the initial state of the operating system (OS) and hardware. At each evaluation, the virtual machine rolls back to a snapshot to provide the same experimental conditions. A static workload (e.g., TPC-C and Sysbench) is used to evaluate FASTune and compare it to the methods mentioned above. The evaluation is based on two dimensions: tuning time and database performance improvement. In Section 7.2, a dynamic workload is used to evaluate the stability of FASTune. We count the number of dangerous configurations and database fluctuations during the tuning process. Figure 5 shows the results of Sysbench and TPC-C running on MySQL and openGauss.

Figure 5 illustrates the similarity between the tuning results of MySQL and openGauss. In comparison to other tuning systems, FASTune achieves optimal performance in a shorter duration. By finding the optimal configurations within 3 h, FAStune significantly reduces the tuning time. It is worth noting that all methods resulted in a TPC-C throughput of 6% to 18% higher on openGauss which could be due to database version limits. The experiment results depict that the proposed model enhances database performance by 1–1.5 times within 5 h, requiring substantially lower time (60–80%) for optimal performance attainment as compared to FastTune conventional models. During the TPC-C test, the database platform was able to achieve a throughput of up to 6000 txn/s, while openGauss was able to secure up to 7800 txn/s. The higher throughput on openGauss may have resulted from inherent database variations. Due to databases' complex and random nature, fluctuations may occur after the intelligent agent's convergence, as observed in Figure 5a, where performance often oscillates between 2500 and 3000.

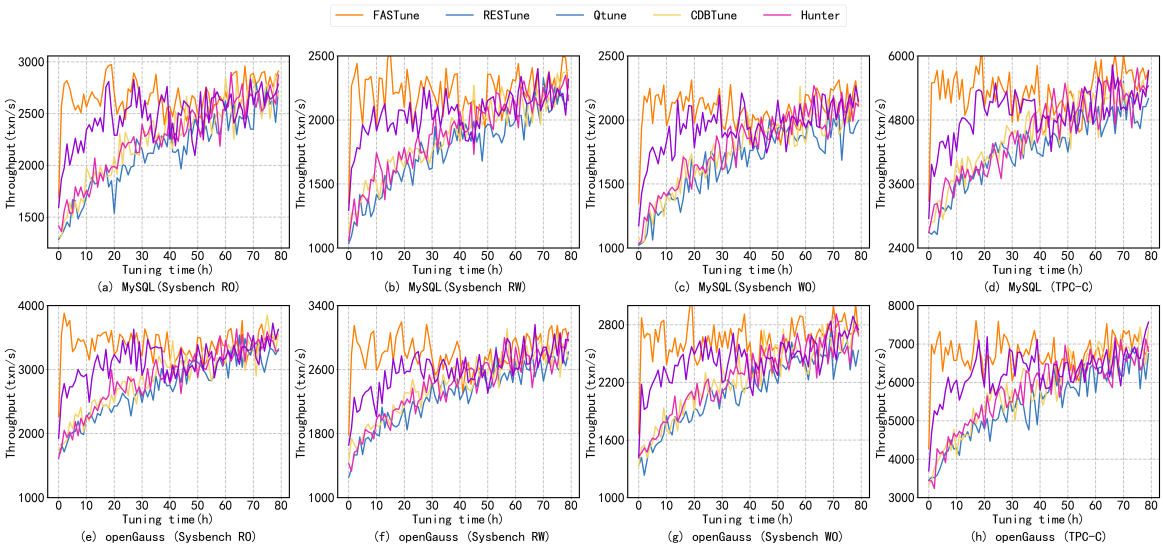

Figure 5. Performance Evaluation—Comparisons of state-of-the-art database tuning systems, (**a**–**d**) demonstrate throughput of MySQL on Sysbench and TPC-C. (**e**–**h**) demonstrate throughput of OpenGauss on Sysbench and TPC-C.

Figure 6 shows the count of configurations that lead to a sharp performance decline (#Dangerous) and database failure (#Failure) during the tuning process.

