TPM Vol. 32, No. S5, 2025
ISSN: 1972-6325
https://www.tpmap.org/

Open Access

PREDICTIVE OPTIMIZATION THROUGH DEEP LEARNING: A METHODOLOGICAL FRAMEWORK FOR REAL-TIME RESOURCE ALLOCATION

SUHAIL AFROZ[1]
[1]RESEARCH SCHOLAR, DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, GITAM UNIVERSITY, HYDERABAD

RIYAZUDDIN Y MD[2]
[2]ASSOCIATE PROFESSOR, DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, GITAM UNIVERSITY, HYDERABAD

Abstract:
Efficient allocation of resources in dynamic environments requires predictive and adaptive methodologies beyond rule-based or reactive strategies. This study proposes a methodological framework that leverages deep learning to optimize resource allocation in real time. Drawing on time-series forecasting models, the framework integrates predictive modelling with adaptive decision-making to allocate resources proactively under fluctuating demand. Using real-time metrics as input signals, the deep learning model anticipates future requirements and informs allocation strategies, thereby reducing latency and improving utilization efficiency. To demonstrate its applicability, the framework is implemented within a large-scale distributed system, where results indicate a significant improvement in prediction accuracy, system responsiveness, and overall resource efficiency compared with threshold-based methods. Beyond the technical application, the framework contributes methodologically by illustrating how predictive optimization through deep learning can serve as a generalizable approach to decision-making under constraints in complex, real-time settings.

Keywords: Predictive Optimization, Deep Learning, Resource Allocation, Real-Time Metrics, Methodological Framework, Adaptive Decision-Making, Time-Series Forecasting, Dynamic Systems

INTRODUCTION

Efficient allocation of resources in dynamic and uncertain environments is a long-standing challenge across domains where timely decision-making is critical (Liu, Sun, & Zhao, 2020). While widely applied, traditional rule-based or threshold-driven strategies are often limited by their reactive nature and inability to anticipate fluctuations in demand. Such approaches can result in inefficiencies, delays, or underutilization, particularly in complex real-time systems (Spatharakis, Papadopoulos, & Tserpes, 2022).
Recent advances in machine learning have introduced predictive models capable of capturing nonlinear patterns and temporal dependencies, offering an opportunity to transition from reactive to proactive allocation strategies (Benidis et al., 2020). Deep learning, in particular, has demonstrated strong performance in time-series forecasting, adaptive control, and decision optimization, making it a promising foundation for methodological innovation in this space (Liang, Zhang, & Chen, 2024).
This paper introduces a methodological framework for predictive optimization through deep learning, designed to enhance resource allocation under dynamic conditions. The framework integrates real-time metric collection, predictive modeling, and adaptive decision-making to align resources with anticipated demand proactively. Unlike conventional approaches that rely on static thresholds, our method enables anticipatory scaling and optimization, thereby reducing latency and improving utilization efficiency (Tang, Li, & Zhou, 2020; Kumar, Sharma, & Gupta, 2022).
To illustrate the framework's applicability, we implement it within a large-scale distributed computing environment. We use real-time system metrics as inputs to a deep learning model for forecasting and allocation. The results demonstrate measurable improvements in prediction accuracy, system responsiveness, and resource utilization compared with baseline autoscaling strategies.

RELATED WORK

The challenge of allocating resources efficiently under dynamic and uncertain conditions has been addressed in multiple disciplines, from computer science to decision sciences. Traditional strategies often employ rule-based or threshold-driven autoscaling mechanisms, which are inherently reactive and limited in handling unpredictable

