International Journal of Basic and Applied Sciences, 14 (5) (2025) 891-911

International Journal of Basic and Applied Sciences

Website: www.sciencepubco.com/index.php/IJBAS
https://doi.org/10.14419/535sc283
Research paper

DCN-LB: A Deep Learning-Driven Cloud Load Balancing Framework for Efficient Resource Optimization

Damodhar M. [1] *, Dr. Ch. D. V. Subba Rao [2]

[1] *Research Scholar, Department of Computer Science and Engineering, S V U College of Engineering, Tirupati, India*
[2] *Professor, Department of Computer Science and Engineering, S V U College of Engineering, Tirupati, India*
**Corresponding author E-mail: mdr.damodhar@gmail.com*

Received: July 31, 2025, Accepted: September 18, 2025, Published: September 29, 2025

Abstract

Cloud computing is a technology that meets the needs of a vast number of users. Predicting workload and scheduling are often the elements that determine cloud performance. This study addresses the challenge of efficient load balancing in scalable cloud environments, where traditional methods fail under dynamic workloads, leading to resource wastage and performance degradation. To overcome this, an advanced framework integrating Deep Learning (DL) and Reinforcement Learning (RL) is proposed. Simulation data on CPU usage, memory, network traffic, and execution times are collected and preprocessed using mean imputation and Min-Max normalization. A Sliding Window Approach with Multi-Scale Convolutional Bidirectional LSTM (MS-Conv-BiLSTM) is used for time-series feature extraction. An Attention-based LSTM forecasts workload levels, while a lightweight CNN assists in task classification. Adaptive load balancing decisions are optimized using the Double Deep Q-Network (DDQN), aiming to reduce latency and response time while maximizing throughput and resource utilization. Experimental results confirm that the DL-RL framework significantly enhances real-time load balancing performance in cloud environments.

Keywords: *Cloud Computing; Load Balancing; Workload Prediction; Reinforcement Learning.*

1. Introduction

Businesses use cloud computing to control and deploy applications through scalable computing resources available on demand [1]. The main obstacle in cloud computing involves distributing workload between several servers to achieve optimal operational efficiency alongside high availability and low response times [2]. Load balancing defines this distribution process. Load balancing within cloud computing systems directs network traffic exactly so servers handle appropriate workloads to enhance resource efficiency, together with system reliability [3]. Multiple servers participate in load-balancing operations in cloud computing using different distribution algorithms and techniques that handle incoming network traffic or computational tasks [4]. The applied techniques function as static or dynamic systems based on whether they incorporate real-time server load data and response times, and system availability metrics [5].

Different workload needs require specific load balancing algorithms, such as round-robin and least connection, and weighted algorithms [6]. The modern cloud load balancer implements AI alongside ML capabilities to determine upcoming traffic routes by actively modifying resource distribution [7]. Cloud-based load balancing systems offer substantial advantages to users. Application performance increases efficiently because latency decreases while system failures decrease. After all, traffic is distributed efficiently and the system scales automatically according to demand requirements [8]. The optimization of operational expenses alongside resource effectiveness becomes possible through load balancing because it avoids resource wastage and minimizes operational expenses [9]. Cloud service providers AWS, Microsoft Azure, and Google Cloud present their load-balancing solutions, which operate within their cloud infrastructure [10].

The development of cloud-based load balancing technology has experienced major advancements because of advancing Artificial Intelligence (AI) and machine learning (ML), along with edge computing [11]. AI through automation uses predictive load balancing to analyse past traffic data and take automated workload distribution actions that stop performance problems [12]. Real-time optimization becomes achievable through this intelligent load management system that reduces periods of unavailability, together with better application performance. Edge computing enables load balancers to route traffic toward end users so cloud service performance strengthens and latency decreases, particularly when delivering IoT and real-time analytic services [13]. The protection of modern cloud load balancing systems remains essential because cyber threats continue to develop [14]. Modern load-balancing solutions use future security technology to include Distributed Denial of Service (DDoS) attack prevention, together with encrypted data transmission and artificial intelligence, which detects anomalies in real-time [15]. Cloud load balancing solutions currently experience multiple problems because their workload distribution methods are inefficient, which produces both underutilized resources and longer response times. Traditional methods normally become

unable to adapt quickly because they fail to handle immediate changes in workload, along with sudden traffic increases, which leads to delayed service plus functional breakdowns. The protection of system security requires better anomaly detection methods and preemptive defence techniques because DDoS attacks alongside data breaches remain top priorities. For this reason, this research intends to develop a novel DL model for load balancing in a cloud computing environment.

1.1. Background of the study

One of the characteristics of cloud computing is the deployment of applications with scalable and on-demand resources; efficient workload distribution becomes a primary concern. Traditional approaches to load balancing typically do not adapt to dynamic traffic, which manifests as underutilized resources, increased response time, and reduced system reliability. Present-day cloud environments call for intelligent solutions capable of reacting to workload changes instantaneously; their applications include the integration of AI and ML for load balancing to conduct predictive analysis and decision-making autonomously to improve performance. This work attempts to overcome these drawbacks by proposing a DL-based model that will ensure better workload distribution, higher resource utilization, and an increase in cloud computing responsiveness and security.

1.2. Key contributions

- To extract Time-Series feature extraction using Sliding Window Approach for deep feature extraction from workload time-series data to produce better workload predictions and resource deployment, which minimizes energy consumption while increasing cloud system performance.
- To enable workload prediction for better load balancing, an MS-Conv-BiLSTM is proposed to enhance distributed task scheduling and minimum response times, and achieve the lowest possible latency and best throughput.
- A Lightweight CNN is proposed for load balancing, which focuses on reducing the computation overhead. Additionally, a DDQN is proposed for continuous monitoring by realizing real-time workload prediction and classification, and dynamic load balancing for improving system efficiency with better resource utilization.

The literature review highlights several existing works that have addressed similar challenges and provided foundational insights into the topic.

