Journal of Computer Technology and Software

ISSN: 2998-2383

Vol. 3, No. 8, 2024

Deep Learning-Based Container Lifecycle Prediction in Cloud Computing Environments

Heyao Liu
Northeastern University, Boston, USA
liuheyao.arya@gmail.com

Abstract: In cloud computing environments, containers serve as the core units for resource scheduling and service deployment. Accurate prediction of their lifecycles is critical for improving system resource utilization and service stability. This paper addresses the challenge of dynamic factors affecting container lifecycles and the limited modeling capacity of traditional methods. It proposes a lifecycle prediction model based on deep regression networks. The proposed model takes multidimensional runtime monitoring metrics and system state information as inputs. It employs a multi-layer nonlinear structure to extract temporal features. An attention mechanism and embedding fusion module are integrated to enhance the modeling of key behavioral patterns. During training, regularization strategies are applied to improve model generalization. Mean squared error is used as the optimization objective to ensure stability and accuracy in continuous time prediction. To comprehensively evaluate the effectiveness of the proposed method, a container-level lifecycle prediction dataset was constructed based on the Google Cluster Trace. Experimental analysis was conducted from multiple perspectives, including hyperparameter sensitivity, system structure complexity, and sampling strategies. The results show that the proposed model outperforms several mainstream baseline methods in terms of MAE, RMSE, and R^2. It accurately reflects lifecycle trends and demonstrates strong adaptability and modeling stability.

Keywords: Container lifecycle prediction, deep regression network, time series modeling, resource-aware scheduling

1. Introduction

With the widespread adoption of cloud computing and containerization technologies, microservice architectures have become increasingly important in modern distributed systems. Containers, as ideal carriers for resource isolation, rapid deployment, and elastic scaling, have demonstrated high flexibility and controllability in practical applications[1]. However, managing the lifecycle of containers during runtime still faces many challenges. Accurate prediction of container lifecycles is increasingly critical, especially for resource scheduling, anomaly detection, and performance optimization. Understanding the dynamic evolution of containers from creation to termination has become a key factor in ensuring system stability and improving resource utilization[2].

Traditional container management strategies often rely on static rules or manual configurations. These approaches are difficult to adapt to the complex and changing resource loads in multi-tenant environments. In real-world production scenarios, container runtime durations are affected by multiple factors, such as task type, service dependencies, request density, and fluctuations in system load. The high variability of these factors makes simple statistical prediction methods inadequate. As a result, developing intelligent prediction models that can automatically learn from historical behaviors and generalize well has become a crucial research direction for improving container lifecycle management.

In recent years, deep learning has achieved remarkable progress in time-series modeling and behavior prediction tasks. These advancements provide strong theoretical and practical support for container lifecycle prediction. Deep regression networks, in particular, have demonstrated strong expressive power and modeling efficiency. Their ability to handle high-dimensional, multi-source heterogeneous data and to learn end-to-end representations makes them suitable for this task. Applying deep regression networks to container lifecycle prediction helps uncover hidden patterns in runtime data and enables precise characterization of lifecycle durations and fluctuations. This supports intelligent scheduling strategies and proactive resource reservation[3].

Moreover, predicting container lifecycles is not only a technical challenge but also directly related to resource scheduling and service quality in cloud-native architectures. From edge computing to hybrid cloud platforms, containers exhibit diverse lifecycle patterns under different computing environments. This variation places higher demands on the generalization and robustness of prediction models. Building lifecycle-aware prediction frameworks can help foresee potential load changes and estimate container durations in advance. This enhances system-level foresight, availability, and elasticity[4].

Therefore, studying container lifecycle prediction based on deep regression networks has both theoretical significance and practical value. Such models promote a shift from experience-driven to data-driven and intelligent decision-making in container management. They improve platform responsiveness and resource efficiency in complex operational scenarios. In addition, the behavioral patterns identified by these models can

support anomaly detection, load balancing, and energy control. This further strengthens the self-management capabilities and sustainability of cloud infrastructures.