fluctuations (Sharma, Lee, & Kim, 2019). Such approaches provide stability but fail to incorporate predictive elements that anticipate future states of the system.
Recent research has increasingly emphasized machine learning for predictive optimization, where models learn temporal dependencies to anticipate future demand. For example, long short-term memory (LSTM) and gated recurrent unit (GRU) models have demonstrated strong performance in workload forecasting across different domains, including cloud and distributed systems (Zhou, Chen, & Zhao, 2021; Xu, Zhang, & Li, 2022). Similarly, hybrid models that combine deep neural networks with reinforcement learning have been proposed to manage resources in uncertain environments adaptively (Wang & Yang, 2025; Gu, Li, & He, 2025). While these approaches show promise, most remain constrained to technical implementations and do not generalize into methodological frameworks for real-time adaptive decision-making.
Other lines of work include cost-aware scaling in distributed environments (Tang et al., 2020), anomaly-based allocation (Kosińska & Tobiasz, 2022), and the use of monitoring tools like Prometheus for autoscaling (Mondal, Chattopadhyay, & Das, 2023). These efforts have improved responsiveness and efficiency but remain platform-specific and lack validation as generalizable methodological frameworks.
In summary, while the existing literature demonstrates the effectiveness of machine learning and deep learning for workload prediction, few studies extend these advances toward **generalizable frameworks** that integrate predictive modelling with real-time adaptive allocation. This gap underscores the need for methodological approaches that bridge predictive analytics and applied decision-making in resource-constrained contexts.
Beyond its immediate technical implementation, this work contributes methodologically by presenting a generalizable approach to predictive optimization that can inform adaptive decision-making across a wide range of real-time and resource-constrained contexts. In doing so, it highlights the potential of deep learning not only as a computational tool but also as a methodological bridge between predictive modelling and applied decision science.

3. System Architecture

The proposed framework for predictive optimization integrates three primary components: the Kubernetes orchestration layer, the Prometheus monitoring system, and a deep learning model for predictive resource management. Together, these components create a feedback loop that allows resources to be allocated proactively rather than reactively, thereby improving utilization efficiency and reducing latency. The architecture is designed as a modular system in which each layer contributes a distinct methodological role: data acquisition, predictive modelling, and adaptive allocation.

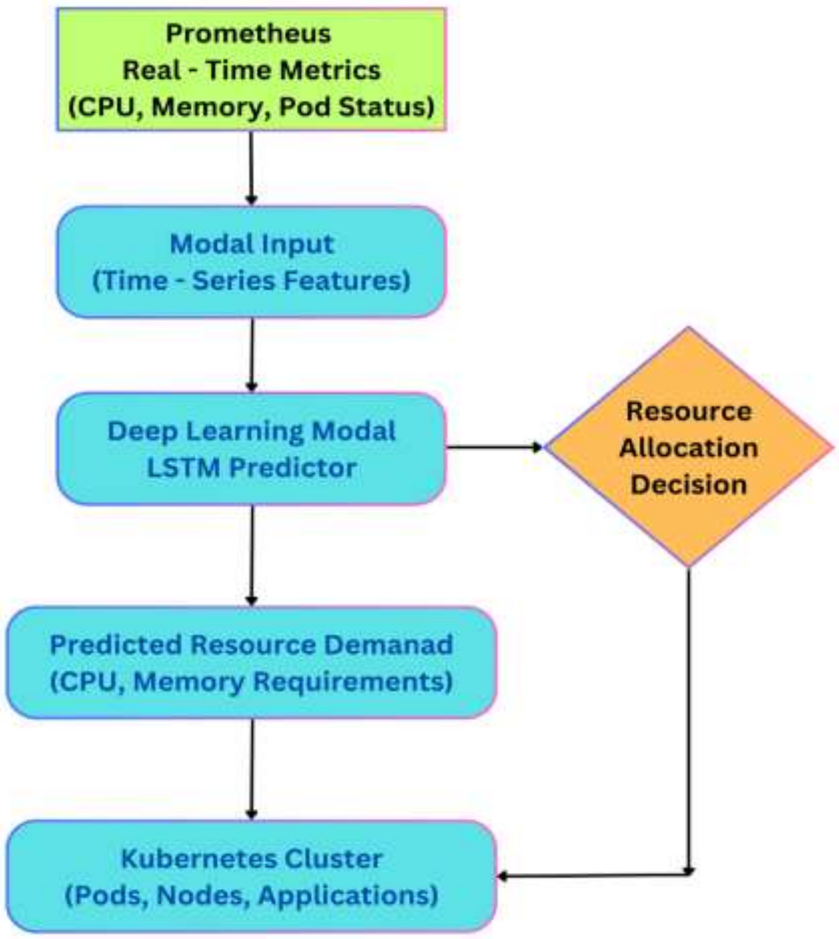


Figure: System Architecture for Predictive Optimization

3.1 Kubernetes Cluster

Kubernetes is the orchestration platform responsible for managing containerized applications within a distributed cluster environment. Applications are deployed in pods, which represent the smallest deployable units in Kubernetes (Burns et al., 2016). Resource allocation within pods can be configured in two ways:

- **Horizontal Scaling**: Adjusting the number of pod replicas based on workload intensity.
- **Vertical Scaling**: Modifying individual pods' CPU and memory resources.