Prior studies on cloud load balancing have made significant contributions, but still exhibit critical limitations that hinder scalability and efficiency. For example, Raja [24] employed static load balancing policies, which fail to adapt under dynamic workload fluctuations. Such rigidity often results in VM overloading and degraded QoS when task arrivals become bursty or heterogeneous. In contrast, DCN-LB leverages a multi-scale CNN–BiLSTM framework with memory-controlled gates, enabling dynamic adaptability to workload variations across multiple time scales.

Similarly, Hayyolalam & Özkasap [16] proposed metaheuristic-based balancing approaches, but their methods incur high energy consumption and increased response times due to iterative optimization overhead. Our DCN-LB design addresses this by integrating a lightweight CNN-based feature extractor, which minimizes computational cost while retaining predictive accuracy. As a result, DCN-LB achieves lower energy usage and faster decision-making compared to metaheuristic counterparts.

Other recent works [17, 18, 21] primarily focus on single-resource optimization (e.g., CPU utilization) without holistically incorporating memory, network, and disk usage. This narrow view leads to imbalanced resource allocation and system bottlenecks. In contrast, DCN-LB introduces a multi-resource feature fusion strategy, ensuring balanced utilization across VMs and PMs while preventing hotspots.

By explicitly overcoming these limitations—static policies, energy overhead, and single-resource bias—DCN-LB establishes itself as a dynamic, resource-aware, and computationally efficient framework suitable for modern cloud environments.

In 2025, Hayyolalam & Özkasap [16] have utilized the higher energy usage and operational expenses resulting from the growing need for different cloud applications because resource management and workload distribution become complex. The proposed CBWO method merges the Black Widow Optimization algorithm with Chaos theory to develop a novel load-balancing solution. Tests indicate that CBWO successfully outcompetes other approaches by providing better performance through a 67% shorter makespan duration accompanied by a 29% decrease in energy utilization. In 2025, Haris & Zubair [17] have suggested that the delivery of internet-based computing and storage services through cloud computing requires efficient load balancing to remain a major challenge. The proposed approach integrates the Battle Royale Optimization technique with Deep Reinforcement Learning to develop the BRDRL hybrid algorithm for improving load balancing. According to CloudSim simulations, BRDRL provides better performance than DRL because it achieves a 3.9% higher throughput while simultaneously improving response time by 15.3%. In 2025, Hegde et al [18] have suggested that Cloud Computing (CC) provides internet-based service delivery, which solves issues related to scheduling and security, and load balancing. This research proposes the Adam-Pufferfish Optimization Algorithm (Adam-POA) for VM task assignment through a round-robin method that improves load balancing. Adam-POA demonstrates a resource availability of 0.880, along with a capacity of 0.915, a load of 0.529, and a reliability of 0.857.

In 2025, Krishna & Vali [19] identified that CC delivers resources through scalable and affordable sharing that needs optimal load balancing strategies to deliver superior performance. The article presents the Meta Reinforcement Learning Driven Hybrid Lyrebird Falcon Optimization (Meta-RHDC) model, which performs dynamic load balancing. - The CloudSim simulation results demonstrate that Meta-RHDC outperforms current approaches by cutting the makespan by up to 30.57% together with energy usage reduction of 42.59% while increasing resource utilization and scalability. In 2025, Roselin & Insulata [20] presented a blockchain implementation of load balancing that presents a decentralized distributed algorithm which optimizes cloud computing workload management. The solution optimizes VM selection procedures while incorporating smart contracts, which enhances resource management and cloud service operation effectiveness. CloudSim simulations confirm that the system achieves better results in terms of makespan and execution time, resource utilization, and throughput.

In 2025, Milan et al [21] discussed the growing carbon emissions from computing servers and devices require energy-efficient solutions due to their high energy consumption. The proposed research develops an innovative task scheduling method that uses artificial bee behavioral principles to optimize load balancing in green cloud computing. CloudSim simulation studies validate that the method successfully increases makespan reductions, together with better energy efficiency and stronger overall performance results.

In 2025, Kumar et al [22] have if Cloud services are currently in a phase of rapid development, load balancing presents the main obstacle to achieving energy efficiency. This research introduces a combined DL-based load balancing technique that incorporates deep embedding clustering (DEC) with a deep Q recurrent neural network (DQRNN). CloudSim simulations produced enhanced performance by using optimal parameters of 0.147 for load distribution, 0.726 for capacity allocation, 0.527 for resource consumption, and 0.895 for success rate achievement. In 2025, Alwhelat et al [23] explored an AI-based resource management system for cloud computing that employs DDPG

protocols linked to reinforcement learning approaches to optimize versatile resource usage. The model develops its policy through real-time feedback when both DL and RL are integrated. The experimental findings support DDPG as a strong candidate for developing flexible cloud load-balancing technology. In 2025, Raja [24] discussed that Cloud computing needs dynamic load balancing because it functions to maximize resource usage and stop servers from becoming congested. An Enhanced Dynamic Balancing Algorithm serves as a non-AI method to distribute loads through server capacity checks, workload monitoring, and system load analysis. The experimental outcome demonstrates two main benefits, which include better system response performance and stability when compared to conventional methods. In 2025, Naik & Madhavi [25] introduced that the educational environment gets better through Cloud Computing since this technology allows dynamic scalability and virtualized use of resources. The research presents the Adam Gazelle Optimization Algorithm (AGOA) as a solution for collaborative e-learning applications that provide course recommendations. The experimental data confirms that AGOA operates with 0.139 predicted load, 0.158 J energy usage, and 10.1 migration cost to enhance cloud system performance.

Prior studies on cloud load balancing have made significant contributions, but still exhibit critical limitations that hinder scalability and efficiency. For example, Raja [24] employed static load balancing policies, which fail to adapt under dynamic workload fluctuations. Such rigidity often results in VM overloading and degraded QoS when task arrivals become bursty or heterogeneous. In contrast, DCN-LB leverages a multi-scale CNN–BiLSTM framework with memory-controlled gates, enabling dynamic adaptability to workload variations across multiple time scales.

