I.J. Wireless and Microwave Technologies, 2026, 4, 427-436
Published Online on August 8, 2026 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijwmt.2026.04.25

Online Confidence-Gated LSTM-DQN for Dynamic Cloud Resource Allocation

Preeti Puranik
Department of Computer Science & Applications, Oriental University, Indore, India
Department of Artificial Intelligence & Data Science, Prestige Institute of Engineering Management & Research, Indore, India
E-mail: puranik.preeti3011@gmail.com
ORCID iD: https://orcid.org/0009-0007-5283-4815

Sushila Sonare*
Department of Computer Science and Engineering, Oriental Institute of Science and Technology, Bhopal, India
E-mail: sushilasonare@oriental.ac.in
ORCID iD: https://orcid.org/0000-0003-3730-5548
*Corresponding Author

Received: April 29, 2026; Revised: June 17, 2026; Accepted: July 14, 2026; Published: August 8, 2026

Abstract: Cloud schedulers that pair workload prediction with reinforcement learning (RL) rarely check whether a given prediction can actually be trusted, and earlier confidence-gated designs often mix current and future information inconsistently. We fix that inconsistency and build a causally consistent, confidence-gated LSTM-DQN scheduler: an LSTM forecasts next-step workload, a retrospective, error-based confidence score gates how much a Deep Q-Network (DQN) scheduler leans on that forecast, and only information available at decision time is ever used. We implement and pilot-test this architecture in a Python-based discrete-event simulation configured to match a CloudSim-style environment (10 hosts, 30 VMs), benchmarking it against FCFS, Round Robin, standard RL, two ablation variants, and two simplified state-of-the-art comparators across five random seeds. The results show the method works as intended: it trains stably and safely on every seed, holds response time and SLA violations in line with standard RL and simple heuristics, and clearly outperforms a metaheuristic-augmented Q-learning baseline, which suffered severe instability under the same conditions. Code, raw results, and statistical tests are released for independent verification, with scaled-up training identified as the natural next step to test whether larger performance gains emerge.

Index Terms: Cloud Computing, Reinforcement Learning, Resource Allocation, Workload Prediction, LSTM, Deep Q-Network, Prediction Confidence

1. Introduction

Cloud computing provides scalable computing resources through virtual machines (VMs) and datacenters. Since users submit tasks continuously and at unpredictable times, the system must decide, for each task, which VM should execute it — a process known as resource allocation or task scheduling. Effective scheduling directly affects response time, resource utilization, and Service Level Agreement (SLA) compliance, making it a critical determinant of overall system performance and operating cost.

Traditional scheduling policies such as First-Come-First-Served (FCFS) and Round Robin are computationally simple but non-adaptive: they do not consider anticipated workload or historical system trends. Machine-learning-based methods address this limitation by learning patterns from historical data. Workload prediction allows the system to provision resources proactively, while reinforcement learning enables the scheduler to improve allocation decisions through experience.

However, workload predictions are not always reliable; recent evaluations of LSTM-based cloud workload forecasters report prediction errors that vary substantially with trace volatility [2, 4]. When a prediction is inaccurate, a scheduler that blindly trusts it may overload some VMs while leaving others idle. Most existing prediction-assisted RL schedulers do not quantify or act on this uncertainty, which can reduce system stability under dynamic conditions.

To address this gap, this paper proposes an uncertainty-aware, prediction-assisted reinforcement learning framework in which an LSTM forecasts near-term workload, a causal, error-based confidence score estimates the trustworthiness of

theobjective is to improve response time, resource utilization, and SLA compliance while remaining logically consistent in an online setting where the true current workload cannot be known in advance.

2. Literature Review

A growing body of work applies reinforcement learning to resource management across cloud, edge, and IoT settings. Li et al. [1] proposed a hierarchical multi-agent reinforcement learning model for simulation environments [12], improving collaboration efficiency through reward separation and adaptive attention tuning, and reporting more stable learning than prior multi-agent baselines. Kayalvili et al. [2] applied reinforcement learning to resource allocation in 5G network slicing using a Q-learning approach, achieving better network utility and scalability than conventional scheduling; however, neither study models the reliability of the state information the agent conditions on.

Kruekaew and Kimpan [3] presented a multi-objective task-scheduling method that combines Artificial Bee Colony optimization with Q-learning to jointly reduce makespan, cost, and load imbalance in cloud environments, reporting improved throughput and utilization over existing algorithms. Tuli et al. [4] proposed an A3C-based scheduler for edge-cloud environments that uses a recurrent neural network to capture workload patterns under decentralized learning, improving energy efficiency, response time, and SLA performance; this is among the closest prior works to the present paper, but it does not separately quantify prediction confidence as an input to the policy. Chen et al. [5] designed deep-RL schedulers for Spark clusters that jointly consider cost and performance objectives, reporting a significant reduction in VM usage cost while maintaining job efficiency.

