Towards Intelligent Cloud Scheduling: DynaSched-Net with Reinforcement Learning and Predictive Modeling

Yiming Yu *

Posted Date: 3 June 2025

doi: 10.20944/preprints202506.0129.v1

Keywords: oud resource scheduling; reinforcement learning; LSTM-Transformer; load balancing; DynaSched-Net

 doi:10.20944/preprints202506.0129.v1

Article

Towards Intelligent Cloud Scheduling: DynaSched-Net with Reinforcement Learning and Predictive Modeling

Yiming Yu

New York University, New York, USA; yy2210@nyu.edu

Abstract: Dynamic cloud resource scheduling needs real-time adaptation to changing workloads to keep system performance high and stable. Traditional methods like FCFS and RR lack the ability to adjust resources dynamically in complex conditions. This paper presents DynaSched-Net, a dual-network framework that uses a Deep Q-Network (DQN)-based reinforcement learning scheduler and a hybrid LSTM-Transformer predictor. The reinforcement learning module assigns resources based on system states to improve load balance. The predictor learns short-term and long-term workload patterns to guide decisions. A joint loss function helps optimize both parts of the system. Stabilization methods like experience replay and target network updates help keep training stable. Experiments show that DynaSched-Net performs better than traditional methods and provides an efficient way to manage cloud resources.

Keywords: cloud resource scheduling; reinforcement learning; LSTM-Transformer; load balancing; DynaSched-Net

---

1. Introduction

Cloud computing has increased the need for smart resource scheduling to keep systems working well during high and changing loads. Basic algorithms like First-Come-First-Serve (FCFS) and Round Robin (RR) do not adapt and often fail when workloads change, which leads to wasted resources. Reinforcement learning (RL) helps solve this by letting systems learn to adjust resources based on real-time feedback. Wang et al.[1] used deep reinforcement learning to improve resource use in cloud-native wireless networks and showed good results.

RL can adapt to changes, but it does not predict future workloads. This makes it less useful when workloads change quickly. Rossi et al.[2] built a forecasting model that uses transfer learning to predict workload changes better. Arbat et al.[3] also worked on workload forecasting and designed a Wasserstein Adversarial Transformer to capture complex patterns and improve prediction.

This paper introduces DynaSched-Net, a dual-network framework that uses a DQN-based RL scheduler and a hybrid LSTM-Transformer predictor. The RL part assigns resources based on the system's current state. The predictor finds short-term and long-term patterns and gives advice to the scheduler. A joint loss function optimizes both parts at the same time. Stabilization methods like experience replay and target network updates keep training steady. This system improves fast responses and planning and gives a practical way to schedule resources in cloud environments.

2. Related Work

Hybrid machine learning models have shown good results in solving hard optimization tasks in cloud systems. Jin et al.[4] proposed a machine learning framework that improves supply chain risk prediction by combining different learning methods. Wang et al.[5] designed a hybrid FM-GCN-Attention model for personalized recommendations. Their model mixes factorization machines, graph convolutional networks, and attention methods to better capture features.

 doi:10.20944/preprints202506.0129.v1

Chen et al.[6] developed a coarse-to-fine multi-view 3D reconstruction system using SLAM optimization and Transformer-based matching. This work showed that Transformer networks can handle large tasks with complex inputs. Zhou et al.[7] reviewed deep reinforcement learning (DRL) methods for cloud resource scheduling and noted that DRL can adapt well in real time but needs stable training and can be slow. Gu et al.[8] also reviewed DRL methods and pointed out challenges when working with different kinds of workloads.

Zhao et al.[9] suggested using a multi-agent graph reinforcement learning method for large-scale cluster scheduling. Their method allows systems to make decisions together and work better on big tasks, but managing many agents is still a problem.

3. Methodology

We present **DynaSched-Net**, a dynamic cloud resource allocation framework that integrates reinforcement learning (RL) with hybrid LSTM-Transformer-based load forecasting. The RL scheduler adapts task assignment to real-time system states, while the predictive module anticipates future workloads. This joint approach enables efficient resource utilization, reduced response time, and enhanced system balance under high concurrency. Experiments demonstrate that **DynaSched-Net** surpasses traditional methods in load distribution, task latency, and processing efficiency. The overall architecture is illustrated in Figure 1.