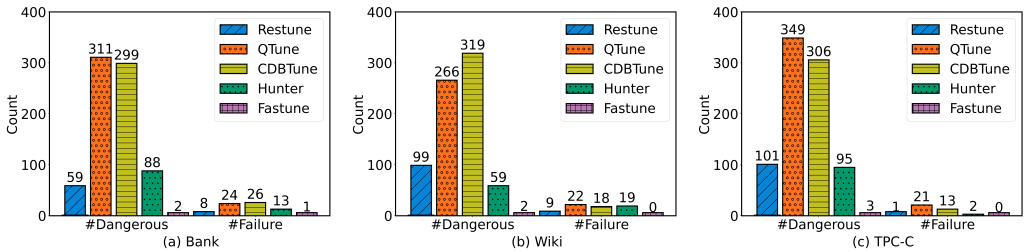

Figure 6. Dangerous configurations statistics during the tuning process. (**a**–**c**) show the number of dangerous and failure configurations under workloads from Bank, Wiki, and TPC-C on openGauss.

The state-of-the-art methods have 59 to 349 dangerous configurations within 1000 tuning intervals. Instead, the dangerous configurations occurred less than ten times. FASTune achieves this by using an action filter to evaluate and discards dangerous configurations directly. To sum up, FASTune can recommend optimal configurations and keep the database stable during the tuning. The best result of FASTune is slightly better than other methods, but the time used is much less.

7.2. Evaluation of Dynamic Workload

In this section, the dynamic workloads are used to evaluate the stability of the tuning system, and BenchBase is used to construct dynamic workloads by switching transaction types, BenchBase is a SQL benchmark framework. The workload shift between Sysbench (RW) and Sysbench (RO); Sysbench (RO) is an OLTP workload that contains heavy write queries, and Sysbench (RW) is a mixed workload with reading and writing. Workload starts with Sysbench (RO) and continues for hours, then switches to Sysbench (RW) and continues for hours. Note that our work train workload forecasters for FASTune before evaluation. Figure 7 shows the throughput of database.

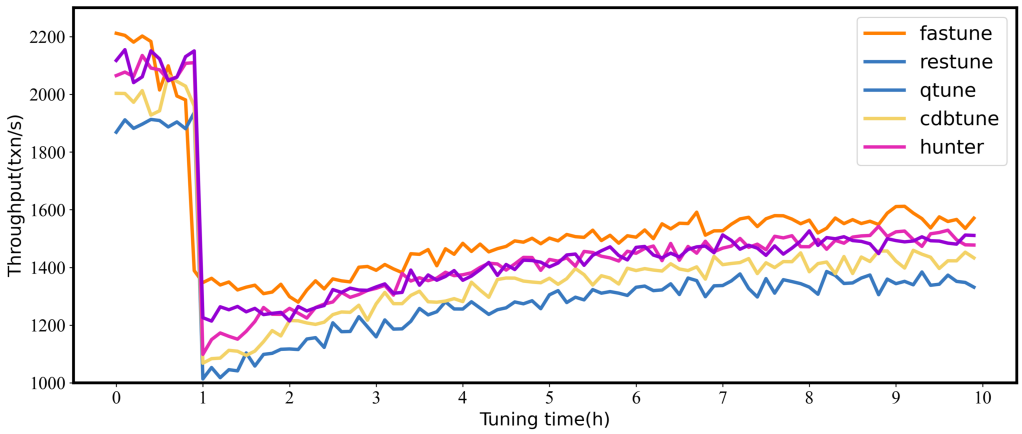

Figure 7. The changes of throughput with workload drift—Comparisons of state-of-the-art tuning systems.

Workload first holds on Sysbench (RO) for one hour, and then switches to Sysbench (RW). The workload drift causes a performance degradation, and all throughput drops below 1200 txn/s except for FASTune. FASTune suffers the least from the drift because FASTune predicts workload drift and adjusts in advance. Note that FASTune has experienced a slight performance degradation as a by-product ahead of the drift. Another potential advantage of FASTune is that FASTune can provide warning before workload drift arrives. We would like to implement this feature in future work.

8. Conclusions

The paper introduces a novel tuning system called FASTune, which is designed to recommend optimal database configurations that can enhance database performance. FASTune employs soft actor–critic networks as an agent to achieve fast and stable exploration. Environment Proxy has been proposed to provide a gateway between the agent and environment, preventing dangerous action arising from the agent. Additionally, FASTune uses a virtual environment to enhance the effectiveness of evaluating configurations. To maintain stability, a workload forecaster based on machine learning is proposed, and the expected queries are fed to the agent to handle the dynamic workload. The experimental results show that FASTune can find optimal database configurations to improve performance while maintaining stability in the tuning. However, FASTune requires collecting workloads in users' environment, which may result in data privacy issues, and the training of the forecaster incurs a non-negligible extra cost. We plan to explore more effective approaches to better support dynamic workload tuning.

