TiAD-DQR: Software Aging States Determination and Rejuvenation Decision Generation for Docker Platform

Jing Liu, *Senior Member, IEEE*, Quan Zhou, Xingyu Chen, Jiantao Zhou, *Member, IEEE*, and Keqin Li, *Fellow, IEEE*

***Abstract*—Docker has evolved into core container platform in cloud-native environments. However, it is susceptible to software aging problem after long-time running, which seriously impairs the overall system reliability and performance. Existing aging related research mainly focuses on verifying the presence of software aging phenomena and predicting resource consumption changes caused by it, with insufficient research on how to implement targeted and effective rejuvenation strategies on the Docker platform. This paper proposes an integrated framework, called TiAD-DQR, to comprehensively and effectively mitigate the platform aging challenge, where Trend Decomposition Dense Encoder (TDDE) and Gaussian Mixture Aging Detection (GMAD) are combined for accurate determination of aging states and then to assist in intelligent rejuvenation decision generation based on the Double Q-Learning (DQL). The sufficient experimental results show that our TiAD-DQR can effectively delay the software aging process, maximize system availability, and significantly improve the service quality and system stability of the Docker platform.**

***Index Terms*—Docker platform, aging states determination, rejuvenation decision generation, double Q-learning (DQL).**

I. Introduction

SOFTWARE aging refers to the phenomenon where software systems experience progressively decreased performance and increased failure rates during continuous operation, manifesting as service errors, system stoppages, or prolonged response times. A key reason for this is that although software errors exist as inherent defects from development, their accumulation during operation leads to gradual performance degradation and aging-related faults [1]. With the rapid advancement of cloud-native technologies, Docker has become the mainstream container platform for container creation, deployment, and management. Empirical evidence from Oliveira et al. [2] underscores that resource mismanagement, particularly memory leaks and fragmentation, is prevalent in long-time running Docker environments, often escalating into systemwide performance degradation, more directly, Vinícius et al. [3] in their 2022 experimental study targeting Docker's core runtime component dockerd, using Docker 17.05.0-ce, explicitly demonstrate that the Docker platform suffers from distinct software aging effects in long-running scenarios. This not only renders the platform more susceptible to reliability issues but also escalates into cascading risks. As aging intensifies, critical services may experience sudden performance collapses during peak loads, and mission-critical tasks could face unpredictable interruptions. Additionally, the cumulative effect of prolonged degradation can erode system availability to a point where even basic operations falter, ultimately triggering severe economic losses that extend beyond direct downtime to encompass reputational damage and customer attrition.

Received 8 August 2025; revised 14 October 2025; accepted 23 October 2025. Date of publication 28 October 2025; date of current version 11 December 2025. This work was supported in part by the Natural Science Foundation of China under Grant 62462047, in part by the Natural Science Foundation of Inner Mongolia of China under Grant 2023ZD18, and in part by the Engineering Research Center of Ecological Big Data, Ministry of Education, Inner Mongolia Engineering Laboratory for Cloud Computing and Service Software. *(Corresponding author: Jing Liu.)*

Jing Liu, Quan Zhou, Xingyu Chen, and Jiantao Zhou are with the College of Computer Science, Inner Mongolia University, Hohhot 010021, China (e-mail: liujing@imu.edu.cn; wr_z0930_keyan@outlook.com; 32409064@mail.imu.edu.cn; cszjtao@imu.edu.cn).

Keqin Li is with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA (e-mail: lik@newpaltz.edu).

Digital Object Identifier 10.1109/TSC.2025.3625955

To effectively address software aging on the Docker platform, there is a need to accurately identify aging states and develop reasonable rejuvenation strategies to reduce downtime. However, traditional rejuvenation measures primarily rely on rebooting, yet such operations may increase downtime and compromise system availability [4]. Thus, there is a need to develop strategies that ensure system state restoration, reduce rejuvenation costs, and maintain service quality and stability. In view of this, this paper proposes an integrated framework named TiAD-DQR, designed for the Docker platform. The framework combines the TDDE-GMAD approach, which integrates the TDDE and GMAD, for precise aging state identification. It also employs a reinforcement learning algorithm based on DQL [5] to generate intelligent rejuvenation decisions, maximizing Docker platform availability by balancing immediate response and long-term sustainability. The principal contributions of this work are as follows.

- Based on the TDDE model, we developed an aging trend prediction approach. Existing time-series models for Docker aging often fail to filter non-aging fluctuations or capture long-term patterns, leading to low accuracy. Our TDDE uses trend decomposition to eliminate noise and a dense encoder to extract long-term features, boosting long-term prediction precision.
- Based on the GMAD model, we proposed an aging states determination method. Traditional methods rely on

manual thresholds, which are inflexible for dynamic Docker environments and cause misjudgments. Our GMAD combines unsupervised anomaly detection and clustering to auto-generate thresholds, precisely dividing Docker into four aging states without manual intervention.

- Using the DQL algorithm, we designed a rejuvenation decision approach. Existing Docker rejuvenation mostly uses fixed operations like periodic restarts, ignoring actual aging states and causing unnecessary downtime. Our DQL defines state space with current and predicted aging states, builds a reward matrix balancing immediate and long-term benefits, and selects optimal actions to reduce downtime.
- The TiAD-DQR framework integrates these three modules into a closed-loop system, different from fragmented existing solutions. TDDE provides accurate predictions for GMAD, GMAD guides DQL's action selection, and DQL's decisions update the system state. This synergy comprehensively addresses Docker's software aging, improving the platform's reliability, service quality and stability.

II. Related Work

This section systematically reviews and classifies existing studies into two main categories: cloud-based aging detection methods and rejuvenation decision methods.

A. Cloud-Based Aging Detection Methods

Cloud-based aging detection methods primarily rely on using time series models to predict software resource usage, assisting in forecasting potential future aging issues by analyzing resource consumption trends.

Early studies focused on traditional statistical and machine learning models for specific scenarios. Grottke et al. analyzed software aging issues in Web servers, such as memory leaks and resource exhaustion, and provided a series of monitoring and mitigation strategies [6]. Magalhães and Silva developed a predictive model based on ARIMA and Holt-Winters for predicting performance anomalies in web applications due to software aging [7]. Jia et al. used multivariate linear regression to analyze the aging trends of software in Web servers and discussed how to use these prediction models to optimize rejuvenation strategies [8]. Yan et al. in two papers used SVM and a hybrid method combining ARIMA and ANN, respectively, to predict resource consumption in IIS Web servers, aiming to enhance prediction accuracy [9][10]. Additionally, Yan and Guo provided a practical guide to predicting software aging in Web servers using machine learning technologies, offering an effective resource consumption prediction method for potential future aging issues by comparing models such as ARIMA, ANN, and SVM [11].

Subsequent research extended to cloud, edge, and containerized environments. Chen et al. developed a tool named ARF-Predictor, designed to forecast failures and resource consumption trends in cloud-based application software, combining multiple algorithms and technologies to enhance the accuracy and efficiency of prediction [12]. Sudhakar et al. used an ANN model to predict aging and failure times of virtual machines in cloud systems [13]. Andrade et al. studied software aging phenomena in continuously running image classification systems within cloud and edge computing environments, using statistical analysis methods to detect trends in memory usage growth in cloud and edge computing systems [14]. Oliveira et al. investigated software aging phenomena in container-based virtualized environments, particularly Docker, using time series models to predict the progression of resource consumption caused by software aging [15]. Xie et al. used a hybrid model to real-time predict resource loads of Docker containers, to identify potential performance degradation and resource depletion [16].

For deep learning-based prediction models, recent advanced methods such as FEDformer [17], Autoformer [18], Informer [19], and Transformer [20] have shown effectiveness in long-term time series forecasting.

In summary, existing cloud-based aging detection methods predominantly rely on single-step prediction, focusing on single-step accuracy but fails to capture long-term aging trends in time-series data. Although deep learning models are capable of extracting long-term features, they are not tailored to software aging scenarios and often underperform when handling non-stationary, aging-specific patterns in Docker platforms. To overcome these limitations, our proposed TDDE model incorporates trend decomposition and long-term feature extraction, effectively reducing error accumulation and enhancing long-term prediction accuracy.

B. Cloud-Based Rejuvenation Decision Methods

Cloud-based rejuvenation decision methods primarily rely on traditional rebooting and virtualization-based migration techniques aimed at mitigating aging phenomena by optimizing resource allocation, thus enhancing system performance and stability.