2. Related work

In the development of containerized infrastructures, the management and prediction of container lifecycles have gradually become key research areas in cloud computing. Early work mainly focused on static resource allocation and rule-based scheduling strategies[5]. These approaches relied on predefined behavior templates or simple statistical models. While they offered basic predictive capabilities in some scenarios, their generalization was limited, and struggled in dynamic production environments. As container deployments expand and workload uncertainties increase, fixed rules, and thresholds are no longer sufficient for elastic system management and efficient scheduling. This has led to a growing interest in more intelligent lifecycle modeling methods.

In recent years, machine learning methods have been introduced into container performance modeling and lifecycle prediction tasks[6]. These approaches use historical runtime data to model container behaviors. Traditional algorithms such as regression models, decision trees, and support vector machines are commonly applied. They extract features from historical metrics and fit regression models to predict future container runtime or behavioral states. These methods have improved accuracy to some extent. However, due to their limited ability to represent high-dimensional features, they often fail to capture complex nonlinear patterns and hidden correlations among multi-source data. In large-scale concurrent services or multi-tenant scenarios, these models often suffer from stability issues.

With the rise of deep learning, an increasing number of studies have applied neural network architectures to container behavior modeling. Recurrent neural networks, convolutional neural networks, and their variants have been used for time-series representation learning[7]. These models can automatically extract temporal features and multidimensional dependencies from raw data. They show strong adaptability in lifecycle prediction, performance degradation detection, and anomaly detection tasks. To address service heterogeneity, some works have introduced attention mechanisms and graph neural networks. These techniques help model collaborative behaviors and structured dependencies among services, improving model generalization and multi-scenario adaptability.

Moreover, researchers have started to focus on model interpretability and lightweight design during training and deployment. Given the time sensitivity and computational constraints in lifecycle prediction, some studies aim to design compact yet expressive models. These models are often supported by feature selection, multi-task learning, and contrastive learning mechanisms. The goal is to enhance training efficiency and generalization. At the same time, there is a growing trend of integrating multi-source data, including system logs, trace data, and monitoring metrics, into joint modeling. This reflects the ongoing effort to balance prediction accuracy, model stability, and deployment feasibility.

3. Architecture and Methodology

This study proposes a container lifecycle prediction model based on a deep regression network, aiming to build an end-to-end framework that can efficiently model container runtime characteristics and accurately predict the length of its lifecycle. The model takes container-level monitoring indicators, system load information, and service context as input, performs feature extraction and nonlinear mapping through a multi-layer deep network structure, and finally outputs a continuous numerical representation of the remaining lifecycle of the container. In order to improve the modeling ability of complex time dependencies, the model introduces a multi-layer time series encoding module to improve prediction accuracy by capturing short-term fluctuations and long-term trends. The detailed structure of the proposed model is illustrated in Figure 1.

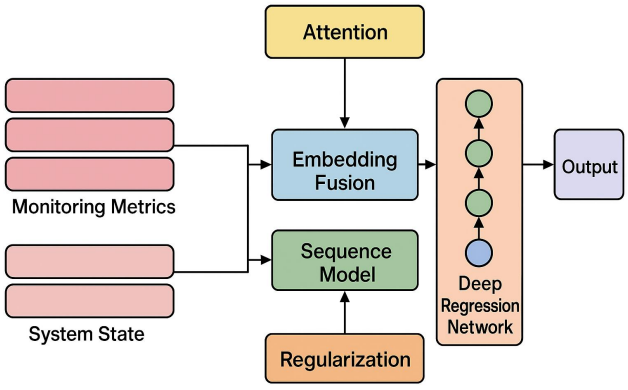

Figure 1. Deep regression-based architecture for container lifecycle prediction

Assume that the input sequence is $X = \{x_1, x_2, ..., x_T\}$, where each $x_t \in R^d$ represents a multidimensional feature vector at time step t. First, the original input is encoded through the temporal embedding layer to obtain the latent space representation:

$$h_t = RELU(W_e x_t + b_e)$$

Where $W_e \in R^{d' \times d}$ is the learnable projection matrix and b_e is the bias term. The representations of all time steps are concatenated and fed into a multi-layer feedforward neural network for deep feature extraction:

$$z = FFN([h_1, h_2, ..., h_T])$$

To enhance the model's ability to respond to different dimensions in the input features, an attention mechanism is introduced to integrate the information of key timing points, which is expressed as follows:

$$a_t = \frac{\exp(q^T h_t)}{\sum_{i=1}^{T} \exp(q^T h_i)}$$

$$c = \sum_{t=1}^{T} a_t h_t$$

Where q is the query vector, a_t represents the attention weight at the tth time step, and c is the weighted global context vector.