Author Contributions: Conceptualization, T.L. and C.L.; Data curation, T.L.; Formal analysis, T.L. and C.L.; Funding acquisition, L.S., L.W., Y.T., C.L. and Y.G.; Investigation, T.L.; Methodology, T.L.; Project administration, L.S., C.L. and Y.G.; Resources, L.S., L.W. and Y.T.; Software, T.L.; Supervision, L.S., C.L. and Y.G.; Validation, T.L.; Visualization, T.L.; Writing—original draft, T.L.; Writing—review and editing, L.S. and C.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the the National Key Technologies R&D Program (2018YFB1701401), Key Project of Public Benefit in Henan Province of China (201300210500), Nature Science Foundation of China (62006210, 62001284, 62206252), Key Scientific Research Projects of Colleges and Universities in Henan Province (23A520015), Key Project of Collaborative Innovation in Nanyang (22XTCX12001), Key Technology Project of Henan Province of China (221100210100), and Research Foundation for Advanced Talents of Zhengzhou University (32340306).

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

DBMS	Database Management System
RL	Reinforcement Learning
DBAs	Database Administrators
OLTP	Online Transaction Processing
OLAP	Online Analytical Processing
MS-SAC	Multi-State Soft Actor–Critic
ML	Machine Learning
RBF	Radial Basis Function
SAC	Soft Actor–Critic
DDGP	Deep Deterministic Policy Gradient
SVM	Support Vector Machine
MCD	Minimum Covariance Determinant
MLE	Maximum Likelihood Estimate
RO	Read Only
RW	Read Write

References

1. Belknap, P.; Dageville, B.; Dias, K.; Yagoub, K. Self-Tuning for SQL Performance in Oracle Database 11g. In Proceedings of the 2009 IEEE 25th International Conference on Data Engineering, Shanghai, China, 29 March–2 April 2009; pp. 1694–1700. [CrossRef]
2. Li, G.; Zhou, X.; Li, S.; Gao, B. QTune: A Query-Aware Database Tuning System with Deep Reinforcement Learning. *Proc. VLDB Endow.* **2019**, *12*, 2118–2130. [CrossRef]
3. Cai, B.; Liu, Y.; Zhang, C.; Zhang, G.; Zhou, K.; Liu, L.; Li, C.; Cheng, B.; Yang, J.; Xing, J. HUNTER: An Online Cloud Database Hybrid Tuning System for Personalized Requirements. In Proceedings of the 2022 International Conference on Management of Data. ACM, Philadelphia, PA, USA, 12–17 June 2022; pp. 646–659. [CrossRef]
4. Zhu, Y.; Liu, J.; Guo, M.; Bao, Y.; Ma, W.; Liu, Z.; Song, K.; Yang, Y. BestConfig: Tapping the Performance Potential of Systems via Automatic Configuration Tuning. In Proceedings of the 2017 Symposium on Cloud Computing. Association for Computing Machinery, SoCC '17, Santa Clara, CA, USA, 24–27 September 2017; pp. 338–350. [CrossRef]
5. Marco, A.; Berkenkamp, F.; Hennig, P.; Schoellig, A.P.; Krause, A.; Schaal, S.; Trimpe, S. Virtual vs. Real: Trading off Simulations and Physical Experiments in Reinforcement Learning with Bayesian Optimization. In Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 29 May–3 June 2017; pp. 1557–1563. [CrossRef]
6. Wei, Z.; Ding, Z.; Hu, J. Self-Tuning Performance of Database Systems Based on Fuzzy Rules. In Proceedings of the 2014 11th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), Xiamen, China, 19–21 August 2014; pp. 194–198. [CrossRef]
7. Zhang, J.; Zhou, K.; Li, G.; Liu, Y.; Xie, M.; Cheng, B.; Xing, J. CDBTune$^+$: An Efficient Deep Reinforcement Learning-Based Automatic Cloud Database Tuning System. *VLDB J.* **2021**, *30*, 959–987. [CrossRef]
8. Van Aken, D.; Pavlo, A.; Zhang, B.; Gordon, G.J. Automatic Database Management System Tuning Through Large-scale Machine Learning. In Proceedings of the 2017 ACM International Conference on Management of Data, ACM, Chicago, IL, USA, 14–19 May 2017; pp. 1009–1024. [CrossRef]
9. Zhang, X.; Wu, H.; Li, Y.; Tan, J.; Li, F.; Cui, B. Towards Dynamic and Safe Configuration Tuning for Cloud Databases. In Proceedings of the 2022 International Conference on Management of Data, ACM, Philadelphia, PA, USA, 12–17 June 2022; pp. 631–645. [CrossRef]
10. Zhang, J.; Liu, Y.; Zhou, K.; Li, G.; Xiao, Z.; Cheng, B.; Xing, J.; Wang, Y.; Cheng, T.; Liu, L.; 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, ACM, Amsterdam, The Netherlands, 30 June–5 July 2019; pp. 415–432. [CrossRef]
11. Trummer, I. DB-BERT: A Database Tuning Tool That "Reads the Manual". In Proceedings of the 2022 International Conference on Management of Data. Association for Computing Machinery, SIGMOD '22, Philadelphia, PA, USA, 12–17 June 2022; pp. 190–203. [CrossRef]
12. Duan, S.; Thummala, V.; Babu, S. Tuning Database Configuration Parameters with iTuned. *Proc. VLDB Endow.* **2009**, *2*, 1246–1257. [CrossRef]
13. Francois-Lavet, V.; Henderson, P.; Islam, S.; Bellemare, M.G.; Pineau, J. An Introduction to Deep Reinforcement Learning. *Found. Trends® Mach. Learn.* **2018**, *11*, 219–354. [CrossRef]
14. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*, 2nd ed.; Adaptive Computation and Machine Learning Series; The MIT Press: Cambridge, MA, USA, 2018.
15. Zhang, X.; Wu, H.; Chang, Z.; Jin, S.; Tan, J.; Li, F.; Zhang, T.; Cui, B. ResTune: Resource Oriented Tuning Boosted by Meta-Learning for Cloud Databases. In Proceedings of the 2021 International Conference on Management of Data, Virtual Event, 20–25 June 2021; pp. 2102–2114. [CrossRef]