Studies focusing on virtualization-based strategies have explored various approaches to address aging. Silva et al. proposed a virtualization-based automated software rejuvenation plan by continuously monitoring system performance metrics, effectively preventing transient faults and software aging [21]. Puliafito explored time-based rejuvenation strategies for virtual machine monitors, which optimize the operation and availability of the monitors by collecting and analyzing resource usage data [22]. Fakhrolmobasheri et al. proposed a power-aware software rejuvenation strategy for cloud data centers, analyzing the rejuvenation effects on virtual machine monitors using a stochastic activity network model to enhance energy efficiency and system stability [13]. Torquato et al. studied the use of virtual machine migration as a rejuvenation strategy to improve the availability and reliability of cloud systems, evaluating its positive impact on overall system performance through simulations of various workloads [23].

Research on application and component-level rejuvenation has focused on targeted strategies for specific systems. Parri et al. applied fault injection and accelerated life testing techniques to assess the impact of soft errors on software aging in component-based web applications and implemented micro rejuvenation strategies by altering component configurations to slow down the aging process [24].

Studies specifically targeting Docker environments have investigated rejuvenation effects in containerized settings. Torquato and Vieira conducted experimental studies on applying the SWARE method within Dockerd daemons to provoke aging and assess the effects of rejuvenation strategies, finding that rebooting significantly improves system performance and stability [25]. Vinícius et al. conducted a detailed study on software aging issues on the Docker platform, using the SWARE method to detect aging signs and test rejuvenation strategy effects. Experimental results showed that LSTM machine learning algorithms offer higher prediction accuracy than traditional time series models, and strategies like system rebooting can significantly reduce resource consumption [3].

In summary, existing cloud-based rejuvenation methods are predominantly relying on system rebooting. Such strategies lack targeted measures for diverse aging states and may disrupt system stability due to excessive or untimely operations. To address these limitations, this paper employs the Double Q-Learning algorithm for rejuvenation decision-making, which balances immediate response needs and long-term sustainability, reduces system downtime, and ensures service continuity and quality. For performance comparison, we select Q-learning and SARSA, two representative reinforcement learning algorithms, as baseline methods, given their widespread application in adaptive decision scenarios.

C. *Research Limitations and Our Contribution*

Current studies on the Docker platform aging often focus on providing decision support by improving prediction accuracy, however, they tend to overlook the long-term characteristics of software aging data, which limits the precision of predictive models. Additionally, there are insufficient approaches on how to use predictive outcomes to implement effective rejuvenation measures, often resulting in a lack of diversity and intelligence in rejuvenation decision. Therefore, our work proposes a comprehensive framework designed for the Docker platform, TiAD-DQR. This framework enables a more complete process. Accurate long-term trend prediction informs robust state determination, and precise state judgments guide adaptive rejuvenation strategies. This integration achieves more coherent, targeted aging management.

III. Construction of the TiAD-DQR Framework

As shown in Fig. 1, our framework uses the TDDE-GMAD aging states awareness approach to accurately determine the long-term aging states of software, combined with the DQL reinforcement learning algorithm for intelligent rejuvenation decision. This approach significantly improves system performance and stability, while providing a more efficient and sustainable rejuvenation strategy for the Docker platform. To elaborate, we first introduce the three constituent modules separately, followed by a detailed description of the overall framework they form.

A. *Software Aging Trends Prediction Based on the TDDE Model*

First, the time series data on software aging is decomposed to preserve the trend component. Then, this component is standardized and inputted into the Time-series Dense Encoder (TiDE) model for prediction. Finally, the predicted data is denormalized to obtain the final prediction results. The entire construction process of the TDDE model can be summarized in the following three steps.

1) Trend Decomposition Process

In the context of software aging, the monotonicity of the data represents its overall trend, which is crucial for understanding the aging behavior of software [26]. However, the common seasonal components in the data, manifested as time-related repetitive patterns, may obscure the real trend. Therefore, to emphasize monotonicity and reduce the interference of seasonal fluctuations, this paper opts to remove the seasonal components. This process can be represented by the following (1).

$$\widehat{T}_t = \frac{1}{m} \sum_{j=-k}^{k} Y_{t+j} \tag{1}$$

where $\widehat{T}_t$ represents the estimate of the trend component at time point t, m is the total number of observations, k is the half-width of the sliding window, and Y_{t+j} is the original observation at time point $t + j$.

Fig. 2 shows the decomposition results of three key aging metrics, CPU utilization, memory usage, and network received, collected from the Docker platform. The data was sampled at 1-minute intervals, with each time step on the horizontal axis corresponding to 1 minute. These metrics were selected because CPU utilization directly reflects container workload and computational resource consumption, serving as a critical parameter for detecting computing resource leaks and performance degradation; memory usage monitors container memory allocation and release patterns, effectively identifying aging phenomena such as memory leaks; and network received indicates container network I/O load and service responsiveness, enabling timely detection of network resource anomalies.

Collectively, these metrics comprehensively characterize hardware resource states and can pinpoint leakage sources. As the subfigures illustrate, by decomposing and retaining the trend components, not only is the data's trendiness more pronounced, which helps improve the accuracy of prediction, but it also effectively reduces fluctuations in the data that are unrelated to aging. These fluctuations may originate from temporary factors such as excessive instantaneous loads, and the anomalies they cause are usually transient and recoverable, unlike the continuous performance degradation caused by software aging.

2) TiDE Model Construction Process

The TiDE model is a dedicated model for long-term time series forecasting, featuring an encoder-decoder structure. It combines the simplicity and efficiency of Multilayer Perceptrons (MLP) and is capable of handling complex nonlinear dependencies and covariate information in time series data, thereby enhancing the speed and accuracy of prediction [27].