Finally, the lifetime prediction value is calculated through a fully connected regression layer:

$$\hat{y} = W_o c + b_o$$

Where $W_o \in R^{1 \times a'}, b_o \in R$ is the output layer parameter. The model is optimized using the mean square error loss function:

$$L = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

Where N is the number of samples, y_i is the real life cycle value, and $\hat{y}_i$ is the model prediction result. This method achieves efficient learning of the container life cycle through a full-process deep modeling mechanism and has good scalability and adaptability.

4. Experimental Results

4.1 Dataset

The dataset used in this study is derived from the publicly available Google Cluster Trace 2011. This dataset records scheduling logs and resource usage information from a real large-scale computing cluster. It captures the runtime trajectories of thousands of containerized tasks in a production environment. Key fields include task start and end times, CPU and memory usage, scheduling decisions, and service priorities. These features provide a rich and realistic data foundation for modeling container lifecycles.

The dataset exhibits strong temporal properties and high diversity. It reflects the dynamic characteristics of container workloads in real cloud platforms. By labeling the lifecycle of each container instance and extracting multidimensional resource usage across its runtime, time-series inputs for prediction can be constructed. This supports supervised learning for lifecycle regression modeling. In addition, the dataset covers a long time span and contains a large volume of data, making it suitable for evaluating model robustness and generalization in various scenarios.

During preprocessing, container tasks with complete lifecycle records were selected. Features such as CPU usage, memory usage, and scheduling counts were normalized. A sliding window approach was used to construct time-series samples. This ensures that each sample contains sufficient contextual information while meeting the forward-looking requirement of prediction. The final sample set was divided into training, validation, and test sets, providing a solid foundation for model training and performance evaluation.

4.2 Experimental setup

All experiments in this study were conducted on a local high-performance server. The hardware environment included two Intel Xeon Gold 6226R processors, each with 16 cores and a base frequency of 2.9 GHz, 512 GB of DDR4 memory, and two NVIDIA A100 GPUs with 40 GB of memory each. This setup ensures efficient parallel processing of large-scale time-series data and supports high-performance training of deep learning models. A 2 TB NVMe SSD was used as the storage device to handle frequent I/O operations and fast access to large intermediate results. This configuration supports multi-stage training, parameter tuning, and cross-validation, ensuring stable performance in resource-intensive tasks.

For the software environment, the operating system was Ubuntu 22.04 LTS. Python version 3.10 was used. The deep learning framework was PyTorch 2.1, combined with CUDA 12.1 and cuDNN 8.9 for GPU acceleration. Data preprocessing and visualization were performed using mainstream data science libraries, including Pandas, NumPy, Matplotlib, and Seaborn. During training, batch management was handled using PyTorch's DataLoader. The training process was monitored in real-time with TensorBoard, which tracked loss trends, parameter updates, and validation performance.

The experiments followed a standard supervised learning setup. The dataset was split into training, validation, and test sets with a ratio of 7:2:1. An early stopping strategy was used during training to prevent overfitting. The maximum number of epochs was set to 100. The batch size was 64. The Adam optimizer was used with an initial learning rate of 1e-4. To ensure reproducibility, all runs were conducted with fixed random seeds. Final evaluation results were based on the average performance across multiple runs.

4.3 Experimental Results

This paper first conducts a comparative experiment, and the experimental results are shown in Table 1.