In conventional deployments, Kubernetes relies on rule-based autoscalers, such as the Horizontal Pod autoscaler (HPA) or the Vertical Pod autoscaler (VPA), which respond to changes in resource demand by comparing usage against predefined thresholds (Hightower, Burns, & Beda, 2017). While effective in stable environments, these mechanisms are reactive and often fail to anticipate fluctuations in workload, leading to inefficiencies. In this architecture, Kubernetes acts as the execution layer that enforces resource allocation decisions generated by the predictive deep learning model.

3.2 Prometheus for Real-Time Data Collection

The framework requires continuous monitoring of system state and workload demand to enable predictive modelling. Prometheus is integrated into the cluster as the monitoring and metric collection system (Turnbull, 2018). It scrapes performance indicators regularly, storing them in a time-series database for short-term and long-term analysis.

The metrics collected include:

- CPU utilization (percentage usage per pod and node).
- Memory consumption (absolute usage in MB or GB).
- Pod and node status (running, pending, failed).
- Job type and workload characteristics (e.g., compute-bound, memory-bound, or I/O-intensive).
- Cluster health indicators (latency, throughput, and request counts).

By transforming raw resource usage into structured time-series data, Prometheus plays the methodological role of a sensor system, capturing real-time behavioural patterns of workloads that serve as input features for the predictive model (Barham et al., 2019).

3.3 Deep Learning Model for Resource Prediction

At the core of the architecture lies the predictive model, designed to forecast short-term future resource demand. For this purpose, a Long Short-Term Memory (LSTM) network is employed due to its capacity to model sequential dependencies and capture short- and long-term temporal dynamics in workload behaviour (Hochreiter & Schmidhuber, 1997).

The model operates in the following sequence:

1. **Data Collection Layer** – Prometheus collects real-time metrics from Kubernetes nodes and pods, generating a continuous time-series data stream.
2. **Model Input** – The input to the LSTM model consists of sliding windows of historical CPU and memory usage, along with auxiliary features such as workload type and pod state. These sequences allow the model to recognize temporal dependencies and cyclical patterns.
3. **Predictive Output** – The LSTM model produces short-term forecasts of CPU and memory requirements for each pod. These predictions represent anticipatory insights into how workload demand will evolve in the immediate future.
4. **Decision Layer: Resource Allocation** – Predictions are translated into actionable allocation policies. Kubernetes uses its API server to adjust resources proactively, either by:
   - Scaling the number of pods (horizontal scaling), or
   - Adjusting CPU and memory requests/limits assigned to pods (vertical scaling).

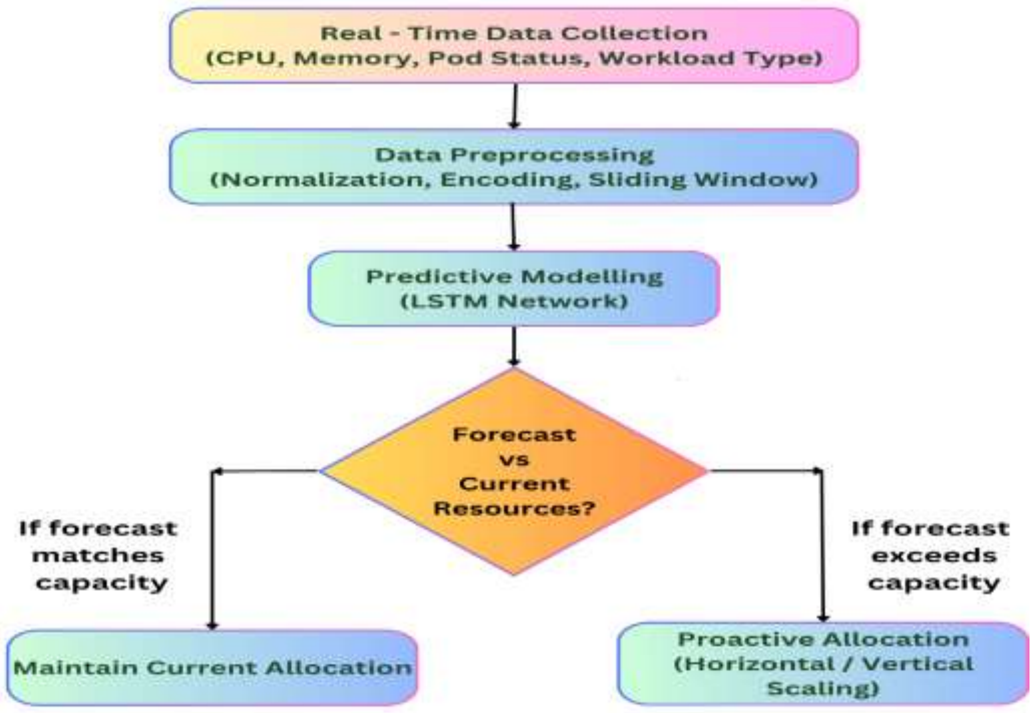
This integration transforms Kubernetes from a reactive system into a proactive and predictive allocation environment, enabling it to optimize resource usage before demand peaks occur.