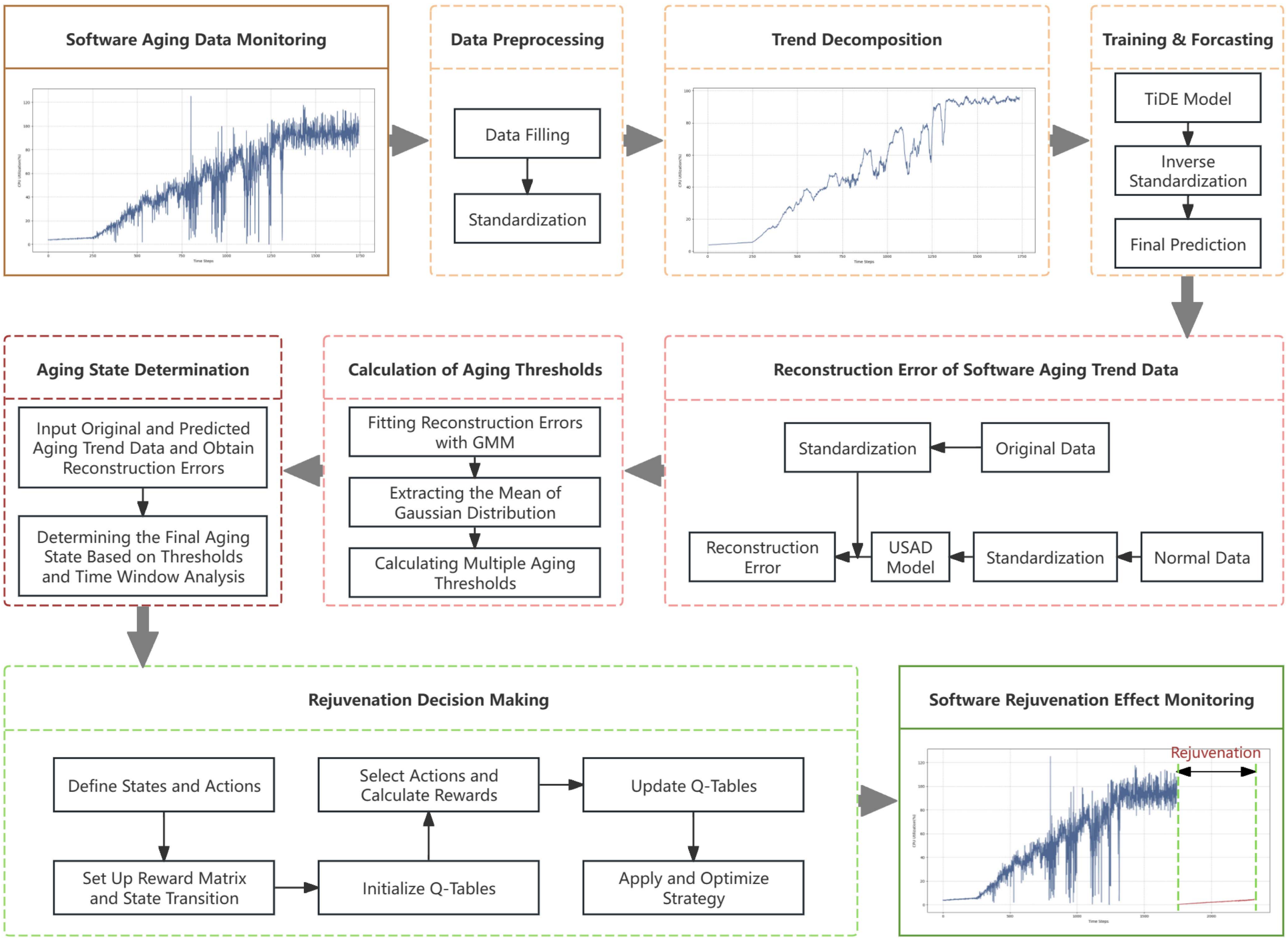

Software Aging Data Monitoring
Data Preprocessing
Data Filling
Standardization
Trend Decomposition
Training & Forcasting
TiDE Model
Inverse Standardization
Final Prediction
Aging State Determination
Input Original and Predicted Aging Trend Data and Obtain Reconstruction Errors
Determining the Final Aging State Based on Thresholds and Time Window Analysis
Calculation of Aging Thresholds
Fitting Reconstruction Errors with GMM
Extracting the Mean of Gaussian Distribution
Calculating Multiple Aging Thresholds
Reconstruction Error of Software Aging Trend Data
Standardization
Original Data
Reconstruction Error
USAD Model
Standardization
Normal Data
Rejuvenation Decision Making
Define States and Actions
Select Actions and Calculate Rewards
Update Q-Tables
Set Up Reward Matrix and State Transition
Initialize Q-Tables
Apply and Optimize Strategy
Software Rejuvenation Effect Monitoring
Rejuvenation

Fig. 1. Architecture of the TiAD-DQR framework.

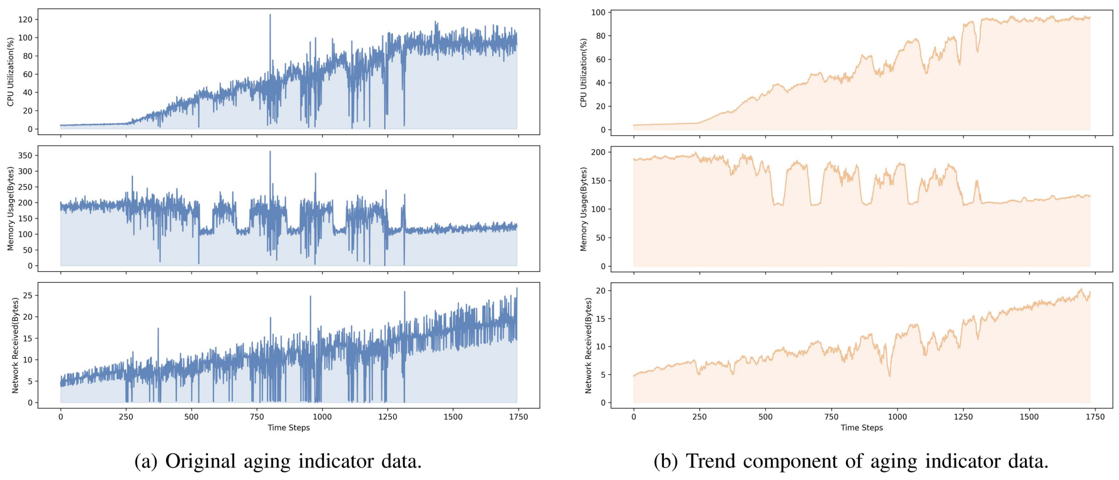


Fig. 2. Decomposition of time series aging indicator data.

The construction process of the TiDE model is shown in Fig. 3 and includes the following steps.

Feature Projection. This projects the dynamic covariates at each time point into a low-dimensional space, reducing dimensions and capturing key information.

Encoding Process. The dense encoder section uses a Residual Block to process historical information and dynamic covariates, mapping them into a high-dimensional space to capture the intrinsic features and dynamic relationships of the data.

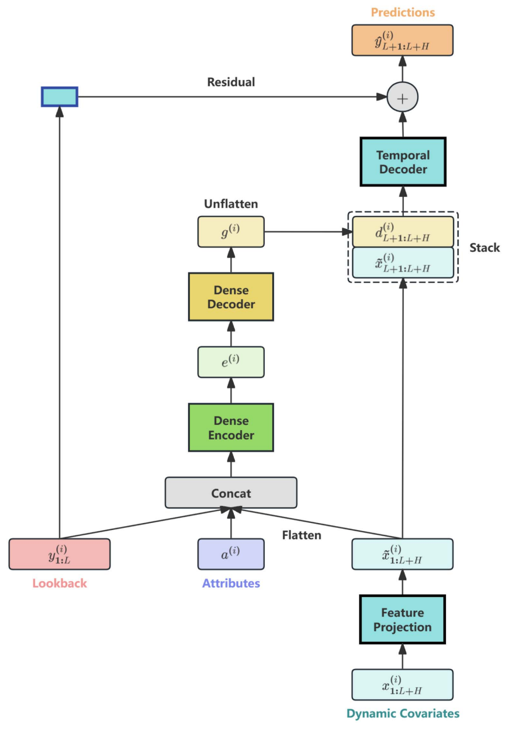


Fig. 3. Architecture of the TiDE model.

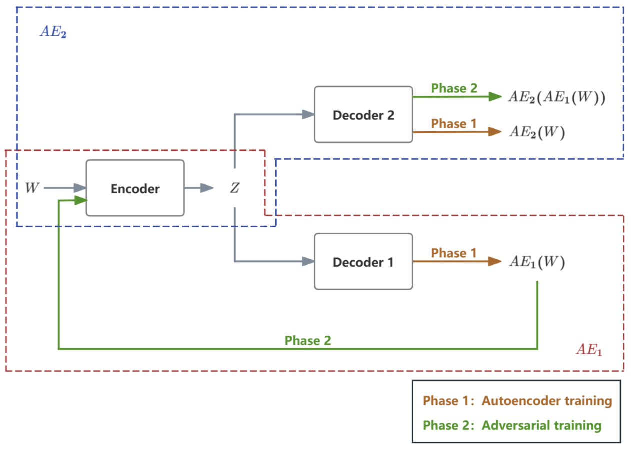


Fig. 4. Architecture of the USAD model.

Decoding Process. The dense decoder section uses a Residual Block to map the encoded features back to the original space, generating prediction for future time points, including a dense decoder and a temporal decoder.

3) Obtaining Final Prediction

As normalization is used during the model training process to improve training efficiency and model performance, the output of the model is also within this normalized range. Therefore, to make the prediction results meaningful to the end user, it is necessary to perform a denormalization operation so that the output has the same scale and distribution as the original data. This process can be represented by the following (2).

$$X_{original} = X_{std} \times \sigma + \mu \tag{2}$$

where $X_{original}$ is the denormalized prediction value within the range of the original data, X_{std} is the normalized model output, and σ and μ are the mean and standard deviation of the original data, respectively.

B. Software Aging States Determination Based on the GMAD Model

First, the UnSupervised Anomaly Detection (USAD) model is trained using data from the software running in its normal state, allowing the model to accurately reconstruct this data and amplify the reconstruction errors when faced with aging trend data. Then, by performing GMM clustering analysis on the reconstruction errors output by the USAD model, thresholds for different software aging states are automatically determined, differentiating the various aging states. Finally, to enhance the accuracy of aging states determination and reduce the impact of noise, a time window analysis is employed, thus more accurately assessing the software's aging condition. The entire construction process of the GMAD model can be summarized in the following three steps.

1) USAD Model Construction Process

The USAD model combines the advantages of autoencoders and adversarial training. It consists of an encoder network E and two decoder networks D_1 and D_2, which together form two autoencoders, AE_1 and AE_2, sharing the same encoder E. The model enhances sensitivity to anomalous data by learning the feature distribution of normal data, where the autoencoder structure captures long-term dependencies and dynamic changes in time-series data. Adversarial training amplifies reconstruction errors when encountering anomalous data, thus boosting the model's ability to recognize anomalies [28]. GMAD leverages this characteristic to determine aging states based on the magnitude of reconstruction errors. Crucially, the TDDE's prior trend decomposition removes peak fluctuations, allowing USAD to analyze purified aging trends. This pipeline design enhances robustness and reduces misjudgment risks. The construction process of the USAD model is illustrated in Fig. 4 and primarily includes the following steps.