Table 1: Comparative experimental results

Method	MAE	RMSE	R^2

Ours	0.138	0.194	0.931
LSTM[8]	0.184	0.248	0.882
GRU[9]	0.176	0.239	0.891
1DCNN[10]	0.162	0.226	0.903
Transformer[11]	0.149	0.211	0.918

The results in the table show that the proposed deep regression network achieves the best performance in the container lifecycle prediction task. The MAE and RMSE reach 0.138 and 0.194, respectively, which are significantly better than those of the baseline models. This indicates that the model has stronger fitting capabilities for modeling the temporal behaviors of containers. It captures complex resource usage patterns and lifecycle characteristics more accurately, effectively reducing prediction errors.

Compared to traditional recurrent structures such as LSTM and GRU, the proposed model shows significant improvements across all evaluation metrics. While LSTM and GRU possess certain abilities in modeling temporal sequences, they tend to suffer from performance bottlenecks when dealing with high-dimensional features and long-term dependencies. This leads to unstable prediction accuracy. The proposed model enhances the ability to capture lifecycle evolution by introducing deeper nonlinear mappings and dynamic feature fusion mechanisms. As a result, it performs better in terms of both stability and generalization.

Both 1D CNN and Transformer models also demonstrate good regression performance in the experiments. This suggests that convolutional structures and attention mechanisms are effective in extracting local and global temporal features. However, compared to the proposed model, these methods still face limitations when handling heterogeneous resource loads and fluctuating service behaviors. They struggle to fully adapt to the complex dynamics of container lifecycles. The proposed method addresses this by integrating multi-scale features and residual structures, which improves robustness against abnormal lifecycle trends.

Overall, the experimental results confirm the adaptability and accuracy advantages of the proposed model in the container lifecycle prediction task. The model performs well in controlling errors and achieves a high R^2 score of 0.931. This reflects its strong ability to fit lifecycle trends. Such capability is valuable for enabling intelligent resource scheduling and system alerting at the container level. It provides a reliable technical foundation for adaptive operations in cloud platforms.

In addition, this paper provides a detailed analysis of how varying the Dropout rate influences the prediction performance of the proposed model. By systematically adjusting the Dropout parameter during training, the study investigates its effect on the mode's ability to generalize across different data conditions. To support this analysis, the corresponding model architecture is illustrated in Figure 2, offering a clear structural reference for understanding the role of Dropout within the overall framework.

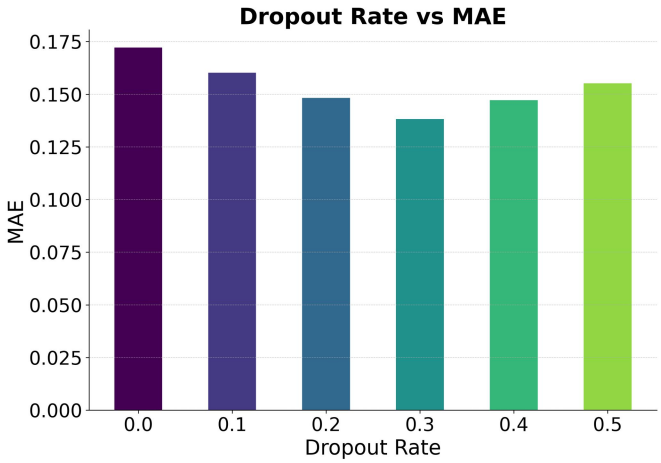

Figure 2. Analysis of the impact of Dropout rate on model prediction performance

The results in the figure show that the dropout rate has a significant impact on the prediction performance of the model. As the dropout rate increases from 0.0 to 0.3, the MAE value gradually decreases. This indicates that a moderate dropout mechanism helps alleviate overfitting during training. The model focuses more on representative feature expressions, which improves its generalization ability in predicting container lifecycles.

When the dropout rate reaches 0.3, the MAE value reaches its minimum. This suggests that the model performs best at this level, accurately capturing the temporal characteristics and resource consumption patterns of container operations. In the complex task of lifecycle prediction, moderate regularization improves the model's robustness to input noise and enhances both stability and accuracy.