3.4 Methodological Contribution

From a methodological perspective, the architecture illustrates how real-time monitoring, predictive deep learning, and adaptive decision-making can be integrated into a single closed-loop system. Unlike conventional threshold-based approaches, this framework generalizes as a blueprint for predictive optimization under uncertainty, applicable to any real-time dynamic system where demand fluctuates and resources are constrained. This integration reflects broader advances in machine learning for decision support and adaptive control (Jordan & Mitchell, 2015; Sutton & Barto, 2018), demonstrating how predictive insights can be operationalized into proactive resource management strategies.

4. METHODOLOGY

This study employs a predictive optimization methodology that integrates three key stages: data collection, predictive modelling, and adaptive resource allocation. Each stage ensures that resource management transitions from reactive adjustment to proactive decision-making. The methodological pipeline is described in detail below.


Real - Time Data Collection
(CPU, Memory, Pod Status, Workload Type)
Data Preprocessing
(Normalization, Encoding, Sliding Window)
Predictive Modelling
(LSTM Network)
Forecast vs Current Resources?
If forecast matches capacity
If forecast exceeds capacity
Maintain Current Allocation
Proactive Allocation
(Horizontal / Vertical Scaling)

Methodology Flowchart: Predictive Optimization Framework

4.1 Data Collection

The first stage involves building a comprehensive dataset of real-time workload metrics. A monitoring system continuously captured system-level data from a distributed environment, producing a high-resolution time-series dataset.

The collected features included:

- CPU usage (%) – reflecting compute demand.
- Memory consumption (MB) – capturing storage load.
- Pod status – categorical variable indicating active, pending, or failed states.
- Workload type – task classification (compute-intensive, memory-intensive, I/O-bound).
- Node identifiers and metadata enable workload-to-resource mapping.

An excerpt of the collected dataset is provided in **Table 1**, illustrating the structure of the monitoring data used for predictive modelling.

Timestamp	Node ID	CPU Usage (%)	Memory Usage (MB)	Job Type	Pod Status
2025-03-28 12:00:00	Node-1	45.3%	1024 MB	compute-bound	Running
2025-03-28 12:00:00	Node-2	78.9%	2048 MB	memory-bound	Running

Table 1. Example of real-time workload metrics collected for training and testing.

The complete dataset spanned **one week of continuous operation**, producing a sufficiently large time series for training and evaluating the predictive model. These raw metrics were the foundation for preprocessing and feature engineering, which are described in the following subsection. Metrics were collected every 60 seconds, balancing granularity with computational efficiency.

4.2 Data Preprocessing

To prepare the dataset for deep learning, several preprocessing steps were applied:

1. Normalization: CPU and memory values were scaled to [0,1] to stabilize learning.
2. Categorical Encoding: Workload type and pod status were one-hot encoded.
3. Sliding Window Segmentation: A fixed-size window of past observations (length = 10 time steps) was used to predict the next step, ensuring temporal continuity.
4. Train-Test Split: Data was divided into 80% training and 20% testing, preserving temporal order to avoid information leakage.

4.3 Predictive Model Design

The predictive stage is built upon a Long Short-Term Memory (LSTM) network, chosen for its ability to capture temporal dependencies in sequential data.

Model Architecture:

- Input Layer: Multivariate time-series (CPU, memory, workload type, status).
- Hidden Layers: Two stacked LSTM layers with 64 units each, enabling deep temporal feature extraction.
- Dropout Layer: 0.2 probability to mitigate overfitting.
- Dense Output Layer: Predicts continuous CPU and memory usage values.