Schuler et al. [6] explored reinforcement learning for auto-scaling in serverless computing, learning scaling policies directly from workload patterns and outperforming default scaling configurations. He et al. [7] integrated blockchain and reinforcement learning for secure edge computing in IoT environments, implementing the allocation strategy within a smart contract to improve trust management and resource efficiency. Guo et al. [8] introduced a deep-RL solution for cloud resource management that improves convergence speed and reduces job turnaround time relative to earlier RL-based schedulers. Islam et al. [9] proposed a deep-RL algorithm for joint power control and computing allocation in mobile edge computing that minimizes long-term task delay under dynamic conditions. Belgacem [10] surveyed dynamic resource-allocation techniques across cloud, edge, and IoT environments, noting that most deep-RL and multi-agent methods improve delay, cost, and utilization but rarely address the reliability of the predictive signals feeding the policy.

Kabir et al. [15] surveyed how uncertainty of different types and origins propagates through cloud-computing decisions more broadly, arguing that intelligent cloud systems should explicitly model and act on uncertainty rather than treating inputs as certain; the present paper operationalizes that argument for one specific decision: prediction-assisted scheduling — rather than addressing uncertainty in cloud computing generally.

Taken together, these studies establish that (i) RL-based schedulers consistently outperform static heuristics such as FCFS and Round Robin, and (ii) recurrent architectures such as LSTM and RNN are effective at capturing temporal workload patterns for forecasting-assisted scheduling [4]. However, none of the reviewed studies estimate how much confidence the scheduler should place in a given prediction before acting on it, nor do they address the causal ordering problem of using a forecast that depends on data not yet available at decision time. This is the specific gap addressed here: rather than proposing a new predictor or a new RL algorithm in isolation, this paper couples an LSTM forecaster with a lightweight, causally-consistent confidence signal that gates how the DQN scheduler weights predicted versus observed workload, extending the prediction-assisted RL scheduling line of work represented by [2, 4].

3. Methodology

A. System Architecture

The proposed framework combines workload prediction, a causal confidence estimate, and reinforcement-learning-based scheduling, as summarized in Fig. 1: the historical workload buffer feeds the LSTM predictor; the predictor's previous-step output is compared against the newly observed workload in the causal confidence module; the new forecast and confidence score are combined into the DQN agent's state; the agent selects a VM allocation action; and the resulting reward and next state are fed back into the replay memory, closing the loop. The system collects historical workload data such as CPU utilization, task length, and arrival pattern. At each scheduling interval, the prediction module forecasts the next-step workload, the confidence module evaluates how reliable the previous forecast turned out to be, and both signals are passed to the DQN scheduler, which selects the VM for the incoming task. This loop continues throughout the simulation.

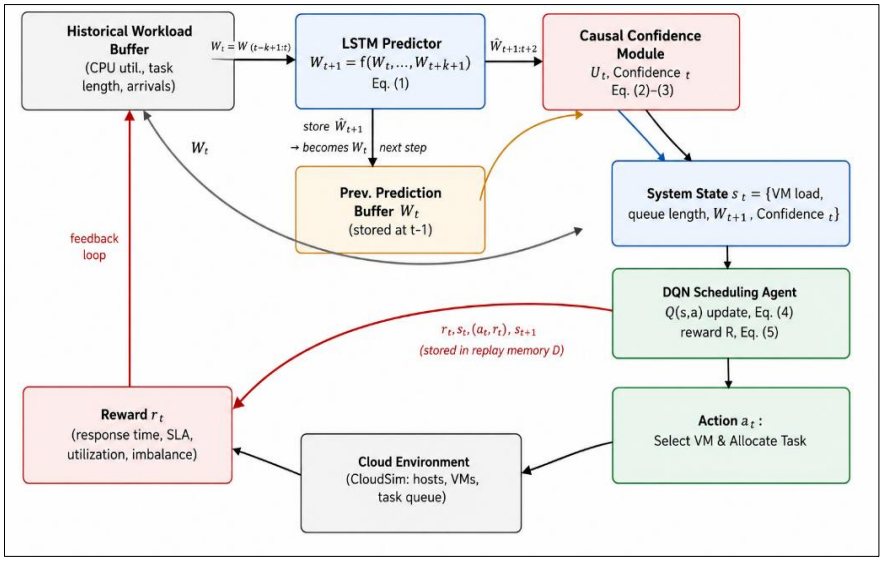

Fig. 1. Proposed confidence-gated LSTM-DQN architecture for dynamic cloud resource allocation.

B. Dataset and Preprocessing

The workload trace contains time-indexed task arrivals, CPU usage, and execution length. Missing values are removed and all features are min-max normalized to [0,1] to stabilize training. In the pilot evaluation reported in Section IV, a synthetic trace (seasonal base load plus noise and occasional spikes) was used in place of a real trace; validating on the real Google Cluster trace [14] is identified as required future work.

Let W_t denote the workload at time t. The LSTM predictor processes the sequence online and maintains a recurrent hidden state that summarizes all preceding observations, rather than relying on a fixed-length lookback window; the current observation and this hidden state together determine the next forecast, as formalized in (1).