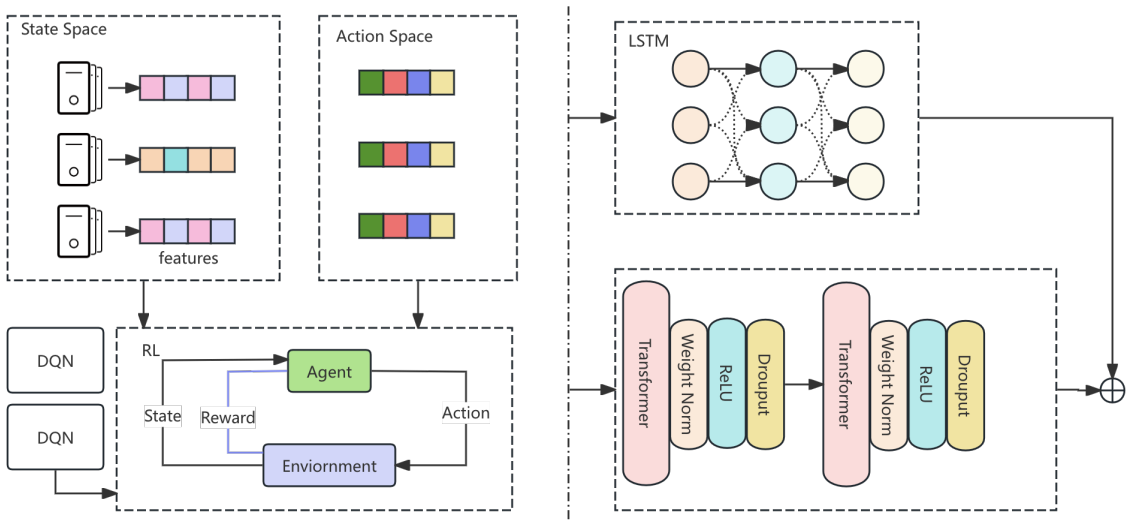

Figure 1. The pipeline of the DynaSched-Net Module.

3.1. Reinforcement Learning Component

We model the dynamic scheduling task as a Markov Decision Process (MDP), where the RL agent observes system states, takes actions, and receives rewards. The agent learns a policy to maximize long-term rewards, promoting efficient resource allocation and balanced load distribution.

3.1.1. State and Action Definition

The state space $\mathcal{S}$ captures system features at time t, including load, available resources, and task queue length:

$$\mathcal{S}_t = [\text{load}_t, \text{available\_resources}_t, \text{task\_queue}_t, \dots] \tag{1}$$

The action space $\mathcal{A}$ defines resource allocation decisions as continuous values:

$$\mathcal{A}_t = [r_{task_1}, r_{task_2}, \dots, r_{task_n}] \tag{2}$$

where r_{task_i} denotes the resources assigned to task i at time t.

 doi:10.20944/preprints202506.0129.v1

3.1.2. Reward Function

The reward function $r(s_t, a_t)$ guides the agent by penalizing inefficient resource usage. Actions leading to overload or underutilization incur negative rewards:

$$r(s_t, a_t) = -(\alpha \cdot \text{over\_load}(s_t, a_t) + \beta \cdot \text{under\_load}(s_t, a_t)) \tag{3}$$

Here, α and β adjust the penalty trade-off between over- and under-utilization levels.

3.1.3. Deep Q-Network (DQN) Architecture

We employ a Deep Q-Network (DQN) to approximate the Q-function, which evaluates state-action pairs using the Bellman equation:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \tag{4}$$

where γ is the discount factor for future rewards.

The DQN comprises fully connected layers mapping input state s_t to Q-values:

$$\text{Q-values} = \text{NeuralNetwork}(s_t) \tag{5}$$

Training minimizes the loss between predicted and target Q-values:

$$\mathcal{L}(\theta) = \mathbb{E}\left[(y_t - Q(s_t, a_t, \theta))^2\right] \tag{6}$$

with target value:

$$y_t = r(s_t, a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}, \theta^-) \tag{7}$$

where θ^- denotes parameters of the fixed target network.

3.1.4. Training Process for the RL Agent

The RL agent is trained via experience replay to stabilize learning. At each step, the agent stores transitions $(s_t, a_t, r(s_t, a_t), s_{t+1})$ in a replay buffer and samples mini-batches to update the Q-network parameters θ. The training involves:

1. Randomly initialize Q-network weights.
2. Observe state s_t, select and execute action a_t.
3. Record next state s_{t+1} and reward $r(s_t, a_t)$.
4. Store the transition in the replay buffer.
5. Sample mini-batches and compute the loss.
6. Apply gradient descent to update θ.
7. Periodically sync the target network for stability.

3.2. Prediction Network for Load Forecasting

To assist the RL agent, we design a hybrid prediction network that forecasts future system load and task arrivals. It combines LSTM for short-term patterns and Transformer for capturing long-range dependencies.

3.2.1. LSTM Encoder

The LSTM encoder models the temporal patterns in past system loads and task arrivals. At each time step t, the output is:

$$h_t = \text{LSTM}(x_t, h_{t-1}) \tag{8}$$

where x_t is the input and h_{t-1} is the previous hidden state.

 doi:10.20944/preprints202506.0129.v1

3.2.2. Transformer Encoder

To model long-range dependencies in time-series data, we employ a Transformer encoder with self-attention. This mechanism assigns different weights to time steps, aiding in forecasting complex load patterns:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{9}$$

where Q, K, and V denote the query, key, and value matrices, and d_k is the key dimension.