Autoencoder Training. Two similar autoencoders, AE_1 and AE_2, are trained using a normal data set to minimize the reconstruction error between the input and output.

Adversarial Training. After training the autoencoders, adversarial training methods are employed to further optimize the model. Here, AE_2 is trained to differentiate between real data and data reconstructed by AE_1, while AE_1 is trained to deceive AE_2 by generating reconstructions indistinguishable from real data by AE_2.

Anomaly Detection. The USAD model uses reconstruction errors to detect anomalies. An increase in reconstruction error indicates that the input data deviates from the feature distribution of normal data learned by the model, thereby signaling potential anomalies.

2) Aging Thresholds Determination Based on GMM Clustering

To accurately identify different aging states from the reconstruction errors output by the USAD model, this paper employs Gaussian Mixture Model (GMM) clustering methods to automatically determine the thresholds for aging states, thus effectively recognizing different software aging states. The application of the GMM method primarily includes the following steps.

Model Fitting. The reconstruction error data is fitted using GMM by maximizing the log-likelihood function, which can be represented by the following (3).

$$l(\theta) = \sum_{i=1}^{N} \log \left(\sum_{k=1}^{4} \pi_k \mathcal{N} \left(x^{(i)}; \mu_k, \sum k \right) \right) \tag{3}$$

where θ represents the model parameters, x is the reconstruction error data, N is the total number of data points, π_k is the mixture weight of the k-th Gaussian distribution, and $\mathcal{N}(x^{(i)}; \mu_k, \sum k)$ is the probability density function of the Gaussian distribution with mean μ_k and covariance matrix $\sum k$.

Thresholds Calculation. The means of four Gaussian distributions from the GMM model are extracted and sorted in ascending order to determine the central positions of different aging states. Then three thresholds are calculated based on these sorted means, each threshold being the midpoint of adjacent means. These thresholds divide the reconstruction error space into four distinct intervals, each corresponding to a specific aging state.This process can be represented by the following (4).

$$T_i = \frac{\mu_i + \mu_{i+1}}{2} \tag{4}$$

where T_i is the threshold for the i-th aging state.

Aging States Classification. Based on the calculated thresholds, the reconstruction error data is categorized, and each data point is assigned an aging state label, including healthy, aging, severely aging, and failed states.

The core of this process is its ability to focus on abnormal patterns, enhancing the capability for anomaly detection while significantly reducing the impact of random noise. Specifically, GMM clustering focuses on analyzing atypical patterns in the reconstruction errors, which are key indicators of aging states, thus enabling precise capture of abnormal software behaviors. Additionally, due to the unique design of the USAD model, which amplifies anomalous patterns through autoencoders and adversarial training, the application of GMM clustering on these amplified errors further enhances the detection capabilities for subtle signs of aging.

3) Obtaining Final Aging States Based on Time Windows

To enhance the stability of aging states determination and reduce the impact of random prediction errors on the final results, this paper adopts a time-window-based approach to analyze the aging states of software. By considering the aging states over a period rather than at a single time point, this time-window-based method can reduce the effects caused by instantaneous errors in model prediction or random fluctuations in the data. This method offers a more stable and reliable determination of aging states, helping to avoid overinterpretation or misjudgment of software aging trends. This process can be represented by the following (5).

$$Label(i) = \left\lfloor \frac{1}{window} \sum_{j=i}^{i+window-1} y(j) \right\rfloor \tag{5}$$

where $Label(i)$ is the final aging state of the i-th window, $\lfloor \cdot \rfloor$ denotes rounding down, and $y(j)$ is the aging state label of the j-th time point.

Algorithm 1. Rejuvenation Decision Generation.

```
 1: **Input:** States $S$, Actions $A$, Reward matrix $R$, Learning
    rate $\alpha$, discount factor $\gamma$, exploration rate $\epsilon$, Max
    episodes $N$, max steps $T$
 2: **Output:** Trained Q-tables $Q_1$ and $Q_2$
 3: **Initialize:** $Q_1(s,a), Q_2(s,a), \forall s \in S, a \in A$
 4: **for** episode $= 1$ to $N$ **do**
 5:   Initialize $s$ randomly
 6:   **for** step $= 1$ to $T$ **do**
 7:     Choose action $a$ and calculate rewards
 8:     $a \leftarrow \epsilon$-greedy$(Q_1 + Q_2, s)$
 9:     Execute the action $T$, obtain new state $s'$
10:     $(s', a) \leftarrow$ execute$(a)$
11:     Update Q-table
12:     **if** $Q_{\text{update}} = Q_1$ **then**
13:       $Q_1(s,a) \leftarrow Q_1(s,a) + \alpha\Delta(r, Q_2, s')$
14:     **else**
15:       $Q_2(s,a) \leftarrow Q_2(s,a) + \alpha\Delta(r, Q_1, s')$
16:     **end if**
17:     $s \leftarrow s'$
18:     **if** $s$ is terminal **then break**
19:     **end if**
20:   **end for**
21: **end for**
22: **return** $Q_1, Q_2$
```

C. Software Rejuvenation Decision Generation Based on the DQL Algorithm

Software rejuvenation, as an effective preventative measure against software aging, is crucial for ensuring the stable operation of software systems. However, deciding when and how to implement software rejuvenation is a complex decision generation process. It requires consideration of the system's current state as well as the long-term benefits of rejuvenation actions. Therefore, this paper employs the DQL algorithm to enable more intelligent and diversified software rejuvenation decision. The Algorithm 1 for rejuvenation decision generation based on DQL is as follows.

Define states and actions. States should include the aging state of the current time window and the predicted aging state of the next time window. These can be defined as a tuple (Current State, Future State), where each state can be either ”healthy”, ”aging”, ”severely aging”, or ”failed”. The four actions for rejuvenation are ”take no action”, ”restart container”, ”migrate tasks”, or ”reboot operating system”.

Define the reward matrix and state transition function. Set up the reward matrix, which defines the rewards obtained under given states and actions. And define the state transition function, which specifies the new state to which the system will transition after executing a specific action from the current state.

Initialize Q-tables. Initialize two Q-tables, Q^A and Q^B. For all state-action pairs (s, a), set $Q^A(s, a)$ and $Q^B(s, a)$ to zero.

Choose action and calculate rewards. For a given state s, use an ϵ-greedy strategy based on the average of the current Q^A and Q^B tables to choose an action a, balancing exploration and exploitation. After executing the chosen action a, calculate the immediate reward r and the new state s' based on the reward matrix and state transition function.

Update Q-table. Randomly select one Q-table for updating. For the selected Q-table, apply the expectation maximization update rule, updating the Q-value based on the immediate reward received and the discounted future reward. The unselected Q-table is used to provide the value of the optimal action for the next state, helping to reduce overestimation issues. When using Q^A to select action a^*, the update for the Q^A table is represented by the following (6).

$$Q^A(s,a) \leftarrow Q^A(s,a) + \alpha \left[r + \gamma Q^B(s', \underset{a'}{\operatorname{argmax}} Q^A(s', a')) - Q^A(s,a) \right] \tag{6}$$

where s is the current state, a is the current action, r is the reward, α is the learning rate, γ is the discount factor, s' is the next state, and a' is the optimal action chosen in state s' based on Q^A. When selecting actions using Q^B, the update rule for the Q^B table is similar to that of Q^A, but Q^A is used to compute the maximum action value for the next state.

Loop iteration. Repeat the above steps for multiple iterations, adjusting the strategy with each iteration based on environmental feedback until the termination conditions of reaching the maximum number of iterations or Q-value convergence are met. Monitor and record the Q-value changes at each step to assess the effectiveness of the learning process and strategy.

Strategy application. Use the average values of the learned Q^A and Q^B tables for generating rejuvenation decision to optimize system performance and reduce downtime.

This method dynamically selects optimal rejuvenation strategies by aligning action intensity with the system's current and future aging severity. It promotes cost-effective actions such as container restarts for mild aging and penalizes mismatched decisions like unnecessary reboots. This balanced approach minimizes downtime, prevents over-intervention, and enhances long-term system adaptability and intelligence.