However, as the dropout rate further increases to 0.4 and 0.5, the MAE value begins to rise. This suggests that an excessively high dropout rate weakens the information flow within the network. The model loses some key structural information during feature encoding. This loss reduces the model's sensitivity to changes in service behavior and decreases the accuracy of lifecycle prediction.

These findings suggest that when designing deep regression networks for container lifecycle prediction, the dropout parameter should be carefully set. A proper balance between model expressiveness and overfitting control is essential. Choosing an appropriate regularization strategy not only improves performance on the training set but also enhances the model's adaptability and generalization in complex cloud environments.

This paper also gives a comparison of model performance under different hidden layer dimension settings, and the experimental results are shown in Figure 3.

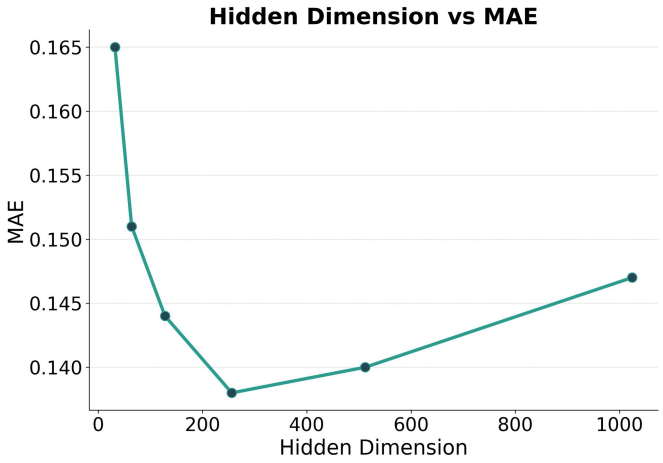

Figure 3. Comparison of model performance under different hidden layer dimension settings

The results in the figure show clear differences in model performance under different hidden layer dimensions. As the hidden dimension increases from 32 to 256, the MAE value continues to decrease. This indicates that expanding the network capacity at this stage helps enhance the model's feature representation ability. It enables more effective modeling of complex nonlinear relationships and temporal dependencies in container runtime data.

When the hidden dimension reaches 256, the MAE value drops to its lowest point. This suggests that this configuration achieves the best balance between modeling capacity and parameter complexity. For container lifecycle prediction, which is influenced by many factors, an appropriate hidden size can effectively extract long-term dependency features. It also avoids the loss of critical information caused by insufficient parameters. This improves prediction robustness in multi-tenant environments.

However, further increasing the hidden dimension to 512 and 1024 leads to a rise in MAE. This indicates that the model may enter an overfitting state. While larger dimensions improve representation capacity, they also introduce redundant parameters. The model becomes overly sensitive to small variations in the training data, which reduces its generalization to unseen data. This issue is especially prominent in production environments where container lifecycles fluctuate significantly.

Moreover, this paper explores the impact of data sampling frequency on the effectiveness of lifecycle prediction, emphasizing how different temporal granularities of input data may influence the model's perception of dynamic patterns. This aspect is particularly relevant for capturing fine-grained variations in container behavior. To illustrate this analysis more intuitively, the corresponding experimental setup and findings are visually presented in Figure 4.

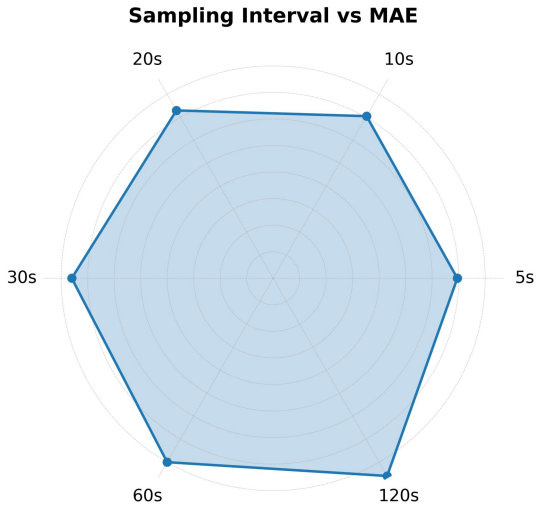

Figure 4. The impact of data sampling frequency on life cycle prediction results

The figure shows that data sampling frequency has a significant impact on the performance of container lifecycle prediction. The overall trend indicates that as the sampling interval increases from 5 seconds to 120 seconds, the MAE value rises steadily. This suggests that high-frequency sampling provides finer-grained temporal information. It helps the model capture rapid changes in resource usage during container execution, thereby improving prediction accuracy.