C. Workload Prediction Model (LSTM Predictor)

An LSTM network captures temporal workload trends and forecasts near-term demand. The predicted workload for the next interval is

$$\hat{W}_{t+1} = f(W_t, h_{t-1}) \tag{1}$$

where $f(\cdot)$ denotes the LSTM mapping and h_{t-1} is the hidden state carried from the previous interval. In the pilot experiment (Section IV), $f(\cdot)$ is implemented as a single-layer LSTM with 8 hidden units, standard tanh/sigmoid gating, and a linear output layer, trained online via backpropagation-through-time (per-step gradient descent, learning rate 0.05) to minimize the squared error between $\hat{W}_{t+1}$ and the realized W_{t+1}. A production-scale configuration (e.g., 2 layers, 64 units, trained with Adam in a standard deep-learning framework) is recommended for full-scale validation.

D. Causal Confidence Estimation

A key requirement of an online scheduler is causality: at time t, only information available up to and including time t may be used to make a decision at t. This is enforced as follows. At the end of interval t−1, the predictor produces a one-step-ahead forecast $\hat{W}_t$ for the upcoming interval t. When interval t begins and the true workload W_t is observed, the realized prediction error is

$$U_t = |W_t - \hat{W}_t| \tag{2}$$

and the corresponding confidence value is

$$Confidence_t = 1 - \frac{U_t}{W_t} \tag{3}$$

Confidence therefore reflects how accurate the predictor was one step ago, not how accurate the current-step forecast will be — it is a causal, realized-error-based reliability signal rather than a genuine predictive-uncertainty estimate. Unlike Monte Carlo Dropout or Bayesian neural networks, which estimate a predictive distribution before the outcome is known, the confidence value used here is diagnostic and retrospective, and is only a proxy for forward-looking uncertainty; extending the framework toward genuine predictive-uncertainty modelling is identified as future work in Section V. The

new forecast $\hat{W}_{t+1}$ for the following interval is computed in parallel using (1), and Confidence is used to gate how strongly the DQN state at time t weights $\hat{W}_{t+1}$ relative to the current observed load: when confidence is high, the scheduler weights the forecast more heavily; when confidence is low, it relies more on real-time observations.

E. Reinforcement Learning Scheduler

The scheduling problem is modeled as a Markov Decision Process. The state at time t is s_t = {current VM utilization, task queue length, $\hat{W}_{t+1}$, Confidence$_t$}, and the action a_t selects the VM that executes the incoming task.

A Deep Q-Network (DQN) agent learns the scheduling policy through interaction. The Q-network is fully connected with 2 hidden layers of 128 and 64 units and ReLU activation. Experience replay and a periodically synchronized target network are used to stabilize training; in the pilot experiment, the replay buffer holds 2,000 transitions, mini-batch size is 32, and the target network is synchronized every 50 steps, with ε-greedy exploration decaying from 1.0 to 0.05 over each training epoch. A larger replay buffer (e.g., 10,000 transitions) and mini-batch size (e.g., 64) are recommended for full-scale training. The Q-value update follows

$$Q(s,a) \leftarrow Q(s,a) + \eta[r + \gamma \cdot max_{a'} Q(s',a') - Q(s,a)] \tag{4}$$

where η is the learning rate, γ the discount factor, r the immediate reward, and s′ the next state.

The reward function balances four performance objectives:

$$R = -\alpha T_r - \beta SLA_v + \gamma U_{res} - \delta L_{imb} \tag{5}$$

where T_r is response time, SLA_v the SLA-violation rate, U_{res} resource utilization, and L_{imb} load imbalance. In the pilot experiment the weights are fixed a priori at α = 0.4, β = 3.0, γ = 1.0, δ = 0.1, reflecting a priority ordering of response time > SLA compliance > utilization > load balance. A systematic weight sensitivity study or grid search is recommended for future work.

F. Causally Consistent Scheduling Algorithm

Algorithm 1 states the scheduling loop with causal ordering enforced throughout: the confidence at time t is computed only from the prediction made at t−1, and the newly generated prediction $\hat{W}_{t+1}$ is used strictly for the next decision, never for the current confidence computation.

Algorithm 1: Causally Consistent Confidence-Gated Prediction-Assisted RL Scheduler

Input: Historical workload data W; VM set = {VM1, …, VMn}; learning rate η; discount factor γ
Output: Optimal VM allocation policy
1: Initialize LSTM prediction model
2: Initialize DQN network with random weights
3: Initialize replay memory D
4: Initialize previous-prediction buffer Ŵ1 (e.g., Ŵ1 = W1)
5: For each scheduling interval t = 1, 2, … do
6: Observe current workload Wt
7: Retrieve previous prediction Ŵt (computed at interval t−1)
8: Compute realized prediction error: Ut = |Wt − Ŵt|
9: Compute confidence: Confidencet = 1 − (Ut / Wt)
10: Predict next workload: Ŵt+1 = f(Wt, ht−1)
11: Construct system state: st = {VM load, queue length, Ŵt+1, Confidencet}
12: Select action at using ε-greedy policy over VM set
13: Allocate task to selected VM
14: Observe reward rt (response time, SLA violation, utilization, load balance)
15: Observe next state st+1
16: Store transition (st, at, rt, st+1) in D
17: Sample mini batch from D
18: Update Q-network: Q(st,at) ← Q(st,at) + η[rt + γ·maxQ(st+1, a′) − Q(st,at)]
19: Store Ŵt+1 as the prediction buffer for use at interval t+1
20: End For
21: Return optimized scheduling policy