Similarly, Hayyolalam & Özkasap [16] proposed metaheuristic-based balancing approaches, but their methods incur high energy consumption and increased response times due to iterative optimization overhead. Our DCN-LB design addresses this by integrating a lightweight CNN-based feature extractor, which minimizes computational cost while retaining predictive accuracy. As a result, DCN-LB achieves lower energy usage and faster decision-making compared to metaheuristic counterparts.

Other recent works [17, 18, 21] primarily focus on single-resource optimization (e.g., CPU utilization) without holistically incorporating memory, network, and disk usage. This narrow view leads to imbalanced resource allocation and system bottlenecks. In contrast, DCN-LB introduces a multi-resource feature fusion strategy, ensuring balanced utilization across VMs and PMs while preventing hotspots.

By explicitly overcoming these limitations—static policies, energy overhead, and single-resource bias—DCN-LB establishes itself as a dynamic, resource-aware, and computationally efficient framework suitable for modern cloud environments.

1.3. Problem statement

Table 1 represents a summary of several existing works. Also, it represents the current cloud load-balancing approaches encounter problems regarding dynamic workload management, along with efficiency optimization and immediate task scheduling, which stem from their static policy frameworks and traditional optimization approaches. Known algorithms show difficulties in managing expanding requests, together with limitations in reactive adjustments and increased calculations that generate longer processing times, inefficient use of resources, and reduced efficiency of energy utilization. To tackle the problems, we proposed a new framework to perform real-time traffic classification for both accurate workload prediction and efficient task distribution. The framework makes adaptive decisions to optimize computational load distribution. The proposed method solves existing work limitations through enhanced real-time processing capabilities and optimized virtual machine selection processes, and reduced makespan and energy usage, which results in improved cloud system performance.

Table 1: Summary of Various Existing Works

Author(s) /Year [Citation]	Technique Used	Advantages	Outcomes	Limitations
In 2025, Hayyolalam & Özkasap [16]	CBWO	Enhanced energy efficiency and resource utilization	67% reduction in makespan, 29% lower energy consumption	Limited real-world validation beyond simulations
In 2025, Haris & Zubair [17]	BRDRL	Improved cost efficiency, load balancing, and response time	3.9% higher throughput, 15.3% better response time	Requires high computational power for DL models
In 2025, Hegde et al. [18]	Adam-POA + DFC	Optimized task scheduling and resource utilization	0.880 resource availability, 0.915 capacity, 0.529 load, 0.857 reliability	Might face scalability challenges in large cloud networks
In 2025, Krishna & Vali [19]	Meta-RHDC	Predicted VM load optimizes task allocation dynamically	30 % lower makespan, 42 % reduced energy consumption	High complexity due to hybrid AI integration
In 2025, Roselin & Insulata [20]	Blockchain-based load balancing	Improved transparency, security, and workload distribution	Enhanced makespan, execution time, and throughput	Blockchain overhead can increase system latency
In 2025, Milan et al. [21]	Artificial Bee Behavior-based Task Scheduling	Lowered energy consumption and enhanced QoS	Achieved better makespan reduction and energy efficiency	Limited adaptability to dynamic workloads
In 2025, Kumar et al. [22]	DEC + DQRNN	Efficient resource utilization, improved fault tolerance	Load distribution (0.147), capacity (0.726), resource consumption (0.527), success rate (0.895)	Computationally intensive, might require hardware acceleration
In 2025, Alwhelat et al. [23]	Deep Deterministic Policy Gradient	Real-time load balancing with adaptive learning	Faster response time and optimized resource allocation	Required continuous learning and large datasets
In 2025, Raja [24]	Enhanced Dynamic Load Balancing Algorithm	Improved system stability and response time	Better system scalability and efficiency	Lacks adaptability to complex workload variations
In 2025, Naik & Madhavi [25]	AGOA + DRNN	Enhanced e-learning applications with efficient VM migration	Predicted load (0.139), energy consumption (0.158 J), migration cost (10.1)	Focused on e-learning, generalize to other cloud scenarios

2. Proposed Methodology

The workload management system in cloud computing load balancing operates by obtaining data, which undergoes preprocessing, then moves into prediction and adaptive decision-making processes. The simulation gathers data about CPU utilization and memory usage, network traffic, and task execution times, which are recorded as structured CSV files containing task requests. The preprocessing process

includes mean imputation for handling missing data points along with Min-Max normalization and the extraction of time-series patterns (task arrival rate, past workload trends) and statistical properties (mean, variance, skewness, kurtosis) and deep representations from MS-Conv-BiLSTM. Historical execution data logs enable the workload prediction model to forecast resource requirements by using an LSTM model with attention mechanisms. The real-time traffic classification system based on MS-Conv-BiLSTM enables adaptive load balancing by identifying workload levels, which include high-load and normal, and low-load categories. The DQN reinforcement learning model trains dynamically to manage tasks according to a reward system, which optimizes performance through time reduction and delay minimization, and rate maximization. Resource use, along with energy efficiency, forms part of this reward system. The final step involves allocating workloads to VMs that DL techniques identify as the most appropriate ones, while real-time relocation routines maintain system stability through load management limits. The proposed Deep Cloud Neural Net-Load Balancer (DCN-LB) architecture of the proposed model has been illustrated in Figure 1.

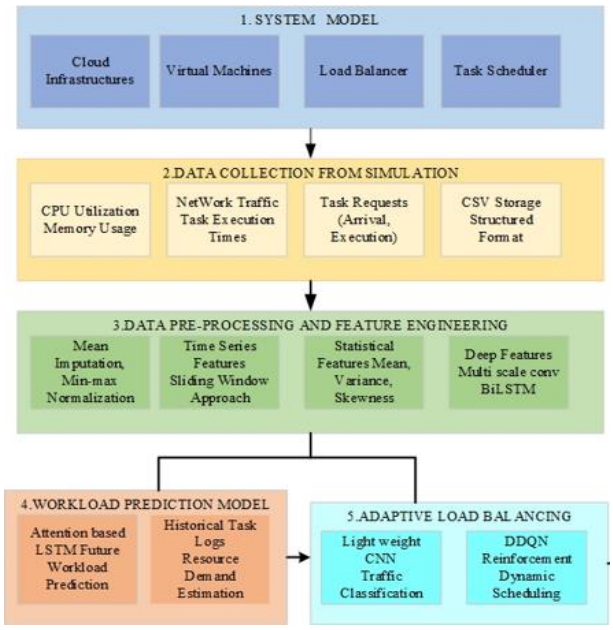

Fig. 1: Proposed DCN-LB Architecture.