D. Integration of the TiAD-DQR Framework

In summary, the TiAD-DQR framework systematically addresses software aging in Docker platforms through a three stage pipeline shown in Fig. 1. First, the TDDE model achieves high precision aging trend prediction by trend decomposition and long-term feature extraction, laying a foundation for subsequent state determination by clarifying the underlying patterns of aging evolution. Second, the GMAD model enables robust aging state determination through automated threshold partitioning and smoothed time-window analysis, leveraging the decomposed trends from TDDE to more accurately distinguish between different aging states and reduce misclassification. Finally, the DQL algorithm generates dynamic rejuvenation strategies by jointly optimizing immediate responsiveness and long-term sustainability, relying on GMAD's precise state judgments to ensure that decisions are targeted and avoid unnecessary interventions. This modular design, where each stage enhances the next, establishing a closed-loop workflow of prediction, determination, and decision that significantly enhances system availability and service stability. Subsequent experiments validate each component's individual efficacy and their synergistic integration.

IV. Experimental Setup

In this section, we introduce the research questions addressed in this paper and detail the datasets utilized and the evaluation metrics employed. All experiments were run on an NVIDIA GTX 1650 Super with 4 GB memory, using TensorFlow for prediction, PyTorch for state determination, and standard Python libraries for rejuvenation.

A. Research Questions

The overall performance of our TiAD-DQR framework is verified by conducting comprehensive experiments. In particular, we aim to answer the following four research questions (RQs).

- *RQ1:* Compared to other time-series prediction methods, can the TDDE model improve the accuracy of aging trends prediction on the Docker platform by reducing non-aging related fluctuations and extracting and leveraging long-term features?
- *RQ2:* Relative to traditional methods, can the GMAD model enhance the accuracy of aging states identification on the Docker platform by automating the determination of aging thresholds?
- *RQ3:* How does the Double Q-Learning algorithm balance immediate response and long-term sustainability when generating rejuvenation decision? How does it compare in terms of stability and effectiveness with other reinforcement learning methods?
- *RQ4:* What is the overall impact on system performance and stability in Docker environments following the implementation of the TiAD-DQR framework?

B. Experimental Environment and Dataset Description

Due to the scarcity of public datasets for Docker platform aging studies, we constructed an experimental dataset using a cloud-edge testbed comprising one cloud node and two edge nodes. The cloud node was equipped with 8 vCPUs and 4 GB RAM, while each edge node had 4 vCPUs and 2 GB RAM. All nodes ran Docker 20.10.12 with Cgroups v2 on CentOS 7. To accelerate aging observation, we adopted the cyclic container operation method from Vinicius et al. [3], performing repeated container creation and deletion under controlled conditions.

The test involved 50 httpd containers, a number adapted to our hardware configuration, with operations executed at 10-second intervals. Each container was constrained to 512 MB memory and 0.5 CPU cores. System metrics including CPU utilization, memory usage, and network received were collected at one-minute intervals using cAdvisor and Prometheus, yielding 1,745 high-resolution records. The complete dataset is publicly available at: https://github.com/DockerSADataset/-DockerSoftwareAging-CloudEdge-Dataset -. The dataset's distinctive contribution lies in its controlled simulation of aging within an authentic cloud-edge environment, with validation through post-load analysis that effectively separates persistent aging effects from transient workload fluctuations.

C. Evaluation Metrics

In order to comprehensively assess the effectiveness of the TiAD-DQR framework and its constituent modules proposed in this study, we employed multiple evaluation metrics to separately evaluate the predictive accuracy, judgment results, and performance of the decision generation strategies according to different experimental goals.

1) Metrics for Aging Trends Prediction: Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) are used to assess the performance of aging trends prediction, as shown in (7) and (8).

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{7}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{8}$$

where y_i is the actual observed value, $\hat{y}_i$ is the predicted value, and n is the total number of observations.

2) Metrics for Aging States Determination: Precision (P), Recall (R), F1 Score (F1), and F1* Score (F1*) [29] are used to assess the performance of aging states recognition, as shown in (9) to (12).

$$P = \frac{TP}{TP + FP} \tag{9}$$

$$R = \frac{TP}{TP + FN} \tag{10}$$

$$F1 = 2 \cdot \frac{P \cdot R}{P + R} \tag{11}$$

$$F1^* = 2 \cdot \frac{\overline{P} \cdot \overline{R}}{\overline{P} + \overline{R}} \tag{12}$$

where TP is true positive, FP is false positive, FN is false negative, while $\overline{P}$ and $\overline{R}$ stand for average precision and average recall respectively.

3) Metrics for Rejuvenation Decision Generation Algorithm: In assessing rejuvenation decision methods based on the DQL algorithm, we focus particularly on the average maximum Q-value change, a metric used to measure the stability of Q-value updates in Q-learning algorithms, as shown in (13).

$$Max\ \ Q = \frac{1}{m}\sum_{k=1}^{m} max_{i,j}\,|Q_{k+1}(s_i, a_j) - Q_k(s_i, a_j)| \tag{13}$$

where $Q_k(s_i, a_j)$ represents the Q value for action a_j taken in state s_i during the k-th iteration, m is the total number of iterations, and $max_{i,j}$ represents the maximum change in all Q values during the k-th iteration.

V. Experimental Results

In this section, we systematically present the experimental results and their analysis for each component of the TiAD-DQR framework. First, we demonstrate the performance of the TDDE aging trends prediction module. Next, the effectiveness of the GMAD aging states determination module is analyzed. Then, we detail the implementation results of the DQL rejuvenation decision generation algorithm. Finally, the overall performance of the TiAD-DQR framework is discussed. Each part is supported by detailed data analysis and charts to verify the effectiveness and reliability of the framework in practical applications.

A. Results of the Aging Trends Prediction: RQ1

1) Baseline Models: To assess performance, this study compares the proposed TDDE model with various benchmark models, including two simple linear and nonlinear models: DLinear and NLinear [30]; four advanced deep learning-based models: FEDformer, Autoformer, Informer, and Transformer; and two widely used time series forecasting models: SARIMA and LSTM.

In the training of all models, MSE is uniformly used as the loss function, with a historical length set at 120 time steps, and the prediction steps of the models set at 15, 30, 60, and 120 respectively. All experiments are conducted on datasets that have been standardized and normalized, divided into training, validation, and test sets, with proportions of 70%, 10%, and 20% respectively. The above settings aim to ensure fairness and scientific validity in the comparisons between the models.

2) Results: The performance of all methods on the RMSE and MAE metrics is presented in Table I, with the best results highlighted in bold. The results indicate that the TDDE model significantly outperforms other baseline models on both evaluation metrics, demonstrating higher prediction accuracy. For instance, with a prediction step of 15, the TDDE model reduced the RMSE by 72.40% and 75.27% compared to the simple models NLinear and DLinear, respectively. Compared to advanced models such as FEDformer, Autoformer, Informer, and Transformer, the reductions are 69.51%, 70.44%, 88.59%, and 88.10%, respectively. Compared to traditional models SARIMA and LSTM, the reductions are 96.96% and 97.95%, respectively. Notably, these improvements become even more pronounced in long-term predictions: at 120-step horizon, TDDE outperforms FEDformer by 34.43%, Autoformer by 37.70%, and LSTM by 98.16%. This significant advantage in long-horizon forecasting directly stems from TDDE's ability to extract and leverage long-term aging features through trend decomposition, while

TABLE I
COMPARISON OF TDDE MODEL WITH OTHER MODELS

Models		TDDE		NLinear		DLinear		FEDformer		Autoformer		Informer		Transformer		SARIMA		LSTM	
		RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE
Software Aging Data	15	**0.1483**	**0.1013**	0.5375	0.3695	0.5997	0.4267	0.4864	0.3283	0.5017	0.3543	1.2995	1.1285	1.2463	1.0545	4.8757	3.4080	7.2351	5.4360
	30	**0.1977**	**0.1330**	0.5429	0.3727	0.6296	0.4526	0.4892	0.3358	0.5089	0.3629	1.5702	1.3877	1.4899	1.2005	5.6415	3.8977	8.8287	6.5536
	60	**0.2864**	**0.1955**	0.5578	0.3837	0.6747	0.4878	0.5197	0.3670	0.5476	0.3922	1.5884	1.4183	1.5728	1.3725	7.8354	5.3421	9.9588	7.1434
	120	**0.3474**	**0.2581**	0.5831	0.3994	0.7143	0.5138	0.5299	0.3777	0.5576	0.4117	1.8425	1.5948	1.6397	1.4363	8.7764	6.1605	18.8933	13.3826

TABLE II
COMPARISON OF TDDE AND TIDE MODELS IN ABLATION STUDY

Input-120	TDDE		TiDE	
Predict	RMSE	MAE	RMSE	MAE
15	**0.1483**	**0.1013**	0.4948	0.3309
30	**0.1977**	**0.1330**	0.5017	0.3381
60	**0.2864**	**0.1955**	0.5123	0.3492
120	**0.3474**	**0.2581**	0.5307	0.3710

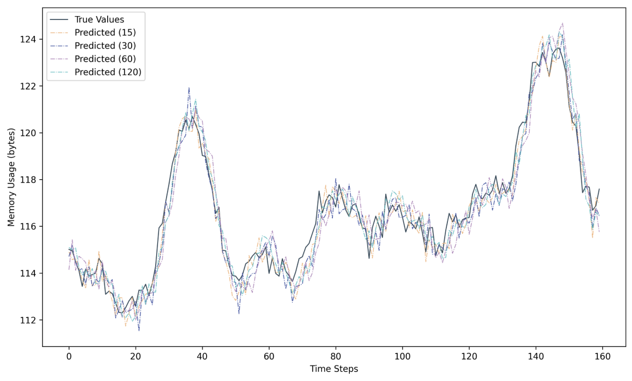


Fig. 5. Comparison of true and predicted memory usage.

other models fail to decouple persistent aging trends from short-term noise, leading to error accumulation over time. These data highlight the TDDE model's exceptional performance in predicting aging trends on the Docker platform.