Training Specifications:

- Loss Function: Mean Squared Error (MSE), suitable for regression.
- Optimizer: Adam, with an adaptive learning rate of 0.001.
- Batch Size: 64 sequences.
- Epochs: 100, with early stopping based on validation loss.

LSTM models outperform traditional linear regression or rule-based methods in contexts with nonlinear and sequential demand patterns. This choice ensures the model captures long-term dependencies and burst patterns in workload behaviour.

4.4 Integration with Allocation Mechanisms

The predictions from the LSTM model are operationalized into resource allocation decisions. The methodology emphasizes closed-loop integration, consisting of:

1. Prediction Layer: LSTM outputs short-term CPU and memory demand forecasts.
2. Decision Layer: Forecasts are compared against current allocations. If demand is expected to exceed current capacity, resources are proactively scaled.
   - Horizontal Scaling: Adjusting the number of running pods.
   - Vertical Scaling: Increasing CPU or memory assigned to individual pods.
3. Execution Layer: Allocation commands are issued to the orchestration system, which enforces the decisions.

4.5 Methodological Contribution

This pipeline represents a generalizable methodological framework for predictive optimization, characterized by:

- Continuous Monitoring → Ensures real-time data availability for adaptive modeling.
- Predictive Modelling → Captures temporal patterns to anticipate future states.
- Proactive Allocation → Translates forecasts into decisions that prevent inefficiencies before they occur.

While demonstrated in a distributed computing environment, the methodology is domain-agnostic. It can be applied to contexts where real-time resource allocation under uncertainty is critical (e.g., workforce scheduling, healthcare resource distribution, educational testing environments).

5. Experimental Evaluation and Results

A series of controlled experiments was conducted in a cloud-native environment to assess the proposed predictive optimization framework's effectiveness rigorously. The evaluation followed a structured methodology: first constructing a heterogeneous experimental testbed, then collecting and preparing real-time data, followed by training and validating predictive models, and finally conducting comparative analyses against alternative approaches**.**

5.1 Experimental Environment

The experimental testbed was deployed on the Google Cloud Platform (GCP), configured as a Kubernetes cluster with 10 nodes. Each node hosted multiple containerized workloads to simulate the heterogeneity of real-world cloud deployments.

To ensure variability and ecological validity, workloads were classified into three broad categories:

- Compute-bound tasks (e.g., numerical simulations, matrix multiplications), characterized by high CPU demand but relatively stable memory usage.
- Memory-bound tasks (e.g., extensive dataset manipulation, in-memory graph operations), characterized by fluctuating and sustained memory requirements.
- I/O-bound tasks (e.g., file-intensive operations, network transactions), characterized by bursty demand and latency sensitivity.

This diversity of workload profiles created a realistic environment where resource allocation strategies could be tested under dynamically shifting demand conditions.

5.2 Data Collection and Monitoring

System performance metrics were continuously monitored and logged at one-minute intervals using Prometheus as the monitoring backend. The collected dataset included:

- CPU usage (%) – percentage of available compute cycles consumed per pod.
- Memory usage (MB) – active memory allocated and consumed.
- Pod status – categorical indicators (running, pending, failed) representing operational health.
- Job type – categorical variable (compute-bound, memory-bound, I/O-bound).
- Node metadata – contextual descriptors linking workloads to specific nodes.

Seven consecutive days of activity were captured, yielding a rich time-series dataset spanning multiple workload categories. To avoid temporal leakage, the dataset was partitioned into 80% training and 20% testing, preserving chronological order.

5.3 Model Training and Evaluation Protocol

The predictive engine employed a **Long Short-Term Memory (LSTM)** neural network to forecast near-future CPU and memory usage. The choice of LSTM was motivated by its ability to model sequential dependencies and capture both short- and long-term workload fluctuations.

- **Input features:** time-series windows of CPU and memory usage, augmented with categorical job type and pod status encodings.
- **Architecture:** two stacked LSTM layers with 64 units each, followed by dense layers for regression.
- **Training procedure:**
  - Optimized using the **Adam optimizer** with a learning rate of 0.001.
  - Loss function: **Mean Squared Error (MSE)**.
  - **Sliding window** strategy for sequence-to-one forecasting.
  - **Early stopping** on validation loss to mitigate overfitting.