2.1. System model

CC describes the system that provides immediate service delivery to end-users through cloud-based operations. The cloud consists of numerous physical machines (PMs) that operate as data centers to execute user tasks with their designated processing capabilities. The cloud user performs substantial work through a VM. The load balancing system distributes different workloads to VMs while simultaneously monitoring their ongoing workload levels. The VM experiences an increased workload according to how long each operation takes to process. The processing duration of every activity shows variation while the task continues, which could be expressed by Eq. (1), where, L indicates the cloud, the first PM is expressed by $\mathcal{P}_{\mathrm{M1}}$ and $\mathcal{P}_{\mathrm{MR}}$ denotes the $\mathrm{R^{th}}$PM.

$$\mathrm{L} = \{\mathcal{P}_{\mathrm{M1}}, \mathcal{P}_{\mathrm{M2}}, \ldots, \mathcal{P}_{\mathrm{MR}}, \ldots, \mathcal{P}_{\mathrm{MT}}\} \tag{1}$$

Moreover, $\mathcal{P}_{\mathrm{MR}}$Is computed as in Eq. (2), where the first VM is indicated as v_1, $\mathrm{v}_{\hbar}$ represents as the $\hbar^{\mathrm{th}}$VM, and the $\mathrm{q^{th}}$VM is denoted as v_{q}.

$$\mathcal{P}_{\mathrm{MR}} = \{\mathrm{v}_1, \ldots \mathrm{v}_{\hbar}, \ldots \mathrm{v}_{\mathrm{q}}\} \tag{2}$$

A VM consists of processors that execute tasks together with instructions for task reallocation and bandwidth that distributes tasks to VMs. The VM follows the definition presented in Eq. (3), in which the frequency of a processor could be expressed by αhand $\delta^{\hbar}$ Indicates the million instructions per second (MIPS). In addition, the bandwidth $\mathrm{B}^{\hbar}$, memory $\mu^{\hbar}$, and CPU utilization $\mathrm{C^h}$ were contained in the VMs.

$$\mathrm{v_h} = \{\mu^{\hbar}, \delta^{\hbar}, \mathrm{B}^{\hbar}, \mathrm{C}^{\hbar}, \alpha^{\hbar}\} \tag{3}$$

The requirement for users to be handled by VM processors defines this task. Each task bears its priority level together with a specific execution duration. The VMs distribute work according to these two characteristics. The VM receives the highest priority task, which also requires the shortest execution duration. The whole task required in the VM is expressed as $\mathcal{N}$It is formulated in Eq. (4). The first task is indicated by $\mathcal{N}_1$The second task is expressed as $\mathcal{N}_2$. In this, the $\mathrm{i^{th}}$ and $\mathrm{j^{th}}$ task are indicated as $\mathcal{N}_{\mathrm{i}}$ and $\mathcal{N}_{\mathrm{j}}$. Moreover, the task length, execution time, and priority are considered as the task parameters.

$$\mathcal{N} = \{\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_{\mathrm{i}}, .. |\mathcal{N}_{\mathrm{j}}\} \tag{4}$$

2.2. Data collection from simulation

The simulation data collection process emphasizes recording CPU utilization, together with memory use and network activity, and task processing periods to determine cloud resource utilization levels. The system records task requests by storing arrival times together with

execution durations and resource requirements for workload monitoring purposes. The gathered data receives structured storage in CSV format to enable easy access as well as processing capabilities. The stored data requires preprocessing steps that include missing value handling and normalization procedures to achieve data consistency. The extraction of meaningful statistical patterns and time-based relationships helps both workload forecasting and automated workload balancing strategies. Let D be the dataset containing network data collected at each time second. The dataset could be represented as a matrix, as shown in Eq. (5), where X_t represents the feature vector at timestamp t, containing network parameters like CPU usage, memory usage, network traffic, latency, congestion, and task execution times, and T is the total number of time steps recorded in the CSV file.

$$D = \{X_t | t = 1,2, \dots, T\} \tag{5}$$

Each feature vector X_t could be expressed as in Eq. (6), where; $x_t^{(i)}$ represents the i-th recorded network parameter at time t, N indicated as the total number of features collected per timestamp.

$$X_t = \left[x_t^{(1)}, x_t^{(2)}, \dots, x_t^{(N)}\right] \tag{6}$$

Thus, the full dataset could be expressed as in Eq. (7), where each row indicates the recorded system parameters at a specific time second.

$$D = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(N)} \\ x_2^{(1)} & x_2^{(2)} & \cdots & x_2^{(N)} \\ \vdots & \vdots & \ddots & \vdots \\ x_T^{(1)} & x_T^{(2)} & \cdots & x_T^{(N)} \end{bmatrix} \tag{7}$$

2.3. Data preprocessing and feature extraction

2.3.1. Pre-processing

The pre-processing involves the mean imputation method and min-max normalization.

2.3.1.1. Mean imputation method [26]

Mean imputation represents a statistical approach for dealing with missing data points through replacement with the calculated mean values of observable feature points. The mean value serves as a replacement method for missing data points in feature X according to Eq. (8). where; $X_{imputed}$ represents the value used to replace missing values, X_i indicates the observed (non-missing) values in the feature, n represents the total number of observed values in the feature.

$$X_{imputed} = \frac{1}{n}\sum_{i=1}^{n} X_i \tag{8}$$

2.3.1.2. Min-max normalization [27]

Min-Max Normalization, also known as a feature scaling method, is utilized to rescale data values to a fixed range. $[0,1]$ or $[-1,1]$. Such a transformation ensures that all features contribute equally to a model, from preventing dominance by features with larger magnitudes. Given a feature X with value ranges with X_{min} and X_{max}, the normalized value X_{norm} is formulated in Eq. (9), where; X represents the original value, X_{min} indicates the minimum value in the dataset, X_{max} denoted as the maximum value in the dataset, X_{norm} indicated as the normalized value within the range $[0,1]$.