3) Ablation Study: As shown in Table II, the TDDE model exhibits significant improvements in the RMSE and MAE metrics compared to the TiDE model. Additionally, a performance comparison of the TDDE model at different prediction steps on the aging metric of memory usage is shown in Fig. 5, showing that the model fits the aging trends well across various prediction lengths. These results further confirm that by decomposing time-series data to extract long-term aging features and suppress non-aging fluctuations, the TDDE model significantly enhances the accuracy of long-term aging trend prediction on the Docker platform.

B. Results of the Aging States Determination: RQ2

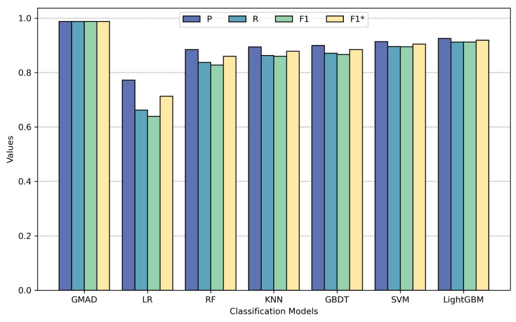
1) Baseline Models: To assess performance, this study compared the proposed GMAD model with various traditional benchmark models, including supervised classification models: Logistic Regression (LR), Random Forest (RF), K-Nearest Neighbors (KNN), Gradient Boosting Decision Trees (GBDT), Support Vector Machine (SVM), and Light Gradient Boosting Machine (LightGBM) [31]; as well as unsupervised clustering models: Hierarchical Clustering (HC), KMeans, and Density-Based Spatial Clustering of Applications with Noise (DBSCAN).

All experiments were conducted on a dataset that was standardized and normalized, divided into training, validation, and test sets with proportions of 70%, 10%, and 20%, respectively. Aging states were categorized into four types: healthy, aging, severely aging, and failed states. All models utilized the same test set, but due to methodological differences, the GMAD model used only non-aging data during training and validation. Additionally, the clustering models automatically determined the aging thresholds by the midpoints between two clusters to assess the aging states. The above settings aim to ensure fairness and scientific rigor in the comparisons between the models.

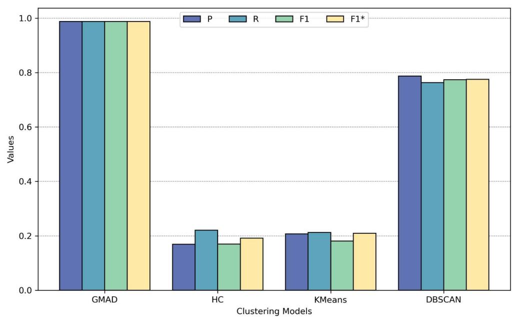
2) Results: The performance of all methods across the P, R, F1, and F1* metrics is shown in Table III, marking the best results in bold. The results show that the GMAD model significantly surpasses other baseline models in the P, R, F1, and F1* evaluation metrics, exhibiting higher judgment accuracy. For example, in terms of the F1 metric, GMAD outperforms traditional classification models such as LR, RF, KNN, GBDT, SVM, and LightGBM with increases of 54.57%, 19.31%, 14.83%, 13.95%, 10.36%, and 8.29% respectively. Relative to traditional clustering models such as HC, KMeans, and DBSCAN, the increases are 480.48%, 445.22%, and 27.59% respectively. These figures underscore the outstanding performance of the GMAD model in determining the aging states on the Docker platform.

TABLE III
COMPARISON OF GMAD MODEL WITH OTHER MODELS

Models		GMAD	LR	RF	KNN	GBDT	SVM	LightGBM	HC	KMeans	DBSCAN
Software Aging Data	P	**0.9877**	0.7724	0.8841	0.8942	0.8991	0.9138	0.9257	0.1693	0.2070	0.7871
	R	**0.9874**	0.6625	0.8375	0.8625	0.8708	0.8958	0.9125	0.2208	0.2125	0.7633
	F1	**0.9874**	0.6388	0.8276	0.8599	0.8665	0.8947	0.9118	0.1701	0.1811	0.7739
	F1*	**0.9875**	0.7133	0.8601	0.8781	0.8847	0.9047	0.9191	0.1917	0.2097	0.7750



(a) Comparison of GMAD and traditional classification models.



(b) Comparison of GMAD and traditional clustering models.

Fig. 6. Performance comparison of GMAD with traditional models.

TABLE IV
COMPARISON OF GMAD AND GMM MODELS IN ABLATION STUDY

	P	R	F1	F1*
GMAD (window)	**0.9877**	**0.9874**	**0.9874**	**0.9875**
GMAD (non-window)	0.9589	0.9583	0.9579	0.9586
GMM	0.2563	0.2208	0.1732	0.2372

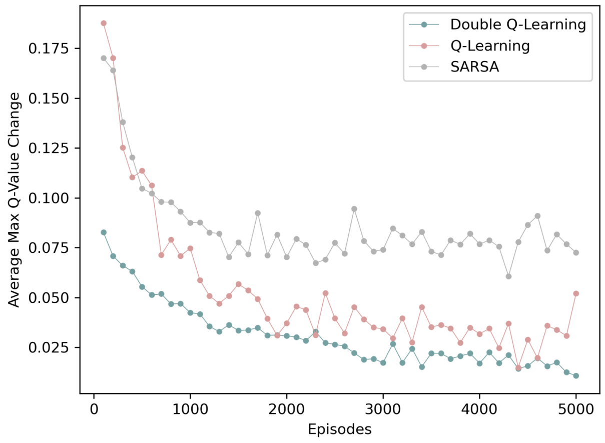
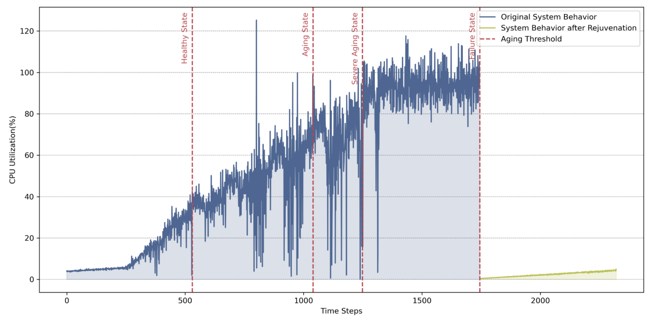


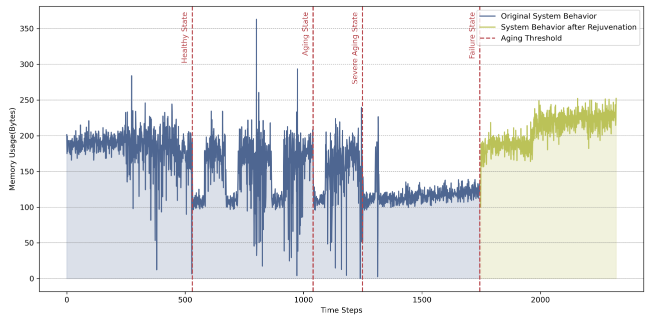
Fig. 7. Evolution of average maximum Q-values across learning episodes.

Moreover, the notable performance disparities between the GMAD model and traditional classification and clustering models is illustrated in Fig. 6. Traditional classification models are observed to have high performance. However, they depend on supervised learning and need manually labeled data, which could be restrictive in real-world applications. Conversely, traditional clustering methods, while providing some automation, often fall short in effectiveness evaluation.