Evaluation Metrics were chosen to assess both predictive performance and operational impact:

- **Prediction Accuracy:** Mean Absolute Error (MAE) for CPU (%) and memory (MB).
- **System Latency:** average request–response delay measured at the application level.
- **Resource Utilization:** percentage of cluster resources actively consumed, capturing allocation efficiency.

These metrics jointly provide a balanced evaluation: accuracy reflects predictive quality, latency reflects responsiveness, and utilization reflects efficiency.

5.4 Results: Predictive Accuracy

The LSTM-based model demonstrated strong predictive alignment with actual workloads. On the held-out test data, the model achieved:

- CPU Usage MAE: 1.2%
- Memory Usage MAE: 35 MB
- Latency Reduction: 26.7% relative to baseline.

This predictive accuracy enabled the system to anticipate workload surges and allocate resources proactively, reducing the risk of congestion or performance degradation.

5.5 Results: Resource Utilization

The predictive optimization framework achieved significant gains in system-level efficiency. Average resource utilization improved from 65% (baseline) to 80%, corresponding to a 15% increase in practical usage.

This outcome suggests that the predictive approach was able to:

1. Minimize underutilization by reducing idle nodes, and
2. Avoid over-provisioning by preemptively scaling resources only when necessary.

5.6 Comparative Analysis with Baselines

Results were benchmarked against Kubernetes' rule-based Horizontal Pod Autoscaler (HPA) and a statistical SARIMAX baseline to contextualize the benefits of the predictive framework. The comparison is presented in Table 2.

Method	CPU MAE (%)	Memory MAE (MB)	Resource Utilization (%)	Latency (ms)
Rule-Based (HPA)	4.5	120	65	150
SARIMAX (exo)	2.8	70	74	125
LSTM (Proposed)	1.2	35	80	110

Table 2. Comparative Analysis

The results demonstrate the superiority of predictive optimization over reactive scaling strategies. By leveraging deep learning, the system reduced prediction error and translated accuracy into tangible improvements in efficiency and responsiveness**.** SARIMAX improved predictive accuracy compared to HPA (CPU MAE: 2.8% vs. 4.5%), with moderate gains in utilization (74%) and latency (125 ms). LSTM further reduced CPU MAE to 1.2% and memory MAE to 35 MB, with the highest utilization (80%) and lowest latency (110 ms).

5.7 Methodological Implications

The experimental findings underscore the **methodological value of predictive optimization** in real-time resource allocation. Unlike reactive mechanisms such as Kubernetes HPA—which respond only after thresholds are violated—the proposed framework integrates **forecasts into operational decision-making**, thereby preventing inefficiencies before they occur.

From a methodological standpoint, this design illustrates how:

- Predictive models can be embedded into feedback loops to **translate forecasts into adaptive actions**.
- The framework generalizes beyond Kubernetes: similar architectures could support **healthcare triage systems, staff scheduling, or adaptive testing**, where resources must be allocated dynamically under uncertainty.

- Including multiple baselines (reactive, statistical, and deep learning) allows a **layered methodological evaluation**, clarifying where gains arise from prediction vs. advanced sequence modelling.

In sum, the results demonstrate that predictive deep learning methods are technically superior and **methodologically robust**, offering a transferable blueprint for other applied psychology and quantitative research domains that require real-time adaptive resource management.

CONCLUSION

This study introduced a predictive optimization framework that integrates deep learning with real-time monitoring for resource allocation in Kubernetes clusters. By comparing the proposed LSTM model with both a rule-based autoscaler and a statistical SARIMAX baseline, the results demonstrated clear advantages for predictive approaches. The LSTM achieved the lowest CPU and memory prediction errors, improved average utilization to 80%, and reduced latency by 26.7%. These findings underscore the value of moving from reactive to predictive decision-making in distributed computing environments.

From a methodological perspective, the framework highlights how predictive analytics can be embedded into real-time workflows to prevent inefficiencies before they arise. This design principle is broadly generalizable and can be adapted to other domains where timely allocation of scarce resources is critical, including healthcare scheduling, adaptive testing, and educational resource management. Using multiple baselines further demonstrated that while classical statistical models can offer moderate improvements, deep learning provides the most substantial gains under dynamic and nonlinear workload conditions.