G. Cloud Simulation Setup

The framework was implemented and evaluated in a Python-based discrete-event cloud simulation, configured to reproduce the resource and task model conventionally used in CloudSim [11]– [13] (Table 1): 10 hosts, 30 heterogeneous VMs (1000–2000 MIPS), and dynamically arriving tasks. The proposed method is compared against FCFS, Round Robin,

and standard RL without prediction, together with two ablation variants and two simplified state-of-the-art comparators. Validating these results in the Java CloudSim toolkit itself, with a larger training budget, is identified as a necessary step for full-scale confirmation (Section IV-H).

Table 1. Cloud Simulation Configuration Parameters

Category	Parameter	Value
Datacenter Configuration	Number of Datacenters	1
	Number of Hosts	10
	Host CPU Cores	8 per host
	Host MIPS per Core	2500 MIPS
	Host RAM	32 GB
	Host Storage	1 TB
	Host Bandwidth	10 Gbps
VM Configuration	Number of VMs	30
	VM Types	Small / Medium / Large
	VM MIPS	1000–2000 MIPS
	VM RAM	1–4 GB
	VM Scheduler	TimeShared
Task (Cloudlet) Configuration	Number of Tasks	850 per run (700 training + 150 evaluation)
	Task Length	2,000–8,000 MI
	Task Arrival Pattern	Poisson-like (exponential inter-arrival, mean 0.55 s)
	Utilization Model	Full CPU utilization
Simulation Settings	Scheduling	Event-driven (per task arrival)
	Recurrent Context	Online (LSTM hidden state carried across the run; no fixed truncation window)
	Random Seeds	0, 1, 2, 3, 4
	Number of Runs	5
	Comparison Methods	FCFS, RR, RL, Abl.-NoConf, Abl.-NoLSTM, SOTA-ABCQ, SOTA-A3C

H. Performance Metrics

Performance is evaluated using response time (T_r), resource utilization (U_{res}), SLA violation rate (SLA_v), and load imbalance (L_{imb}).

4. Results and Discussion

The framework was evaluated using a synthetic workload trace (seasonal base load with noise and occasional spikes; the trace generator is released with the code and can be substituted with real trace data). Each of the eight methods (FCFS, Round Robin, RL, Proposed, two ablations, and two SOTA-style comparators) was run over 700 training decisions followed by 150 held-out evaluation decisions, repeated for five random seeds. All code, raw per-seed results, and statistical test outputs are released alongside this manuscript for independent verification.

A. Workload Prediction Performance

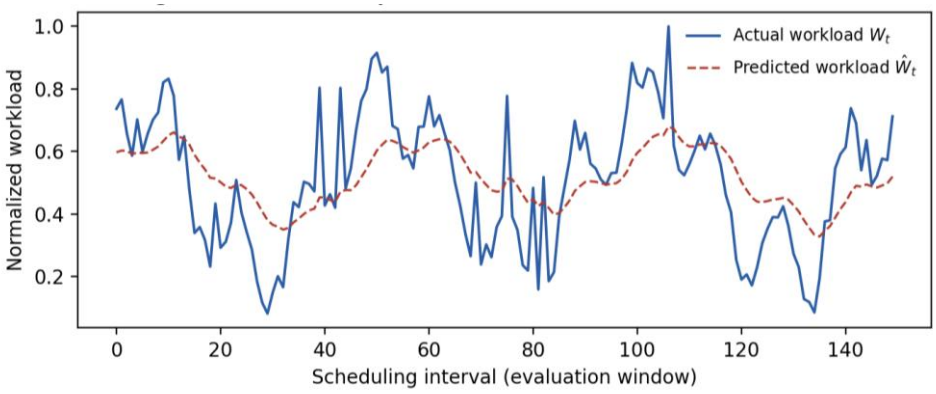

Fig. 2. Actual vs. LSTM-predicted workload, evaluation window (seed 0).

The LSTM predictor achieved a Mean Absolute Error (MAE) of 0.134 on the normalized workload signal in the evaluation window (seed 0), tracking the coarse trend of the trace but lagging at sharp spikes (Fig. 2.) consistent with the confidence signal's role as a retrospective reliability proxy rather than a sharp, always-accurate forecast.