16. Basu, D.; Lin, Q.; Chen, W.; Vo, H.T.; Yuan, Z.; Senellart, P.; Bressan, S. Regularized Cost-Model Oblivious Database Tuning with Reinforcement Learning. In *Transactions on Large-Scale Data- and Knowledge-Centered Systems XXVIII: Special Issue on Database- and Expert-Systems Applications*; Hameurlain, A., Küng, J., Wagner, R., Chen, Q., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2016; pp. 96–132. [CrossRef]
17. Gelbart, M.A.; Snoek, J.; Adams, R.P. Bayesian Optimization with Unknown Constraints. *arXiv* **2014**, arXiv:1403.5607. [CrossRef]
18. Berkenkamp, F.; Krause, A.; Schoellig, A.P. Bayesian Optimization with Safety Constraints: Safe and Automatic Parameter Tuning in Robotics. *arXiv* **2020**, arXiv:cs/1602.04450. [CrossRef]
19. Sui, Y.; Gotovos, A.; Burdick, J.; Krause, A. Safe Exploration for Optimization with Gaussian Processes. In Proceedings of the 32nd International Conference on Machine Learning. PMLR, Lille, France, 6–11 July 2015; pp. 997–1005.
20. Zolaktaf, Z.; Milani, M.; Pottinger, R. Facilitating SQL Query Composition and Analysis. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. Association for Computing Machinery, SIGMOD '20, Portland, OR, USA, 14–19 June 2020; pp. 209–224. [CrossRef]
21. Liberty, E.; Karnin, Z.; Xiang, B.; Rouesnel, L.; Coskun, B.; Nallapati, R.; Delgado, J.; Sadoughi, A.; Astashonok, Y.; Das, P.; et al. Elastic Machine Learning Algorithms in Amazon SageMaker. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. Association for Computing Machinery, SIGMOD '20, Portland, OR, USA, 14–19 June 2020; pp. 731–737. [CrossRef]
22. Tan, J.; Nayman, N.; Wang, M. CobBO: Coordinate Backoff Bayesian Optimization with Two-Stage Kernels. *arXiv* **2022**, arXiv:2101.05147. [CrossRef]
23. Mockus, J. Global Optimization and the Bayesian Approach. In *Bayesian Approach to Global Optimization: Theory and Applications*; Mockus, J., Ed.; Mathematics and Its Applications; Springer: Cham, The Netherlands, 1989; pp. 1–3. [CrossRef]
24. Tan, J.; Zhang, T.; Li, F.; Chen, J.; Zheng, Q.; Zhang, P.; Qiao, H.; Shi, Y.; Cao, W.; Zhang, R. iBTune: Individualized Buffer Tuning for Large-Scale Cloud Databases. *Proc. VLDB Endow.* **2019**, *12*, 1221–1234. [CrossRef]
25. Yan, J.; Jin, Q.; Jain, S.; Viglas, S.D.; Lee, A. Snowtrail: Testing with Production Queries on a Cloud Database. In Proceedings of the Workshop on Testing Database Systems, DBTest'18, Houston, TX, USA, 15 June 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 1–6. [CrossRef]
26. Liu, J.; Zhang, C. Distributed Learning Systems with First-order Methods. *arXiv* **2021**, arXiv:2104.05245. [CrossRef]
27. Galakatos, A.; Markovitch, M.; Binnig, C.; Fonseca, R.; Kraska, T. FITing-Tree: A Data-aware Index Structure. In Proceedings of the 2019 International Conference on Management of Data, SIGMOD '19, Amsterdam, The Netherlands, 30 June–5 July 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 1189–1206. [CrossRef]