When the sampling interval is set to 5 or 10 seconds, the model shows stronger sensitivity in modeling short-term behaviors and bursty workloads. Container lifecycles are often influenced by service-level policies, load fluctuations, and scheduling decisions. A higher sampling frequency preserves these transient features and offers richer context to the model, which enhances predictive performance.

However, when the sampling interval increases to 60 seconds or more, the model performance degrades significantly. This indicates that low-frequency sampling introduces temporal sparsity in the input data. As a result, the model struggles to reconstruct the evolution of container states accurately. Missing information weakens the model's ability to perceive lifecycle trends and to detect potential anomalies, leading to prediction bias.

Therefore, this experiment highlights the critical role of sampling frequency in container lifecycle prediction. A well-chosen sampling granularity helps balance timeliness and information richness. It avoids unnecessary computational costs from over-sampling while ensuring the model receives sufficient dynamic behavior signals. This provides essential data support for advancing intelligent resource scheduling in cloud environments.

Finally, at the end of this paper, a graph is provided to illustrate the variation of the loss function over training epochs. This visualization helps to track the model's convergence behavior and offers additional insight into the training dynamics and optimization process of the proposed framework, as shown in Figure 5.

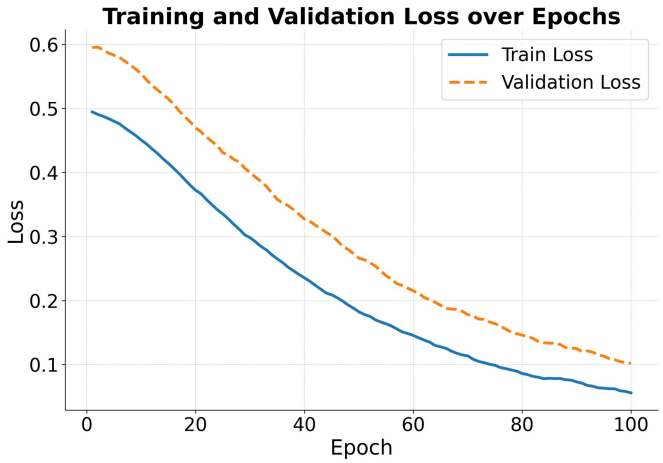

Figure 5. Loss function changes with epoch

The figure shows that the model exhibits a clear convergence trend throughout the training process. Both training loss and validation loss decrease steadily. This indicates that the model is continuously learning key resource evolution features within the container lifecycle and gradually optimizing its parameters to improve prediction accuracy. The stable decline of the loss curves reflects the effectiveness of the training process and confirms the soundness of the model architecture and optimization strategy.

The training loss decreases slightly faster than the validation loss. This suggests that the model fits the training data efficiently. The validation loss better reflects the model's generalization ability on unseen data. The validation curve remains close to the training curve throughout and does not show a significant rebound in later stages. This indicates that overfitting does not occur and the model adapts well to the complex and dynamic behavior of containers in real cloud environments.

After the 60th epoch, both curves become flat. This suggests that the model reaches a near-optimal performance range and further training yields diminishing returns. At this stage, training mainly fine-tunes the parameters and stabilizes performance. This provides a more robust foundation for regression tasks in lifecycle prediction. Such robustness is especially important for adaptability in multi-tenant and large-scale deployment environments.

This result confirms that the proposed deep regression framework demonstrates strong convergence and training stability. In container lifecycle prediction tasks, stable and fast convergence not only reduces training time but also builds a solid basis for efficient inference in production deployments. It provides critical support for enabling intelligent resource scheduling in cloud computing systems.