3.2.3. Output of the Prediction Network

The output of the combined LSTM-Transformer network is the predicted system load for the next T time steps, denoted as $\hat{y}_t$. The prediction model is trained by minimizing the Mean Squared Error (MSE) loss:

$$\mathcal{L}_{\text{predict}} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{10}$$

where y_i is the true load at time i, and $\hat{y}_i$ is the predicted load.

3.3. Loss Function

The **DynaSched-Net** loss integrates objectives from both the RL agent and the prediction network. It jointly optimizes the scheduling policy and forecasting accuracy, with total loss defined as a weighted sum of RL loss and prediction loss, controlled by hyperparameter λ. Figure 2 shows the evolution and dynamic weighting of both losses during training.

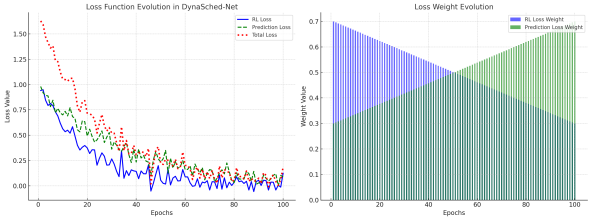

Figure 2. Loss Function Evolution and Loss Weight Changes in DynaSched-Net. The left plot shows the evolution of RL loss, prediction loss, and total loss, while the right plot illustrates the weight changes of RL and prediction losses over epochs.

3.3.1. Reinforcement Learning Loss

To optimize long-term rewards, the RL loss is defined using Q-learning as the mean squared error between the predicted and target Q-values:

$$\mathcal{L}_{\text{RL}}(\theta) = \mathbb{E}\left[\left(r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a', \theta^-) - Q(s_t, a_t, \theta)\right)^2\right] \tag{11}$$

where $Q(s, a, \theta)$ is the estimated Q-value, r the reward, γ the discount factor, and θ^- the target network parameters.

3.3.2. Prediction Loss

The prediction component aims to minimize the forecasting error of the system load. The prediction network's loss is computed as the Mean Squared Error (MSE) between the predicted load $\hat{y}_t$ and the true load y_t. The prediction loss is given by:

$$\mathcal{L}_{\text{predict}} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{12}$$

 doi:10.20944/preprints202506.0129.v1

where: - y_i is the true load for time step i. - $\hat{y}_i$ is the predicted load at time step i. - N is the total number of time steps in the prediction horizon.

3.3.3. Total Loss Function

The total loss function combines the losses from both the RL and prediction components to optimize the overall performance of the framework. The final loss is expressed as a weighted sum of the RL loss and the prediction loss:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{RL}}(\theta) + \lambda \cdot \mathcal{L}_{\text{predict}} \tag{13}$$

where: - λ is a hyperparameter that controls the balance between the RL and prediction components. - The first term, $\mathcal{L}_{\text{RL}}(\theta)$, encourages the RL agent to improve its resource scheduling decisions. - The second term, $\mathcal{L}_{\text{predict}}$, encourages the prediction model to accurately forecast the system load.

By optimizing this combined loss function, the system can both efficiently allocate resources and predict future load accurately, ensuring optimal performance across cloud resource scheduling tasks.

3.4. Data Preprocessing

The raw input comprises `work_order.csv` and `process_time_matrix.csv`. Preprocessing is crucial for converting this data into a format compatible with RL and prediction model training. Key steps are summarized below.

3.4.1. Normalization of Task Data

Task arrival times and durations are normalized using z-score transformation to ensure consistent feature scales and stable training:

$$\tilde{x}_i = \frac{x_i - \mu}{\sigma} \tag{14}$$

where x_i is the raw value, μ and σ are the feature's mean and standard deviation, and $\tilde{x}_i$ is the normalized output. This ensures balanced contribution from each feature during training.

3.4.2. Time-Series Transformation

To enable load forecasting, task arrival times and system load are converted into fixed-length time windows for LSTM-Transformer training:

$$\mathbf{X}_t = [x_{t-1}, x_{t-2}, \ldots, x_{t-k}] \tag{15}$$

where $\mathbf{X}_t$ is the input at time t, and x_{t-k} denotes past feature values. The model predicts the next T steps, capturing temporal dependencies. Figure 3 visualizes this process.

Figure 3. Data preprocessing in DynaSched-Net: (a) Normalized task data; (b) Time-series transformation.

 doi:10.20944/preprints202506.0129.v1

3.4.3. Feature Engineering for Resource Utilization

To enhance scheduling decisions, we compute expert-wise resource features, including current load and remaining capacity. Expert i's load is:

$$\text{load}_i = \sum_{j=1}^{N} \mathbb{I}(task_j \in \text{expert}_i) \cdot \text{task\_duration}_j \tag{16}$$

and remaining resources:

$$\text{remaining\_resources}_i = \text{total\_capacity}_i - \text{load}_i \tag{17}$$

These features inform the RL agent about real-time system utilization.