28. Kraska, T.; Beutel, A.; Chi, E.H.; Dean, J.; Polyzotis, N. The Case for Learned Index Structures. In Proceedings of the 2018 International Conference on Management of Data, SIGMOD '18, Houston, TX, USA, 10–15 June 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 489–504. [CrossRef]
29. Ma, L.; Van Aken, D.; Hefny, A.; Mezerhane, G.; Pavlo, A.; Gordon, G.J. Query-Based Workload Forecasting for Self-Driving Database Management Systems. In Proceedings of the 2018 International Conference on Management of Data, ACM, Houston, TX, USA, 10–15 June 2018; pp. 631–645. [CrossRef]
30. Ma, L.; Zhang, W.; Jiao, J.; Wang, W.; Butrovich, M.; Lim, W.S.; Menon, P.; Pavlo, A. MB2: Decomposed Behavior Modeling for Self-Driving Database Management Systems. In Proceedings of the 2021 International Conference on Management of Data, SIGMOD '21, Virtual Event, 20–25 June 2021; Association for Computing Machinery: New York, NY, USA, 2021; pp. 1248–1261. [CrossRef]
31. Sadri, Z.; Gruenwald, L.; Leal, E. Online Index Selection Using Deep Reinforcement Learning for a Cluster Database. In Proceedings of the 2020 IEEE 36th International Conference on Data Engineering Workshops (ICDEW), Dallas, TX, USA, 20–24 April 2020; pp. 158–161. [CrossRef]
32. Schnaitter, K.; Polyzotis, N. Semi-Automatic Index Tuning: Keeping DBAs in the Loop. *arXiv* **2010**, arXiv:1004.1249. [CrossRef]
33. Van Aken, D.; Yang, D.; Brillard, S.; Fiorino, A.; Zhang, B.; Bilien, C.; Pavlo, A. An Inquiry into Machine Learning-Based Automatic Configuration Tuning Services on Real-World Database Management Systems. *Proc. VLDB Endow.* **2021**, *14*, 1241–1253. [CrossRef]
34. Kunjir, M.; Babu, S. Black or White? How to Develop an AutoTuner for Memory-based Analytics. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, SIGMOD '20, Portland, OR, USA, 14–19 June 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 1667–1683. [CrossRef]
35. Fekry, A.; Carata, L.; Pasquier, T.; Rice, A.; Hopper, A. Tuneful: An Online Significance-Aware Configuration Tuner for Big Data Analytics. *arXiv* **2020**, arXiv:2001.08002. [CrossRef]
36. Fekry, A.; Carata, L.; Pasquier, T.; Rice, A.; Hopper, A. To Tune or Not to Tune? In Search of Optimal Configurations for Data Analytics. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD '20, Virtual Event, 6–10 July 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 2494–2504. [CrossRef]
37. Storm, A.J.; Garcia-Arellano, C.; Lightstone, S.S.; Diao, Y.; Surendra, M. Adaptive Self-Tuning Memory in DB2. In Proceedings of the 32nd International Conference on Very Large Data Bases, VLDB Endowment, VLDB '06, Seoul, Republic of Korea, 12–15 September 2006; pp. 1081–1092.
38. Tran, D.N.; Huynh, P.C.; Tay, Y.C.; Tung, A.K.H. A New Approach to Dynamic Self-Tuning of Database Buffers. *ACM Trans. Storage (TOS)* **2008**, *4*, 3:1–3:25. [CrossRef]