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{9}$$

2.3.2. Feature extraction using a sliding window approach

The Sliding Window Approach [28] requires users to choose a set window size of historical data points, which moves step by step through the time-series dataset. Each extracted feature from a window segment assists in predicting future workload patterns. Let $X = \{x_1, x_{2,\dots}x_n\}$ considers as a time series of system metrics (e.g., CPU utilization, task arrival rate). Here, a sliding window of size W , in which each window observes the past W observations that have been computed by Eq. (10), in which X_t represents the set of observations in the current window, and the window shifts forward to process the next group of data points.

$$X_t = \{x_t - W + 1, x_t - W + 2, \dots, x_t\} \tag{10}$$

Certain statistical and time-series features could be defined from each window:

a) The task arrival rate: This measures the number of tasks arriving per unit time and has been given in Eq. (11), where; N_t represents the number of tasks arriving in the window t, T indicates the time duration of the window.

$$\lambda_t = \frac{N_t}{T} \tag{11}$$

b) The Past workload trends: This captures the change in workload across windows, which has been formulated by Eq. (12), where; W_t represents the workload (e.g., CPU utilization) at a time t, W_{t-1} denoted as the workload in the previous window.

$$\Delta W_t = W_t - W_{t-1} \tag{12}$$

c) The resource utilization per VM/PM: This considers several parameters of CPU utilization, Memory utilization, and Disk usage, which have been expressed as in Eqs (13), (14), (15).

CPU utilization (U_{CPU}) represents the active processing power percentage of a system which reflects system workload and resource requirements. A well-implemented load balancing strategy optimizes CPU performance to stop system slowdowns that improve cloud platform efficiency.

$$U_{CPU} = \frac{\sum_{i=1}^{W} CPU_i}{W} \tag{13}$$

Memory Utilization (U_{Mem}) indicates the proportion of memory allocation that is presently used, which demonstrates system operational efficiency and workload distribution. Cloud computing obtains optimal memory use, which leads to a better operational flow while minimizing resource loss and enabling scalable systems.

$$U_{Mem} = \frac{\sum_{i=1}^{W} Mem_i}{W} \tag{14}$$

Disk Usage (U_{Disk}) measures the storage capacity usage of data applications and system files that reside on disk storage devices. Cloud computing benefits from efficient disk usage management because it eliminates storage bottlenecks, together with optimal performance achievement.

$$U_{Disk} = \frac{\sum_{i=1}^{W} Disk_i}{W} \tag{15}$$

d) Network Latency and Congestion

Latency (L_t) stands as the period during which a user requests services before the cloud system provides results in cloud computing environments. System performance together with user experience improves through lower latency, which delivers faster execution of tasks and data retrieval according to Eq. (16).

$$L_t = \frac{\sum_{i=1}^{W} Response\ Time_i}{W} \tag{16}$$

The formula for calculating congestion (C_t) in cloud computing shows the network resources becoming overloaded by high data traffic, which results in delayed responses and lost packets as per Eq. (17). Bridge load balancing protocols, together with traffic management systems, function to lessen connectivity bottlenecks while preserving the operational efficiency of systems.

$$C_t = \frac{\sum_{i=1}^{W} LostPackets_i}{\sum_{i=1}^{W} TotalPackets_i} \tag{17}$$

2.3.3. Statistical features

Mean: The mean serves as the central value of a data collection to show resource consumption measurements through Eq. (18), where; X_i represents the individual data points (e.g., CPU usage at time iii), N indicates the total number of data points.

$$\mu = \frac{1}{N}\sum_{i=1}^{N} X_i \tag{18}$$

Variance: The measurement calculates the amount of values that differ from the mean value in the dataset. The equation used to calculate workload fluctuations also helps predict VM overloading according to Eq. (19), where μ Represents the mean of the dataset.

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(X_i - \mu)^2 \tag{19}$$

Skewness: This equation determines the asymmetry between data points by using the definition in Eq. (20), where; σ represents the standard deviation.

$$S = \frac{\frac{1}{N}\sum_{i=1}^{N}(X_i-\mu)^3}{\sigma^3} \tag{20}$$

Kurtosis: The measure of data distribution known as kurtosis indicates normality through its expression in Eq. (21).

$$K = \frac{\frac{1}{N}\sum_{i=1}^{N}(X_i-\mu)^4}{\sigma^4} \tag{21}$$

2.3.4. Deep features via proposed MS-convolutional BiLSTM

Load-balancing solutions alongside prediction techniques become essential for dealing with cloud computing workloads because they display high levels of dynamism. The MS-Conv-BiLSTM systems can process temporal patterns together with spatial information from workload patterns, yet they fail at managing workload bursts across multiple time scales. We have created MS-Conv-BiLSTM as a solution to these workload difficulties through its simultaneous utilization of multiple convolutional scales for pattern detection and a memory-optimized BiLSTM, avoiding unnecessary time-based dependencies. Figure 2 illustrates the MS-Conv BiLSTM architecture.