4. Evaluation Metrics

We assess the performance of **DynaSched-Net** using four key metrics:

4.1. Evaluation Metrics Description

1. Standard Deviation of Expert Load: Measures load balance across experts. Lower values indicate better distribution:

$$\sigma_{\text{load}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(L_i - \mu_L)^2} \tag{18}$$

2. Average Response Timeout: Captures the average delay beyond maximum response time:

$$T_{\text{avg}} = \frac{1}{N}\sum_{i=1}^{N}\max(0, T_i - R_i) \tag{19}$$

3. Average Processing Efficiency: Evaluates resource use per task:

$$E_{\text{avg}} = \frac{1}{N}\sum_{i=1}^{N}\frac{P_i}{T_i} \tag{20}$$

4. Resource Utilization Rate: Indicates overall resource usage percentage:

$$U_{\text{rate}} = \frac{\sum_{i=1}^{N} P_i}{\text{Total Available Resources}} \times 100 \tag{21}$$

5. Experiment Results

The results of the ablation study are summarized in Table 1, which shows that **DynaSched-Net** outperforms all baseline models in terms of load balancing, response time, and resource utilization. The ablation study confirms that both the RL agent and the prediction network contribute significantly to the overall performance.

Table 1. Performance Comparison and Ablation Study Results

Model	Std. Dev. of Load	Avg. Response Timeout	Avg. Efficiency	Resource Utilization
FCFS	15.2	35.4	0.78	70%
RR	12.5	28.3	0.82	75%
Min-Min	10.1	22.1	0.86	80%
DynaSched-Net	**8.2**	**19.7**	**0.92**	**90%**
RL only	9.3	22.5	0.85	85%
Prediction only	11.8	27.3	0.80	78%

 doi:10.20944/preprints202506.0129.v1

6. Conclusion

In this work, we introduced **DynaSched-Net**, a hybrid model combining reinforcement learning and prediction networks for cloud resource allocation and load balancing. Our experiments demonstrate that **DynaSched-Net** outperforms traditional scheduling algorithms and achieves optimal performance in all key metrics. The ablation study highlights the importance of both the RL agent and the prediction network in ensuring system efficiency and stability.

References

1. Wang, L.; Wu, J.; Gao, Y.; Zhang, J. Deep reinforcement learning based resource allocation for cloud native wireless network. *arXiv preprint arXiv:2305.06249* **2023**.
2. Rossi, A.; Visentin, A.; Carraro, D.; Prestwich, S.; Brown, K.N. Forecasting workload in cloud computing: towards uncertainty-aware predictions and transfer learning. *Cluster Computing* **2025**, *28*, 258.
3. Arbat, S.; Jayakumar, V.K.; Lee, J.; Wang, W.; Kim, I.K. Wasserstein adversarial transformer for cloud workload prediction. In Proceedings of the Proceedings of the AAAI Conference on Artificial Intelligence, 2022, Vol. 36, pp. 12433–12439.
4. Jin, T. Integrated machine learning for enhanced supply chain risk prediction. In Proceedings of the Proceedings of the 2024 8th International Conference on Electronic Information Technology and Computer Engineering, 2024, pp. 1254–1259.
5. Wang, E. Hybrid FM-GCN-Attention Model for Personalized Recommendation. In Proceedings of the 2025 International Conference on Electrical Automation and Artificial Intelligence (ICEAAI). IEEE, 2025, pp. 1307–1310.
6. Chen, X. Coarse-to-Fine Multi-View 3D Reconstruction with SLAM Optimization and Transformer-Based Matching. In Proceedings of the 2024 International Conference on Image Processing, Computer Vision and Machine Learning (ICICML). IEEE, 2024, pp. 855–859.
7. Zhou, G.; Tian, W.; Buyya, R.; Xue, R.; Song, L. Deep reinforcement learning-based methods for resource scheduling in cloud computing: A review and future directions. *Artificial Intelligence Review* **2024**, *57*, 124.
8. Gu, Y.; Liu, Z.; Dai, S.; Liu, C.; Wang, Y.; Wang, S.; Theodoropoulos, G.; Cheng, L. Deep Reinforcement Learning for Job Scheduling and Resource Management in Cloud Computing: An Algorithm-Level Review. *arXiv preprint arXiv:2501.01007* **2025**.
9. Zhao, X.; Wu, C. Large-scale machine learning cluster scheduling via multi-agent graph reinforcement learning. *IEEE Transactions on Network and Service Management* **2021**, *19*, 4962–4974.