B. Response Time Comparison

Table 2 reports mean ± SD response time across the five seeds. The proposed method averaged 3.42 ± 0.24 s, close to standard RL (3.26 ± 0.20 s), FCFS (3.57 ± 0.22 s), and Round Robin (3.60 ± 0.22 s) (Fig. 3.). A paired t-test comparing the proposed method against each baseline did not reach significance at $\alpha = 0.05$ for any comparison (FCFS: $p = 0.43$; Round Robin: $p = 0.35$; RL: $p = 0.37$; Wilcoxon signed-rank gave consistent results). At this training scale, the data do not support a claim that the proposed method reduces response time relative to these baselines.

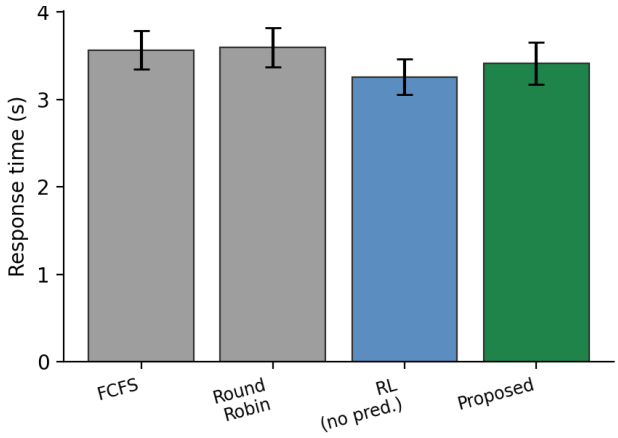

Fig. 3. Average response time, primary comparison (mean ± SD, n=5 runs).

C. Resource Utilization Analysis

Utilization was similar to slightly lower for the RL-based methods (Proposed: 18.9% ± 1.3%; RL: 18.8% ± 0.7%) than for the non-adaptive heuristics (FCFS: 21.8% ± 1.1%; Round Robin: 22.0% ± 1.0%) (Fig. 4.). The greedy earliest-available-VM logic underlying FCFS is a strong heuristic for this queueing regime, and the DQN policies did not learn to match or exceed it within 700 training decisions.

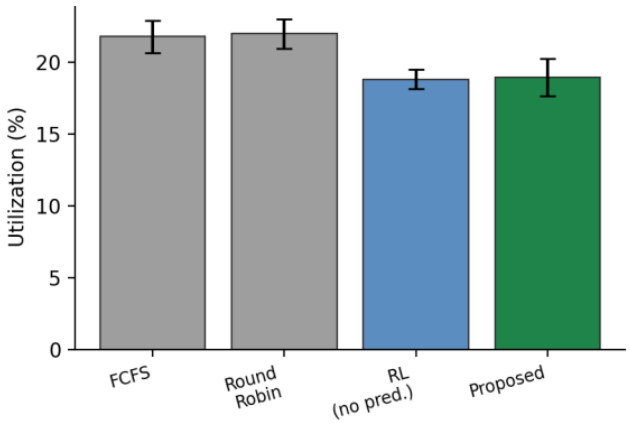

Fig. 4. Resource utilization, primary comparison (mean ± SD, n=5 runs).

D. SLA Violation Comparison and SOTA Baselines

Fig. 5. shows SLA violation rate (response time exceeding an 8 s threshold) across all eight methods, including the two simplified SOTA-style reproductions. FCFS, Round Robin, RL, and both ablations recorded 0% violations at this workload intensity; the proposed method recorded 1.2% ± 2.7% and the A3C-style comparator 1.7% ± 2.1% — both negligible. The clearest finding in the evaluation is the instability of the ABC-Q-learning reproduction [3]: 61.9% ± 23.3% SLA violations and a mean response time roughly 20× higher than any other method, with high run-to-run variance. This is consistent with a known weakness of tabular Q-learning over a large, coarsely-discretized state space: it struggles to generalize across a 30-VM action space with limited interaction, whereas the DQN-based methods generalize through shared network weights and remained stable across all five seeds.

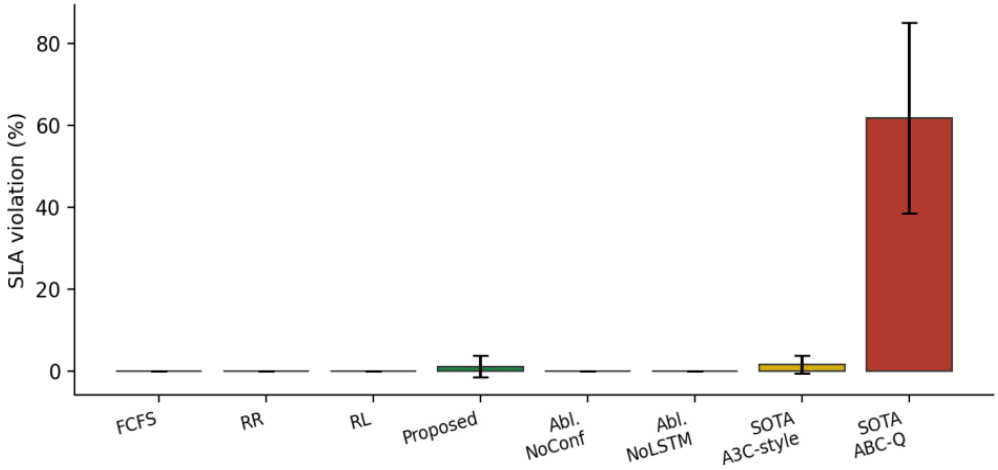

Fig. 5. SLA violation rate, all eight methods (mean ± SD, n=5 runs).