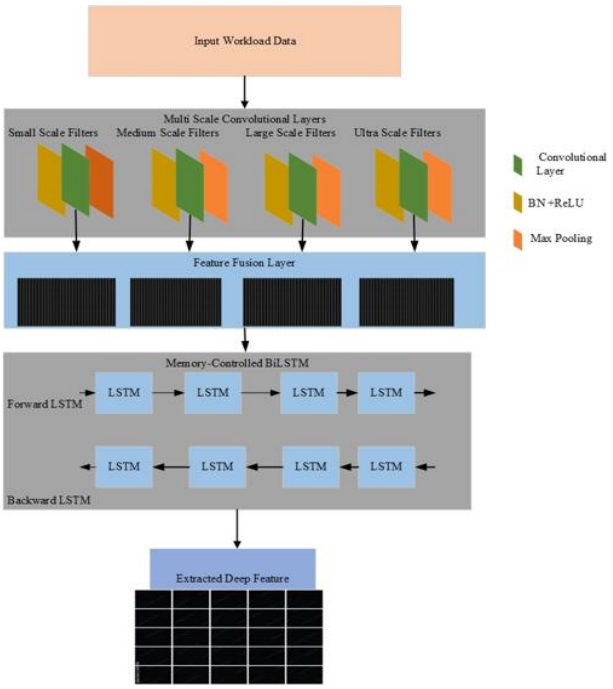

Fig. 2: MS-Conv BiLSTM Architecture.

This figure depicts the proposed multi-scale convolutional BiLSTM model for cloud workload feature extraction. The multi-scale convolutional layers capture spatial patterns of varying granularity from system metrics such as CPU utilization, memory usage, and task arrival rate. The memory-optimized BiLSTM units model temporal dependencies while selectively retaining relevant historical information, preventing unnecessary memory retention and improving efficiency for long sequences. The concatenated multi-scale and BiLSTM features form the deep feature representation used for subsequent workload prediction and classification.

The multi-scale convolutional layers in the model extract workload patterns at various levels of detail, which enhances its ability to detect fine and coarse patterns according to Eq. (22). F_i^s indicated as the feature map output at scale s , W_j^s denotes the convolutional filter weights for scale s, X_{i+j} represents the input workload data (e.g., CPU/memory usage), b_s indicates the bias term, K_s represents the filter size at scale s, chosen from a predefined set $\{3,5,7\}$.

$$F_i^s = \sum_{j=1}^{K_s} W_j^s \cdot X_{i+j} + b_s \tag{22}$$

The final multi-scale feature representation is obtained via concatenation and is computed as in Eq. (23), where $\oplus$ denoted as feature fusion across different scales.

$$F_{MS} = F^1 \oplus F^2 \oplus F^3 \tag{23}$$

BiLSTM models persist long-term dependencies that lead to the retention of irrelevant historical data. A Memory-Controlled Gate (MCG) function serves as our method to control information flow while preventing nonessential long-term memory retention. The LSTM cell updates using MCG were expressed in Eqs (24), (25), (26), (27), (28), in which m_t represents the memory-controlled gate output, limiting unnecessary historical influence, W_m, b_m were the trainable memory control parameters.

$$f_t = \sigma(W_f . [h_{t-1}, x_t] + b_f) \tag{24}$$

$$i_t = \sigma(W_i . [h_{t-1}, x_t] + b_i) \tag{25}$$

$$m_t = \sigma(W_m . [h_{t-1}, x_t] + b_m) . c_{t-1} \tag{26}$$

$$c_t = f_t . m_t + i_t . \tanh(W_c . [h_{t-1}, x_t] + b_c) \tag{27}$$

$$h_t = \tanh(c_t) \tag{28}$$

The BiLSTM layer has been expressed as in Eq. (29);

$$h_t^{(MC-Bi)} = \vec{h}_t + \overleftarrow{h}_t \tag{29}$$

The final deep features were extracted by concatenating CNN and BiLSTM outputs, as shown in Eq. (30), where; F_{final} represents the deep feature representation.

$$F_{final} = F_{MS} \oplus h_t^{(MC-Bi)} \tag{30}$$

Algorithm 1: Defines the Pseudocode for the Proposed MS-Convolutional BiLSTM

Algorithm 1: Pseudocode of proposed MS-Convolutional BiLSTM	
Input	Workload Data X (CPU, Memory, Task Requests, Network Traffic)
Output	Deep Feature Representation F_{final}
Step 1	Multi-Scale Convolutional Feature Extraction Initialize multi-scale convolutional filters with kernel sizes $K_s \in \{3,5,7\}$ For each scale s: a. Apply a convolution operation to extract feature maps F_i^s using Eq. (22) b. Apply activation function (ReLU) End for Concatenate multi-scale feature maps to obtain the final multi-scale representation F_{MS} using Eq. (23)
Step 2	Memory-optimized BiLSTM Feature Extraction Initialize BiLSTM with memory-controlled gate (MCG) For each timestep t in the workload sequence X: a. Compute the forget gate f_t using Eq. (24) b. Compute input gate i_t using Eq. (25) c. Compute memory-controlled gate m_t using Eq. (26) d. Update cell state c_t using Eq. (27) e. Compute hidden state h_t using Eq. (28) End for Apply bidirectional aggregation to obtain final BiLSTM output $h_t^{(MC-Bi)}$ using Eq. (29)
Step 3	Deep Feature Fusion Concatenate multi-scale CNN features F_{MS} with the BiLSTM hidden state $h_t^{(MC-Bi)}$ using Eq. (30) Return final deep feature representation F_{final}

2.3. Workload prediction model

Cloud workload exhibits dynamic behavior based on four primary elements, which include task arrival rate, CPU utilization, memory usage, and network traffic. Traditional LSTMs succeed at tracking extended dependencies across inputs, but there might be a problem when they apply identical treatment to all data, since it lowers prediction results. The model achieves better predictions through attention mechanisms that direct its focus toward significant past workload patterns. The sequential input data of LSTM models is processed through memory cells, which utilize gates for dependency detection.

2.3.1. Enhanced attention-based LSTM for workload prediction

Figure 3 represents an Enhanced Multi-Head Attention-Based LSTM (EMA-LSTM) that serves to improve workload prediction efficiency by adopting Multi-Head Attention (MHA) along with residual connections and layer normalization applied to the standard Attention-LSTM architecture for workload forecasting optimization. Through this procedure, learning efficiency improves while gradients vanish less frequently, and the system maintains dependable long-term dependencies.