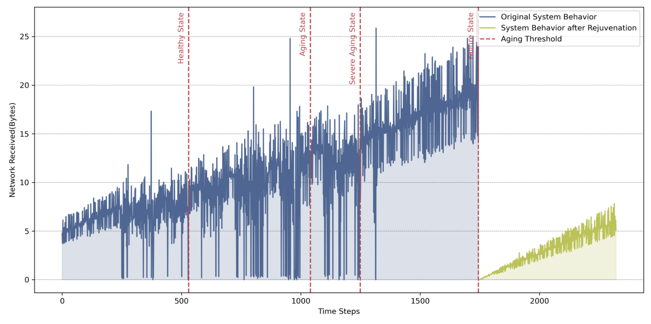
3) Ablation Study: As shown in Table IV, we illustrates that the GMAD model significantly enhances performance on the P, R, F1, and F1* metrics over the standalone GMM model. This outcome validates the strengths of the GMAD model in concentrating on aging data patterns and amplifying aging detection capabilities. It not only automatically sets aging thresholds but also precisely determines the aging states of the Docker platform, addressing the constraints of manually set thresholds in conventional methods. Moreover, compared to the non-window GMAD, the GMAD shows a modest improvement in performance, suggesting that time-window-based analysis can effectively mitigate the adverse effects of random errors, thus increasing the accuracy of aging states assessments.



(a) CPU utilization recovery efficiency.



(b) Memory usage recovery efficiency.



(c) Network received recovery efficiency.

Fig. 8. Recovery efficiency of system indicators post TiAD-DQR framework implementation.

C. *Results of the Rejuvenation Decision Generation: RQ3*

1) Baseline Models: For performance evaluation, the proposed DQL algorithm was compared with two other reinforcement learning algorithms, Q-Learning and SARSA. All algorithms were implemented under identical experimental conditions to guarantee the consistency and fairness of the comparisons.

During the implementation, the same reward matrix and state transition functions were used, actions were selected using an ϵ-greedy strategy, and parameters like learning rate, discount factor, and exploration rate were kept consistent. States are defined as S_0 (healthy), S_1 (aging), S_2 (severely aging), and S_3 (failed). Actions are A_0 (take no action), A_1 (restart container), A_2 (migrate tasks), and A_3 (reboot operating system). These actions represent varying levels of rejuvenation measures. Higher-level actions, while more effective in recovery, can result in longer system downtimes.

TABLE V
FINAL Q-TABLE AND OPTIMAL ACTIONS DERIVED FROM DQL ALGORITHM

State Pair	A_0	A_1	A_2	A_3	Best Action
(S_0, S_0)	0.713599133	-0.004476852	-0.034372211	-0.034298981	A_0
(S_0, S_1)	0.631029606	0.086861706	0.005814353	-0.016431521	A_0
(S_0, S_2)	-0.016173856	-0.004038617	0.236748963	0.00473738	A_2
(S_0, S_3)	-0.010828016	-0.011014834	-0.002784512	0.141064652	A_3
(S_1, S_0)	5	4.25	3.5	3	A_0
(S_1, S_1)	0.008185558	0.905768508	0.000862352	0.0349125	A_1
(S_1, S_2)	-0.009213924	0.00127968	0.675972497	0.02125	A_2
(S_1, S_3)	-0.006197095	-0.003572616	-0.000426503	0.543950945	A_3
(S_2, S_0)	5	4.5	3.5	3	A_0
(S_2, S_1)	1.0350627	0.793909932	0.028574502	0.44722185	A_0
(S_2, S_2)	-0.009649575	0.141678623	0.049875	0.046687734	A_1
(S_2, S_3)	-0.006045291	-0.005551523	0.000136429	0.643817271	A_3
(S_3, S_0)	7.5	6.75	6.5	6.25	A_0
(S_3, S_1)	0.017069775	1.098972272	0.015259877	0.046796875	A_1
(S_3, S_2)	-0.004917269	0.001155388	0.034909422	0.871517882	A_3
(S_3, S_3)	-0.069274363	-0.02174276	-0.006796199	1.28570057	A_3

2) Results: As shown in Table V, the Q-table ultimately derived from the DQL algorithm is displayed, representing the average of two separate Q-tables throughout the training phase. As the algorithm undergoes continuous cyclic iterations, the Q-table within the model approaches convergence, demonstrating that the algorithm integrates considerations of both current and future aging states to select the optimal strategy from multiple rejuvenation actions. For instance, in the state (S_0, S_1), the Q-table suggests the optimal action as A_0, a choice that reflects the strategy of maintaining system stability during initial aging stages, effectively balancing immediate responsiveness with long-term sustainability. Conversely, the optimal action A_1 under the state (S_2, S_2) is unexpected, potentially reflecting the effects of randomness during training or specific settings of the reward mechanism.

Furthermore, we illustrates the comparison of the three algorithms in Fig. 7 based on the trends in average maximum Q-values. Notably, the DQL algorithm converges towards the optimal value more quickly than the Q-Learning and SARSA algorithms, demonstrating its pronounced advantage in learning stability. This stability allows DQL to rapidly approximate the optimal strategy while minimizing fluctuations during training, thus enhancing reliability when designing long-term effective and adaptable rejuvenation strategies. This characteristic is critical in devising rejuvenation strategies, ensuring the efficiency and effectiveness of rejuvenation decision, aiding the system in maintaining high performance and stability amidst continuous aging challenges.

D. *Results of the TiAD-DQR Framework Implementing: RQ4*

We present a comparison of the duration in failed state between the TiAD-DQR framework and traditional reboot

TABLE VI
COMPARISON OF FAILED STATE DURATION BETWEEN TIAD-DQR FRAMEWORK AND TRADITIONAL RESTART STRATEGIES

Strategies	TiAD-DQR Framework	Threshold-Based Restart	Periodic Restart
Failed State Duration	**0s**	38m	3h11m

strategies in Table VI, including a periodic reboot strategy on a 24-hour cycle and a threshold reboot strategy triggered immediately upon the system's CPU utilization reaching 100%. The results distinctly demonstrate that the TiAD-DQR framework significantly excels in preventing the Docker platform from entering failed state, thereby effectively safeguarding the system's availability and stability.

Furthermore, we extensively illustrate three key performance metrics of the Docker platform under extreme conditions in Fig. 8, i.e., CPU utilization, memory usage, and network received. In the diagrams, the blue line indicates the trajectory of the system's original data, and the green line displays the changes in data following the implementation of rejuvenation strategies. The figures reveal that the system underwent four phases: S_0, S_1, S_2, and S_3, ultimately stabilizing in a S_3 state. By implementing the TiAD-DQR framework, which entails using the TDDE-GMAD model to predict and determine aging states and following rejuvenation measures guided by the Q-table generated by the DQL algorithm, system performance was significantly restored to its optimal state. Notably, the memory usage in Fig. 8 demonstrates successful rejuvenation through its return to healthy state operation characteristics, starting with the creation/deletion of 50 httpd containers every 10 seconds. This further validates the capability of the TiAD-DQR framework to sustain Docker platform performance and operational efficiency in extreme conditions. In addition, the time cost of our model training is at the minute level, and the inference cost is at the millisecond level, which can meet the actual application.

VI. Conclusion

This paper addresses the prevalent issue of software aging on the Docker platform by introducing the TiAD-DQR framework, which integrates advanced technologies to predict, assess, and alleviate aging impacts, thereby improving the Docker platform's stability and availability. Its core strengths lie in three integrated modules: the TDDE model enables long-term precise prediction of aging trends to guide rejuvenation decisions; the GMAD model automates accurate aging state determination, overcoming reliance on human experience; and the Double Q-Learning algorithm optimizes rejuvenation decisions by balancing current and future states. Experimental results validate that the TiAD-DQR framework effectively restores system performance to optimal levels, significantly enhancing operational efficiency and service quality.

The current research uses specific experimental scenarios to collect data for studying Docker software aging, and these controlled simulation scenarios effectively support our exploration of core aging trends while providing a reliable foundation for the research. In future work, we plan to extend our research in two key directions. First, we will deploy the experimental environment across multiple edge nodes to collect data under more complex cloud-edge scenarios. Second, we will introduce greater diversity and intensity in workload patterns to build a more comprehensive dataset. These efforts will help develop a more universally applicable framework for detecting Docker software aging across diverse operational contexts.