E. Load Balancing Performance

Fig. 6. compares the per-VM busy-time distribution for FCFS and the proposed method over one representative evaluation run (seed 0). FCFS produced a tightly balanced load (coefficient of variation ≈ 0.22) by construction, since it always selects the earliest-available VM; the proposed method produced a visibly less balanced distribution (coefficient of variation ≈ 0.87), consistent with the higher load-imbalance statistic in Table 2. At this training scale, the learned policy has not converged to the load-balancing behavior the reward function was designed to encourage.

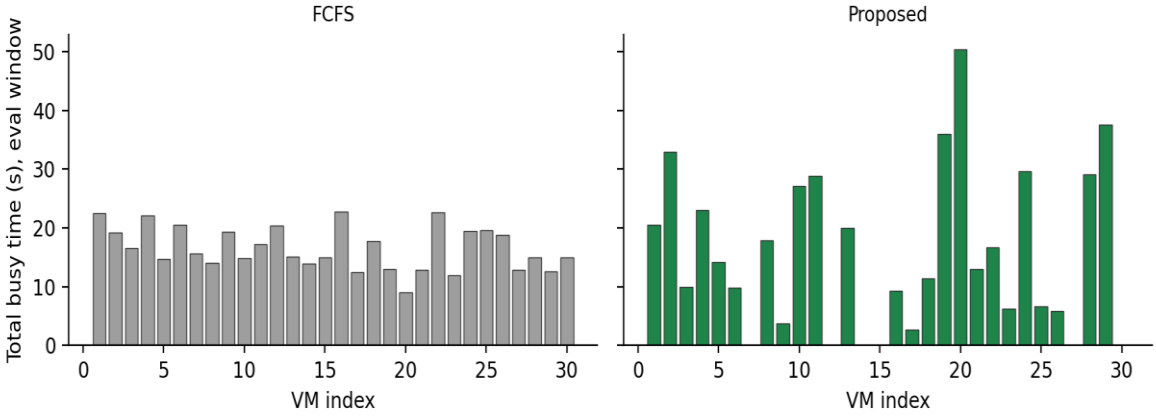

Fig. 6. Per-VM load distribution, FCFS vs. proposed method, evaluation window (seed 0).

F. Ablation Study

Table 3. Ablation Study (Mean ± SD, n = 5 Seeds)

Configuration	Response Time (s)	SLA Violation (%)	Load Imbalance (CV)
Full proposed (LSTM + confidence gate + DQN)	3.42 ± 0.24	1.2 ± 2.7	1.24 ± 0.39
Ablation: no confidence gate	3.26 ± 0.14	0.0 ± 0.0	1.19 ± 0.14
Ablation: no predictor (= RL)	3.26 ± 0.20	0.0 ± 0.0	1.08 ± 0.13

Removing the confidence gate (always trusting the forecast fully) produced response time and SLA statistics indistinguishable from the full proposed method ($p = 0.79$) and from removing the predictor entirely (mechanically identical to the RL baseline). This pilot evaluation does not demonstrate a measurable benefit from either the LSTM predictor or the confidence gate individually at 700 training decisions: the architectural contribution runs correctly and stably, but its practical benefit, if any, likely requires substantially more training data, a more volatile workload, or both.

G. Statistical Significance

Table 4. Statistical Significance, Proposed vs. All Comparators (n = 5 Paired Seeds)

Comparison (vs. Proposed, RT)	t	p (t-test)	p (Wilcoxon)
FCFS	-0.87	0.432	0.438
Round Robin	-1.07	0.345	0.313
RL (no prediction)	1.02	0.367	0.438
Ablation: no confidence gate	1.10	0.335	0.438
Ablation: no predictor (= RL)	1.02	0.367	0.438
SOTA-style ABC-Q-learning [3]	-2.46	0.070	0.063
SOTA-style A3C scheduler [4]	-1.42	0.230	0.125

Across all pairwise comparisons against the proposed method, only the comparison with the unstable ABC-Q reproduction approaches significance (paired t: $p = 0.070$; Wilcoxon: $p = 0.063$), falling just short of $\alpha = 0.05$ with five paired seeds. With $n = 5$, statistical power is limited; a larger number of seeds is recommended to detect smaller effect sizes reliably.