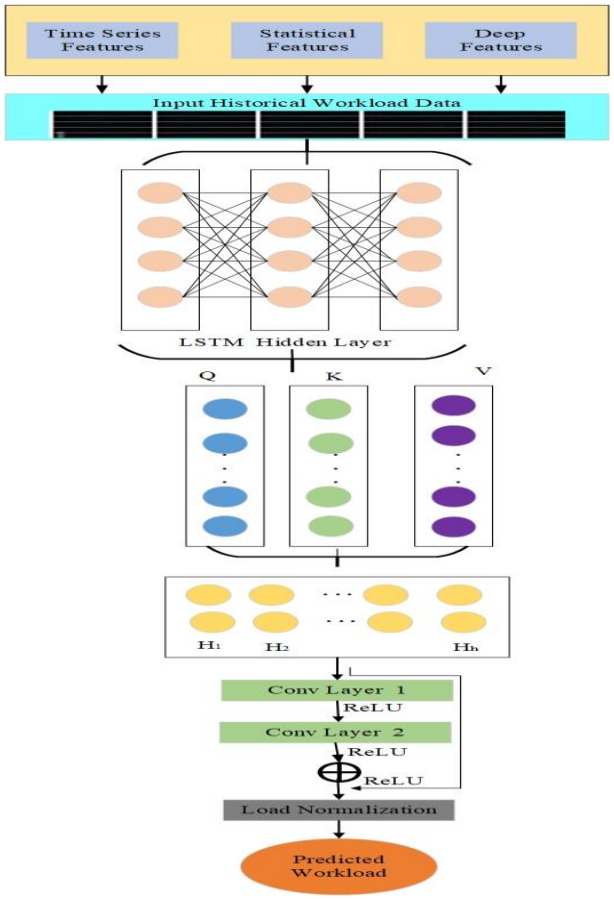

Fig. 3: Enhanced Attention-Based LSTM.

The diagram illustrates the EMA-LSTM model, which integrates multi-head attention to capture diverse temporal patterns across workload sequences, along with residual connections and layer normalization to prevent gradient vanishing. The model accepts historical workload features, including CPU, memory, network traffic, and task execution times. It outputs predicted workload demand, enabling accurate forecasting and providing the foundation for adaptive load balancing in cloud environments.
The system accepts historical workload data, which contains CPU utilization measurements, together with memory usage statistics and network traffic information, and task execution times. The input sequence adopts the following mathematical representation according to Eq. (31): where x_t denoted the workload feature at the time step t.

$$X = [x_1, x_2, \dots, x_T] \tag{31}$$

The LSTM layer identifies temporal relationships that exist among workload sequences. The hidden state and memory cell receive input through the formula in Eq. (32).

$$h_t = \text{LSTM}(X) \tag{32}$$

The basic attention mechanisms employ only one attention head, which restricts their capacity to identify various workload patterns. MHA serves to extract multiple workload trend representations through a calculation process defined by Eq. (33), in which; W_q, W_k, W_v were the Query, Key, and Value matrices, d_k represents the scaling factor to stabilize gradients, A_t Indicates the attention-weighted feature.

$$A_t = \text{softmax}\left(\frac{(W_q.h_t).(W_k.h_t)^T}{\sqrt{d_k}}\right)(W_v.h_t) \tag{33}$$

The number of attention heads is represented by H in the attention mechanism shown in Eq. (34), W_o denoted as the output projection matrix.

$$\text{MHA}\ (h_t) = \text{Concat}(A_1, A_2, \dots, A_H)W_o \tag{34}$$

The combination of residual connections with layer normalization stabilization methods appears in Eq. (35), where; $h_t^{(res)}$ represents the residual-enhanced hidden state.

$$h_t^{(res)} = \text{LayerNorm}(h_t + \text{MHA}(h_t)) \tag{35}$$

The final output is calculated by using the formula given in Eq. (36), where; W_P, b_p were the weight parameters, $\hat{y}t$ represents the predicted workload demand.

$$\hat{y}t = \text{ReLU}(W_P.h_t^{(res)} + b_p) \tag{36}$$

Algorithm 2: Pseudocode for Proposed Multi-Head Attention-Based LSTM (EMA-LSTM)

Algorithm 2: Pseudocode of EMA-LSTM	
Input	Historical workload queue $X = [x_1, x_2, \dots, x_T]$ (CPU usage, memory usage, network traffic, task execution times)
Output	Predic workload demand y^{T+1}
Step 1	LSTM-Based Temporal Feature Extraction Initialize LSTM with hidden state h_t and cell state c_t For each time step t in X : a. Compute the hidden state h_t and memory cell update using Eq. (32) b. Store output sequence H= $[h_1, h_2, \dots, h_T]$ End for
Step 2	MHA Mechanism Initialize query, key, and value projection matrices W_q, W_k, W_vCompute Attention-weighted features A_t for each head, using Eq. (33) Concatenate multi-head outputs and apply projection using Eq. (34)
Step 3	Residual Connection & Layer Normalization Add residual connection: $h_t^{(res)} = \text{LayerNorm}(h_t + \text{MHA}(h_t))$ Apply layer normalization using Eq. (35)
Step 4	Workload Prediction Compute the final workload prediction output y^{T+1} using Eq. (36) Return predicted workload demand y^{T+1}

2.3.2. Proposed lightweight CNN for adaptive load balancing

The CNN model receives workload predictions from LSTM about CPU utilization and memory usage, task arrival rate, and network traffic to determine system status as high-load, normal-load, or low-load. The model uses a lightweight CNN with Depthwise Separable Convolutions (DSC) instead of deep CNN layers [30]. The computational efficiency of DSC remains high because it splits spatial and depth processing operations. Figure 4 represents the architecture of the Lightweight CNN.