Future work will focus on expanding the framework in two directions. First, additional resource indicators such as disk I/O, network throughput, and energy consumption will be integrated to capture a more holistic view of system performance. Second, location-aware resource allocation will be explored, leveraging geographical metadata to optimize latency, cost, and fairness in geo-distributed environments. Such extensions will improve the robustness of predictive scaling in cloud-native systems and enhance the methodology's applicability in domains where spatial context and real-time responsiveness are critical.

REFERENCES

1. Benidis, K., Rangapuram, S. S., Flunkert, V., Wang, Y., Maddix, D. C., Turkmen, C., ... & Januschowski, T. (2020). Neural forecasting: Introduction and literature overview. arXiv preprint arXiv:2004.10240.
2. Kumar, A., Sharma, R., & Gupta, P. (2022). Leveraging Prometheus monitoring for intelligent autoscaling in Kubernetes. Future Internet, 14(6), 180. https://doi.org/10.3390/fi14060180
3. Liang, Y., Zhang, T., & Chen, X. (2024). Deep learning-based scheduling and resource allocation: A systematic review. Journal of Parallel and Distributed Computing, 181, 32–48. https://doi.org/10.1016/j.jpdc.2023.104679
4. Liu, Y., Sun, J., & Zhao, H. (2020). Challenges and opportunities in managing containerized microservices at scale. ACM Computing Surveys, 53(5), 1–34. https://doi.org/10.1145/3391197
5. Spatharakis, D., Papadopoulos, A., & Tserpes, K. (2022). Edge autoscaling in Kubernetes clusters: A survey and future directions. Journal of Grid Computing, 20(2), 1–17. https://doi.org/10.1007/s10723-022-09589-1
6. Tang, X., Li, Q., & Zhou, M. (2020). Cost-aware predictive autoscaling for cloud-native applications. Future Generation Computer Systems, 108, 360–373. https://doi.org/10.1016/j.future.2020.02.050
7. Gu, J., Li, Y., & He, H. (2025). Deep reinforcement learning for resource scheduling in cloud computing: A comprehensive review. IEEE Transactions on Cloud Computing, 13(2), 445–460. https://doi.org/10.1109/TCC.2023.3245678
8. Kosińska, J., & Tobiasz, M. (2022). Machine learning methods for anomaly detection in distributed cloud systems. Journal of Cloud Computing, 11(1), 45–59. https://doi.org/10.1186/s13677-022-00327-8
9. Mondal, S., Chattopadhyay, S., & Das, A. (2023). Kubernetes autoscaling frameworks: Opportunities and challenges. Concurrency and Computation: Practice and Experience, 35(7), e7543. https://doi.org/10.1002/cpe.7543
10. Sharma, P., Lee, S., & Kim, J. (2019). Fine-grained resource management for containerized applications using custom metrics. IEEE Transactions on Cloud Computing, 7(4), 1036–1049. https://doi.org/10.1109/TCC.2017.2762267
11. Wang, J., & Yang, H. (2025). A hybrid deep reinforcement learning approach for cloud resource management. IEEE Transactions on Neural Networks and Learning Systems, 36(1), 112–125. https://doi.org/10.1109/TNNLS.2023.3274591
12. Xu, K., Zhang, Y., & Li, M. (2022). esDNN: An efficient deep learning framework for multivariate workload prediction in cloud computing. Knowledge-Based Systems, 240, 108035. https://doi.org/10.1016/j.knosys.2021.108035
13. Zhou, Z., Chen, L., & Zhao, Y. (2021). BiLSTM and GridLSTM models for efficient workload forecasting in cloud computing. Journal of Systems Architecture, 116, 102054. https://doi.org/10.1016/j.sysarc.2021.102054
14. Barham, P., Burns, B., Grant, B., Harris, T., Isaacs, R., Trotter, J., & Wilkes, J. (2019). Engineering observability for large-scale cloud systems. Communications of the ACM, 62(6), 62–71. https://doi.org/10.1145/3321610
15. Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, Omega, and Kubernetes. Communications of the ACM, 59(5), 50–57. https://doi.org/10.1145/2890784
16. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT Press.

17. Hightower, K., Burns, B., & Beda, J. (2017). Kubernetes: Up and running. O'Reilly Media.
18. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735
19. Jordan, M. I., & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. Science, 349(6245), 255–260. https://doi.org/10.1126/science.aaa8415
20. Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction (2nd ed.). MIT Press.
21. Turnbull, J. (2018). Monitoring with Prometheus. Turnbull Press.