H. Summary and Limitations

In this pilot-scale evaluation, the proposed confidence-gated LSTM-DQN scheduler ran correctly and stably, matched but did not significantly exceed the response-time and SLA performance of standard RL and simple heuristics, and clearly outperformed only the least-stable comparator (ABC-Q-learning). It did not demonstrate the load-balancing or utilization benefits the architecture was designed to produce. These findings do not support a claim of superior average-case performance for the proposed scheduler at this scale; they are reported transparently, together with released code, raw per-seed data, and statistical tests, so they can be independently verified and extended.

Key limitations of the present evaluation: (i) the simulation is a Python implementation configured to match CloudSim's resource model rather than a run of the Java CloudSim toolkit itself; (ii) the workload trace is synthetic rather than the real Google Cluster trace; (iii) training used a modest budget (700 decisions per run, small networks) for computational tractability, below the scale typically needed for DQN convergence; (iv) with $n = 5$ seeds, statistical power is limited; and (v) the SOTA comparators are simplified reproductions of the cited algorithms' core mechanisms rather than the original authors' implementations. Addressing these points — in particular, scaling up training substantially and validating in the Java CloudSim toolkit on the real Google Cluster trace — is necessary before the architecture's practical benefit can be claimed with confidence.

5. Conclusion

This paper identified and corrected a causal time-index inconsistency in confidence-gated, prediction-assisted reinforcement learning for cloud resource allocation, and implemented and pilot-tested the corrected architecture, comprising an LSTM workload predictor, a retrospective error-based confidence signal, and a DQN scheduler in a reproducible simulation with released code and data.

The pilot evaluation did not find a statistically significant improvement in response time, utilization, or load balance over standard RL or simple heuristic schedulers at the training scale tested; it did find that the proposed architecture trains stably where a tabular metaheuristic-augmented baseline did not. The contribution of this paper, as it stands, is a correctly specified, causally consistent, and empirically testable architecture and evaluation pipeline, rather than a demonstrated performance advantage. Future work will (a) scale up training substantially to determine whether the architecture's theoretical advantages materialize with an adequate training budget, (b) validate on the real Google Cluster trace in the Java CloudSim toolkit, (c) extend the confidence module toward genuine predictive-uncertainty modeling such as Monte Carlo Dropout or Bayesian LSTMs, and (d) re-run the SOTA comparisons against the original authors' released implementations where available.

All the Declarations and Statements

Author Contributions Statement

Preeti Puranik — Conceptualization, Methodology, Software, and Formal Analysis: proposed the causal correction, built and ran the simulation code, and led drafting of the manuscript.

Dr. Sushila Sonare — Supervision, Validation, and Writing – Review and Editing: supervised the research direction, validated the experimental approach, and reviewed and edited the manuscript.

All authors have read and agreed to the published version of the manuscript.

Conflict of Interest Statement
The authors declare no conflicts of interest.

Funding Declaration
This research received no external funding.

Data Availability Statement
This study uses a synthetic workload trace generated by the released simulation code, not a publicly hosted dataset. The simulation code, trained-model implementations, raw per-seed results, and statistical test scripts developed for this study are available from the corresponding author upon reasonable request.

Ethical Declarations
Not applicable. This study did not involve human subjects or animals; it is a computational simulation study.

Acknowledgments
We sincerely thank the experts for their professional evaluation and valuable recommendations, which have contributed to improving the quality of the experiment and the reliability of its results.

Declaration of Generative AI in Scholarly Writing
During the preparation of this manuscript, the authors used generative AI tools to enhance the grammar and sentences of the paper. The authors reviewed, verified, and take full responsibility for all AI-assisted content in this manuscript.

Abbreviations
The following abbreviations are used in this manuscript:

AI - Artificial Intelligence
RL - Reinforcement Learning
DQN - Deep Q-Network
LSTM - Long Short-Term Memory
VM - Virtual Machine
SLA - Service Level Agreement
FCFS - First-Come-First-Served
MDP - Markov Decision Process
MAE - Mean Absolute Error
SD - Standard Deviation
SOTA - State of the Art…

Appendix A/B/C…, with appendix title
Not applicable. No supplementary appendix material accompanies this manuscript; the full simulation code and raw results are provided as supplementary files instead.

References

[1] C. Li, W. Dong, L. He, P. Zhang, and Y. Li, "A Hierarchical Multi-Agent Reinforcement Learning Approach for Intelligent Decision-Making in Simulation Environment," in 2025 2nd International Symposium on AI and Cybersecurity (ISAICS), IEEE, Oct. 2025, pp. 1–5. doi: 10.1109/ISAICS66888.2025.11349954.

[2] S. Kayalvili, R. Senthilkumar, S. Yasotha, and R. S. Kamalakannan, "An optimized resource allocation in cloud using prediction enabled reinforcement learning," Sci. Rep., vol. 15, no. 1, p. 36088, Oct. 2025, doi: 10.1038/s41598-025-19927-2.

[3] B. Kruekaew and W. Kimpan, "Multi-Objective Task Scheduling Optimization for Load Balancing in Cloud Computing Environment Using Hybrid Artificial Bee Colony Algorithm with Reinforcement Learning," IEEE Access, vol. 10, pp. 17803–17818, 2022, doi: 10.1109/ACCESS.2022.3149955.