39. Yoon, D.Y.; Niu, N.; Mozafari, B. DBSherlock: A Performance Diagnostic Tool for Transactional Databases. In Proceedings of the 2016 International Conference on Management of Data, SIGMOD '16, San Francisco, CA, USA, 26 June–1 July 2016; Association for Computing Machinery: New York, NY, USA, 2016; pp. 1599–1614. [CrossRef]
40. Kanellis, K.; Alagappan, R.; Venkataraman, S. Too Many Knobs to Tune? Towards Faster Database Tuning by Pre-Selecting Important Knobs. In Proceedings of the 12th USENIX Conference on Hot Topics in Storage and File Systems HotStorage'20, Virtul, 13–14 July 2020; p. 16.
41. Ni, Z.; He, H.; Zhao, D.; Prokhorov, D.V. Reinforcement Learning Control Based on Multi-Goal Representation Using Hierarchical Heuristic Dynamic Programming. In Proceedings of the The 2012 International Joint Conference on Neural Networks (IJCNN), Brisbane, Australia, 10–15 June 2012; pp. 1–8. [CrossRef]
42. Nowé, A.; Brys, T. A Gentle Introduction to Reinforcement Learning. In Proceedings of the Scalable Uncertainty Management-10th International Conference, SUM 2016, Nice, France, 21–23 September 2016; Schockaert, S., Senellart, P., Eds.; Springer: Berlin/Heidelberg, Germany, 2016; Volume 9858, pp. 18–32. [CrossRef]
43. Mazyavkina, N.; Sviridov, S.; Ivanov, S.; Burnaev, E. Reinforcement Learning for Combinatorial Optimization: A Survey. *Comput. Oper. Res.* **2021**, *134*, 105400. [CrossRef]
44. Shen, R.; Zhong, S.; Wen, X.; An, Q.; Zheng, R.; Li, Y.; Zhao, J. Multi-Agent Deep Reinforcement Learning Optimization Framework for Building Energy System with Renewable Energy. *Appl. Energy* **2022**, *312*, 118724. [CrossRef]
45. Deng, J.; Sierla, S.; Sun, J.; Vyatkin, V. Reinforcement Learning for Industrial Process Control: A Case Study in Flatness Control in Steel Industry. *Comput. Ind.* **2022**, *143*, 103748. [CrossRef]
46. He, Z.; Tran, K.P.; Thomassey, S.; Zeng, X.; Xu, J.; Yi, C. A Deep Reinforcement Learning Based Multi-Criteria Decision Support System for Optimizing Textile Chemical Process. *Comput. Ind.* **2021**, *125*, 103373. [CrossRef]
47. Zhang, H.; Peng, Q.; Zhang, J.; Gu, P. Planning for Automatic Product Assembly Using Reinforcement Learning. *Comput. Ind.* **2021**, *130*, 103471. [CrossRef]
48. Mikhaylov, A.; Mazyavkina, N.S.; Salnikov, M.; Trofimov, I.; Qiang, F.; Burnaev, E. Learned Query Optimizers: Evaluation and Improvement. *IEEE Access* **2022**, *10*, 75205–75218. [CrossRef]
49. Kraska, T.; Alizadeh, M.; Beutel, A.; Chi, E.H.; Ding, J.; Kristo, A.; Leclerc, G.; Madden, S.R.; Mao, H.; Nathan, V. SageDB: A Learned Database System. Available online: https://dspace.mit.edu/handle/1721.1/132282 (accessed on 18 July 2022).
50. Li, G.; Zhou, X.; Sun, J.; Yu, X.; Han, Y.; Jin, L.; Li, W.; Wang, T.; Li, S. openGauss: An Autonomous Database System. *Proc. VLDB Endow.* **2021**, *14*, 3028–3042. [CrossRef]
51. Haarnoja, T.; Zhou, A.; Hartikainen, K.; Tucker, G.; Ha, S.; Tan, J.; Kumar, V.; Zhu, H.; Gupta, A.; Abbeel, P.; et al. Soft Actor-Critic Algorithms and Applications. *arXiv* **2019**, arXiv:1812.05905.
52. Watkins, C.J.C.H.; Dayan, P. Q-Learning. *Mach. Learn.* **1992**, *8*, 279–292. [CrossRef]