References

[1] M. Grottke, R. Matias, and K. S. Trivedi, "The fundamentals of software aging," in *Proc. IEEE Int. Conf. Softw. Rel. Eng. Workshops*, 2008, pp. 1–6.
[2] F. Oliveira, J. Araujo, R. Matos, L. Lins, A. Rodrigues, and P. Maciel, "Experimental evaluation of software aging effects in a container-based virtualization platform," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, 2020, pp. 414–419.
[3] L. Vinićius, L. Rodrigues, M. Torquato, and F. A. Silva, "Docker platform aging: A systematic performance evaluation and prediction of resource consumption," *J. Supercomputing*, vol. 78, no. 10, pp. 12898–12928, 2022.
[4] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, "Software rejuvenation: Analysis, module and applications," in *Proc. 25th Int. Symp. Fault-Tolerant Comput.. Dig. Papers*, 1995, pp. 381–390.
[5] H. Hasselt, "Double Q-learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2010, pp. 2613–2621.
[6] M. Grottke, L. Li, K. Vaidyanathan, and K. S. Trivedi, "Analysis of software aging in a web server," *IEEE Trans. Rel.*, vol. 55, no. 3, pp. 411–420, Sep. 2006.
[7] J. P. Magalhães and L. M. Silva, "Prediction of performance anomalies in web-applications based-on software aging scenarios," in *Proc. IEEE 2nd Int. Workshop Softw. Aging Rejuvenation*, 2010, pp. 1–7.
[8] S. Jia, C. Hou, and J. Wang, "Software aging analysis and prediction in a web server based on multiple linear regression algorithm," in *Proc. IEEE 9th Int. Conf. Commun. Softw. Netw.*, 2017, pp. 1452–1456.
[9] Y. Yan, P. Guo, and L. Liu, "A practice of forecasting software aging in an IIS web server using SVM," in *Proc. IEEE Int. Symp. Softw. Rel. Eng. Workshops*, 2014, pp. 443–448.
[10] Y. Yan, P. Guo, and L. Liu, "A novel hybridization of artificial neural networks and ARIMA models for forecasting resource consumption in an IIS web server," in *Proc. IEEE Int. Symp. Softw. Rel. Eng. Workshops*, 2014, pp. 437–442.
[11] Y. Yan and P. Guo, "A practice guide of software aging prediction in a web server based on machine learning," *China Commun.*, vol. 13, no. 6, pp. 225–235, 2016.
[12] P. Chen, Y. Qi, X. Li, D. Hou, and M. R.-T. Lyu, "ARF-predictor: Effective prediction of aging-related failure using entropy," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 4, pp. 675–693, Jul./Aug. 2018.
[13] C. Sudhakar, I. Shah, and T. Ramesh, "Software rejuvenation in cloud systems using neural networks," in *Proc. Int. Conf. Parallel, Distrib. Grid Comput.*, 2014, pp. 230–233.
[14] E. Andrade, F. Machida, R. Pietrantuono, and D. Cotroneo, "Software aging in image classification systems on cloud and edge," in *Proc. IEEE Int. Symp. Softw. Rel. Eng. Workshops*, 2020, pp. 342–348.
[15] F. Oliveira, J. Araujo, R. Matos, and P. Maciel, "Software aging in container-based virtualization: An experimental analysis on docker platform," in *Proc. 16th Iberian Conf. Inf. Syst. Technol.*, 2021, pp. 1–7.
[16] Y. Xie et al., "Real-time prediction of Docker container resource load based on a hybrid model of ARIMA and triple exponential smoothing," *IEEE Trans. Cloud Comput.*, vol. 10, no. 2, pp. 1386–1401, Second Quarter, 2022.
[17] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin, "FedFormer: Frequency enhanced decomposed transformer for long-term series forecasting," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 27268–27286.
[18] H. Wu, J. Xu, J. Wang, and M. Long, "AutoFormer: Decomposition transformers with auto-correlation for long-term series forecasting," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, pp. 22419–22430, 2021.
[19] H. Zhou et al., "Informer: Beyond efficient transformer for long sequence time-series forecasting," in *Proc. AAAI Conf. Artif. Intell.*, 2021, pp. 11106–11115.
[20] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.

[21] L. M. Silva, J. Alonso, and J. Torres, “Using virtualization to improve software rejuvenation,” *IEEE Trans. Comput.*, vol. 58, no. 11, pp. 1525–1538, Nov. 2009.
[22] S. Fakhrolmobasheri, E. Ataie, and A. Movaghar, “Modeling and evaluation of power-aware software rejuvenation in cloud systems,” *Algorithms*, vol. 11, no. 10, 2018, Art. no. 160.
[23] M. Torquato, P. Maciel, and M. Vieira, “Availability and reliability modeling of VM migration as rejuvenation on a system under varying workload,” *Softw. Qual. J.*, vol. 28, no. 1, pp. 59–83, 2020.
[24] J. Parri, S. Sampietro, L. Scommegna, and E. Vicario, “Evaluation of software aging in component-based web applications subject to soft errors over time,” in *Proc. IEEE Int. Symp. Softw. Rel. Eng. Workshops*, 2021, pp. 25–32.
[25] M. Torquato and M. Vieira, “An experimental study of software aging and rejuvenation in dockerd,” in *Proc. 15th Eur. Dependable Comput. Conf.*, 2019, pp. 1–6.
[26] N. Padhy, R. Panigrahi, and K. Neeraja, “Threshold estimation from software metrics by using evolutionary techniques and its proposed algorithms, models,” *Evol. Intell.*, vol. 14, no. 2, pp. 315–329, 2021.
[27] A. Das, W. Kong, A. Leach, S. Mathur, R. Sen, and R. Yu, “Long-term forecasting with tide: Time-series dense encoder,” 2023, *arXiv:2304.08424*.
[28] J. Audibert, P. Michiardi, F. Guyard, S. Marti, and M. A. Zuluaga, “USAD: Unsupervised anomaly detection on multivariate time series,” in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2020, pp. 3395–3404.
[29] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, “Robust anomaly detection for multivariate time series through stochastic recurrent neural network,” in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 2828–2837.
[30] A. Zeng, M. Chen, L. Zhang, and Q. Xu, “Are transformers effective for time series forecasting?,” in *Proc. AAAI Conf. Artif. Intell.*, 2023, pp. 11121–11128.
[31] G. Ke et al., “LightGBM: A highly efficient gradient boosting decision tree,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 3146–3154.

Jing Liu (Senior Member, IEEE) received the PhD degree in computer architecture from the Institute of Computing Technology, Chinese Academy of Sciences, China, in 2011. He is currently an full professor of computer science and technology with Inner Mongolia University. His major research interests include software reliability and cloud computing. He has published more than 70 papers in international conferences and journals.

Quan Zhou is currently working toward the MS degree with Inner Mongolia University, China. His research interests include software reliability and time series prediction.

Xingyu Chen is currently working toward the MS degree with Inner Mongolia University, China. His research interests include software reliability and fault prediction.

Jiantao Zhou (Member, IEEE) received the PhD degree from the Tsinghua University, in 2005. Since 1999, she has been on the faculty with Inner Mongolia University, China, where she is now a professor. Her research interests include formal methods, cloud computing and software engineering.

Keqin Li (Fellow, IEEE) received the BS degree in computer science from Tsinghua University, in 1985 and the PhD degree in computer science from the University of Houston in 1990. He is a SUNY distinguished professor with the State University of New York and a national distinguished professor with Hunan University (China). He has authored or co-authored more than 1200 journal articles, book chapters, and refereed conference papers. He holds nearly 80 patents announced or authorized by the Chinese National Intellectual Property Administration. He is among the world’s top few most influential scientists in parallel and distributed computing, regarding single-year impact and career-long impact based on a composite indicator of the Scopus citation database. He is listed in Scilit Top Cited Scholars (2023-2025) and is among the top 0.02% out of more than 20 million scholars worldwide based on top-cited publications in the last ten years. He is listed in ScholarGPS Highly Ranked Scholars (2022-2024) and is among the top 0.002% out of more than 30 million scholars worldwide based on a composite score of three ranking metrics for research productivity, impact, and quality in the recent five years. He received the IEEE TCCLD Research Impact Award from the IEEE CS Technical Committee on Cloud Computing in 2022 and the IEEE TCSVC Research Innovation Award from the IEEE CS Technical Community on Services Computing in 2023. He won the IEEE Region 1 Technological Innovation Award (Academic) in 2023. He was a recipient of the 2022-2023 International Science and Technology Cooperation Award and the 2023 Xiaoxiang Friendship Award of Hunan Province, China. He is a member of the SUNY Distinguished Academy. He is an AAAS fellow, an AAIA fellow, an ACIS fellow, and an AIIA fellow. He is a member of the European Academy of Sciences and Arts. He is a member of Academia Europaea (Academician of the Academy of Europe).