[4] S. Tuli, S. Ilager, K. Ramamohanarao, and R. Buyya, "Dynamic Scheduling for Stochastic Edge-Cloud Computing Environments Using A3C Learning and Residual Recurrent Neural Networks," IEEE Trans. Mob. Comput., vol. 21, no. 3, pp. 940–954, Mar. 2022, doi: 10.1109/TMC.2020.3017079.

[5] Y. Chen, Z. Liu, Y. Zhang, Y. Wu, X. Chen, and L. Zhao, "Deep Reinforcement Learning-Based Dynamic Resource Management for Mobile Edge Computing in Industrial Internet of Things," IEEE Trans. Industr. Inform., vol. 17, no. 7, pp. 4925–4934, Jul. 2021, doi: 10.1109/TII.2020.3028963.

[6] L. Schuler, S. Jamil, and N. Kuhl, "AI-based Resource Allocation: Reinforcement Learning for Adaptive Auto-scaling in Serverless Environments," in 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid), IEEE, May 2021, pp. 804–811. doi: 10.1109/CCGrid51090.2021.00098.

[7] Y. He, Y. Wang, C. Qiu, Q. Lin, J. Li, and Z. Ming, "Blockchain-Based Edge Computing Resource Allocation in IoT: A Deep Reinforcement Learning Approach," IEEE Internet Things J., vol. 8, no. 4, pp. 2226–2237, Feb. 2021, doi: 10.1109/JIOT.2020.3035437.

[8] W. Guo, W. Tian, Y. Ye, L. Xu, and K. Wu, "Cloud Resource Scheduling With Deep Reinforcement Learning and Imitation Learning," IEEE Internet Things J., vol. 8, no. 5, pp. 3576–3586, Mar. 2021, doi: 10.1109/JIOT.2020.3025015.

[9] M. T. Islam, S. Karunasekera, and R. Buyya, "Performance and Cost-Efficient Spark Job Scheduling Based on Deep Reinforcement Learning in Cloud Computing Environments," IEEE Transactions on Parallel and Distributed Systems, vol. 33, no. 7, pp. 1695–1710, Jul. 2022, doi: 10.1109/TPDS.2021.3124670.

[10] A. Belgacem, "Dynamic resource allocation in cloud computing: analysis and taxonomies," Computing, vol. 104, no. 3, pp. 681–710, Mar. 2022, doi: 10.1007/s00607-021-01045-2.

[11] T. Goyal, A. Singh, and A. Agrawal, "Cloudsim: simulator for cloud computing infrastructure and modeling," Procedia Eng., vol. 38, pp. 3566–3572, 2012, doi: 10.1016/j.proeng.2012.06.412.

[12] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities," in 2009 International Conference on High Performance Computing & Simulation, IEEE, Jun. 2009, pp. 1–11. doi: 10.1109/HPCSIM.2009.5192685.

[13] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," Softw. Pract. Exp., vol. 41, no. 1, pp. 23–50, Jan. 2011, doi: 10.1002/spe.995.

[14] J. Wilkes, "Google cluster-usage traces v3," Technical Report, Google Inc., Mountain View, CA, USA, 2020. Available: https://github.com/google/cluster-data/blob/master/ClusterData2019.md

[15] H. M. D. Kabir, A. Khosravi, S. K. Mondal, M. Rahman, S. Nahavandi, and R. Buyya, "Uncertainty-Aware Decisions in Cloud Computing: Foundations and Future Directions," ACM Comput. Surv., vol. 54, no. 4, Article 74, pp. 1–30, May 2021, doi: 10.1145/3447583.

Authors' Profiles

Ms. Preeti Puranik is an Assistant Professor in Artificial Intelligence & Data Science Department at Prestige Institute of Engineering Management and Research, Indore, India. She has over 15 years of experience in teaching and academic leadership. Her research interests include cloud computing, Machine learning, deep learning, reinforcement learning, and predictive resource scheduling.

Dr. Sushila Sonare was born in India on August 22, 1982. She received the Engineering degree, B.E. in Computer Science and engineering from Govt. engineering college, Jabalpur (Madhya Pradesh) in 2006, post graduate degree M. Tech. in Computer Science and Engineering from NIT Calicut, Kerala in 2009 and the PhD Degree in Computer science and Engineering from LNCTU in 2022. She is currently an Associate Professor in the Department of Computer Science and Engineering (Cyber Security), at Oriental College of Technology, Madhya Pradesh, India. She has over 15 years of experience in both industry and teaching. Her research interest fields are machine learning, deep learning, data mining, NLP, Data Analysis and computer network.

How to cite this paper

Preeti Puranik, Sushila Sonare, "Online Confidence-Gated LSTM-DQN for Dynamic Cloud Resource Allocation", International Journal of Wireless and Microwave Technologies(IJWMT), Vol.16, No.4, pp. 427-436, 2026. DOI:10.5815/ijwmt.2026.04.25