5. Conclusion

This study addresses the problem of lifecycle prediction in containerized cloud environments. It proposes a deep regression-based modeling approach that systematically overcomes limitations in expressiveness, temporal modeling accuracy, and generalization of existing methods. By integrating multi-source temporal features and applying nonlinear mappings, the model accurately captures the dynamic evolution of resource usage during container execution. This enables high-precision life-cycle prediction. The introduction of multi-layer structural optimization and regularization mechanisms improves prediction accuracy while maintaining model stability. The model adapts well to complex service behaviors and resource fluctuations in multi-tenant settings.

A series of well-designed experiments were conducted to evaluate the model's adaptability across different conditions. These evaluations considered hyperparameters, data characteristics, and execution environments. The results show that the proposed model consistently outperforms mainstream time-series prediction methods across multiple metrics. It demonstrates strong practical value and robustness in real cloud environments. Compared with traditional static scheduling strategies, this data-driven prediction method enables more proactive and precise support for dynamic scheduling and capacity planning. It provides both theoretical and methodological foundations for intelligent management of container infrastructures.

The findings not only improve the accuracy of container lifecycle modeling but also open new directions for automated resource optimization in cloud-native architectures. In edge computing, hybrid cloud management, and dense multi-tenant systems, the model shows good scalability. It can be integrated as a core component into scheduling systems to improve resource efficiency, reduce energy consumption, and enhance system responsiveness and service continuity. This makes it highly valuable for engineering applications and future deployment.

Future research may explore extensions of the model structure by incorporating adaptive learning mechanisms or reinforcement learning strategies. This would allow lifecycle prediction to become an active feedback component in scheduling decisions, forming a closed-loop system for resource management. Further improvements could involve multi-modal inputs that combine log analysis, trace data, and structured metrics. This would enhance the model's ability to understand system semantics and behavioral logic. As large-scale service architectures continue to evolve, building more efficient, intelligent, and interpretable prediction systems will be an important direction for advancing intelligent cloud platforms.

References

[1] Wu S, Tao Z, Fan H, et al. Container lifecycle‐aware scheduling for serverless computing[J]. Software: Practice and Experience, 2022, 52(2): 337-352.

[2] Tang X, Liu Q, Dong Y, et al. Fisher: An efficient container load prediction model with deep neural network in clouds[C]//2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom). IEEE, 2018: 199-206.

[3] Argesanu A I, Andreescu G D. A platform to manage the end-to-end lifecycle of batch-prediction machine learning models[C]//2021 IEEE 15th International Symposium on Applied Computational Intelligence and Informatics (SACI). IEEE, 2021: 000329-000334.

[4] Turin G, Borgarelli A, Donetti S, et al. Predicting resource consumption of Kubernetes container systems using resource models[J]. Journal of Systems and Software, 2023, 203: 111750.

[5] H. Wang, "Temporal-Semantic Graph Attention Networks for Cloud Anomaly Recognition," Transactions on Computational and Scientific Methods, vol. 4, no. 4, 2024.

[6] Kim B S, Lee S H, Lee Y R, et al. Design and implementation of cloud docker application architecture based on machine learning in container management for smart manufacturing[J]. Applied Sciences, 2022, 12(13): 6737.

[7] Wang Y, Zhu M, Yuan J, et al. The intelligent prediction and assessment of financial information risk in the cloud computing model[J]. arXiv preprint arXiv:2404.09322, 2024.

[8] Shiri F M, Perumal T, Mustapha N, et al. A comprehensive overview and comparative analysis on deep learning models: CNN, RNN, LSTM, GRU[J]. arXiv preprint arXiv:2305.17473, 2023.

[9] Nosouhian S, Nosouhian F, Khoshouei A K. A review of recurrent neural network architecture for sequence learning: Comparison between LSTM and GRU[J]. 2021.

[10] Ige A O, Sibiya M. State-of-the-art in 1d convolutional neural networks: A survey[J]. IEEE Access, 2024.

[11] Wen Q, Zhou T, Zhang C, et al. Transformers in time series: A survey[J]. arXiv preprint arXiv:2202.07125, 2022.