53. Hester, T.; Vecerik, M.; Pietquin, O.; Lanctot, M.; Schaul, T.; Piot, B.; Horgan, D.; Quan, J.; Sendonaris, A.; Osband, I.; et al. Deep Q-learning from Demonstrations. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence, AAAI'18/IAAI'18/EAAI'18, New Orleans, LA, USA, 2–7 February 2018; pp. 3223–3230.
54. Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; Riedmiller, M. Deterministic Policy Gradient Algorithms. In Proceedings of the 31st International Conference on International Conference on Machine Learning—Volume 32, JMLR.org, ICML'14, Beijing, China, 21–26 June 2014; pp. I-387–I-395.
55. Pisner, D.A.; Schnyer, D.M. Chapter 6—Support Vector Machine. In *Machine Learning*; Mechelli, A., Vieira, S., Eds.; Academic Press: Cambridge, MA, USA, 2020; pp. 101–121. [CrossRef]
56. Cervantes, J.; Garcia-Lamont, F.; Rodríguez-Mazahua, L.; Lopez, A. A Comprehensive Survey on Support Vector Machine Classification: Applications, Challenges and Trends. *Neurocomputing* **2020**, *408*, 189–215. [CrossRef]
57. Buhmann, M.D. Radial Basis Functions. *Acta Numer.* **2000**, *9*, 1–38. [CrossRef]
58. Scholkopf, B.; Sung, K.-K.; Burges, C.; Girosi, F.; Niyogi, P.; Poggio, T.; Vapnik, V. Comparing Support Vector Machines with Gaussian Kernels to Radial Basis Function Classifiers. *IEEE Trans. Signal Process* **1997**, *45*, 2758–2765. [CrossRef]
59. Simpson, D.G. Introduction to Rousseeuw (1984) Least Median of Squares Regression. In *Breakthroughs in Statistics*; Kotz, S., Johnson, N.L., Eds.; Springer Series in Statistics; Springer: Berlin/Heidelberg, Germany, 1997; pp. 433–461. [CrossRef]
60. Hubert, M.; Debruyne, M. Minimum Covariance Determinant. *Wiley Interdiscip. Rev. Comput. Stat.* **2010**, *2*, 36–43. [CrossRef]
61. Hubert, M.; Debruyne, M.; Rousseeuw, P.J. Minimum Covariance Determinant and Extensions. *Wiley Interdiscip. Rev. Comput. Stat.* **2018**, *10*, e1421. [CrossRef]
62. Rousseeuw, P.J.; Driessen, K.V. A Fast Algorithm for the Minimum Covariance Determinant Estimator. *Technometrics* **1999**, *41*, 212–223. [CrossRef]
63. Dikaleh, S.; Xiao, D.; Felix, C.; Mistry, D.; Andrea, M. Introduction to Neural Networks. In Proceedings of the 27th Annual International Conference on Computer Science and Software Engineering CASCON '17, Markham, ON, Canada, 6–8 November 2017; p. 299.
64. Abiodun, O.I.; Jantan, A.; Omolara, A.E.; Dada, K.V.; Mohamed, N.A.; Arshad, H. State-of-the-Art in Artificial Neural Network Applications: A Survey. *Heliyon* **2018**, *4*, e00938. [CrossRef]
65. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. In Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, 7–9 May 2015; Bengio, Y., LeCun, Y., Eds.

66. Ester, M.; Kriegel, H.P.; Sander, J.; Xu, X. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, KDD'96, Portland, OR, USA, 2–4 August 1996; pp. 226–231.
67. Akdere, M.; Çetintemel, U.; Riondato, M.; Upfal, E.; Zdonik, S.B. Learning-Based Query Performance Modeling and Prediction. In Proceedings of the 2012 IEEE 28th International Conference on Data Engineering. IEEE Computer Society, ICDE '12, Arlington, VA, USA, 1–5 April 2012; pp. 390–401. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.