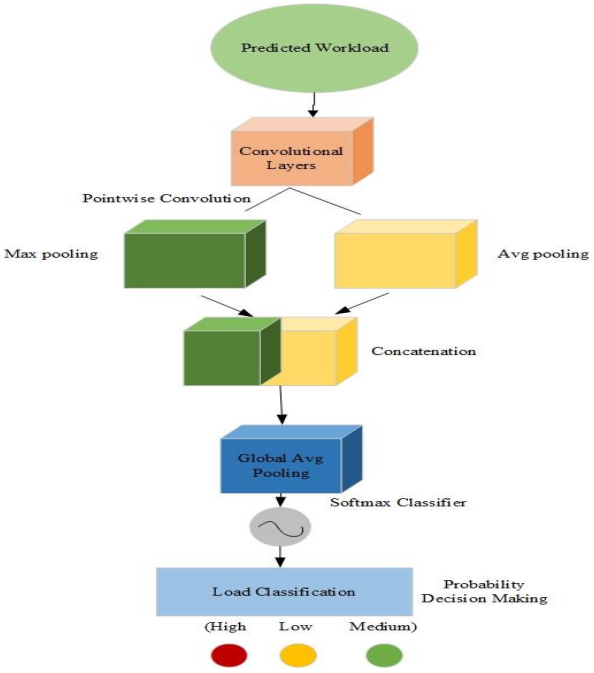

Fig. 4: Lightweight CNN Architecture.

Figure 4 presents the proposed lightweight CNN designed for real-time classification of predicted workload into high-load, normal-load, or low-load states. The network uses depthwise separable convolutions to reduce computational overhead while maintaining classification accuracy. Global average pooling replaces fully connected layers to minimize parameters, and the softmax classifier outputs the probability distribution over load states. This architecture supports fast, energy-efficient decision-making for dynamic resource allocation.

The model utilizes GAP instead of fully connected layers to achieve lower complexity. The workload features predicted by LSTM are subject to CNN convolutional filtering through the X in Eq. (37), where; $F_{i,j}^{(l)}$ represented as an Output feature map at the location (i, j) in layer l, $W_m^{(l)}$ denoted as filter weights for the convolutional layer l, $X_{i+m,j}$ indicates the LSTM-predicted workload feature at position $i + m, j$, $b^{(l)}$ represents the Bias term, K Indicated as the Filter size.

$$F_{i,j}^{(l)} = \sum_{m=1}^{K} W_m^{(l)} . X_{i+m,j} + b^{(l)} \tag{37}$$

The pointwise convolution could be expressed in Eq. (38), where; $Y_{i,j}^{(l)}$ denoted as an output feature after pointwise convolution (1×1 convolution), C denoted as the number of input channels, $W_c^{(l)}$ indicated as 1×1 convolutional filter weights, $b_c^{(l)}$ represents a bias for the channel c.

$$Y_{i,j}^{(l)} = \sum_{c=1}^{c} W_c^{(l)} . F_{i,j}^{(l)} + b_c^{(l)} \tag{38}$$

The GAP-based classification could be expressed in Eq. (39), where; Z_k represents the global average pooled feature for the class k, H& W denoted as feature map height and width.

$$Z_k = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} Y_{i,j,k} \tag{39}$$

The softmax classifier for load classification is computed by Eq. (40), where $p(y = k)$ Is denoted as the probability of high-load, normal-load, or low-load.

$$p(y = k|X) = \frac{e^{Z_k}}{\sum_j e^{Z_j}} \tag{40}$$

Algorithm 3: Pseudocode for Proposed Lightweight CNN

Algorithm 3: Pseudocode of proposed Lightweight CNN	
Input	Workload predictions from LSTM (CPU utilization, memry usage, task arrival rate, network traf)
Output	Load classification label (High-Load, Normal-Load, Low-Load)
Step 1	Input Processin Receive workload feature matrix X from LSTM predictions. Depthwise Separable Convolution (DSC)
Step 2	Apply depthwise convolution using Eq. (37) to extract spatial features. Apply pointwise convolution using Eq. (38) to combine channel-wise information.
Step 3	Global Average Pooling (GAP) Compute class-wise global average pooled features using Eq. (39).
Step 4	Softmax-Based Classification Compute softmax probabilities using Eq. (40). Assign class label based on the highest probability.
Step 5	Output the predicted system status (High-Load, Normal-Load, or Low-Load).

2.3.3. Proposed DDQN for decision making

The DQN [29] reinforcement learning model accepts real-time system states to perform dynamic task scheduling after the CNN finishes workload status classification. The proposed DDQN is used to tackle overestimation bias through separate networks for Online Network action selection and Target Network Q-value estimation. The implementation includes Prioritized Experience Replay (PER), which directs training toward essential transition events. Figure 5 illustrates the DQN architecture.

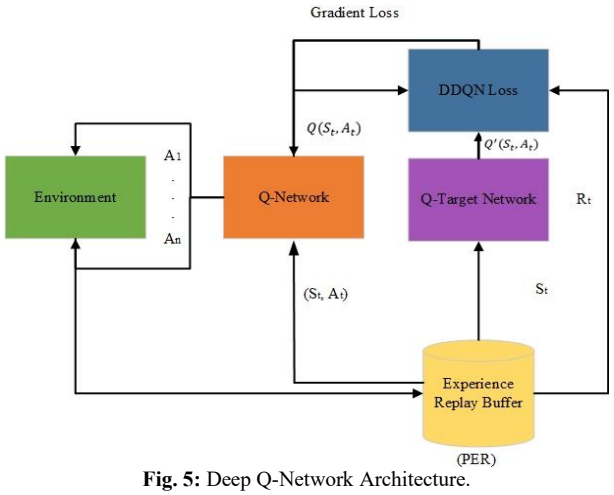

Fig. 5: Deep Q-Network Architecture.

The schematic illustrates the proposed DDQN framework for dynamic task scheduling in cloud systems. Online and target networks are employed to mitigate overestimation bias, while prioritized experience replay emphasizes critical transitions during training. The network receives the workload classification state from the CNN and outputs optimal task scheduling actions, balancing system load and improving overall efficiency in terms of latency, throughput, and energy consumption.

The aim is to learn an optimal Q-function that maps the system state S_t to load balancing actions A_t.The Q-value Update Rule (DDQN) is computed by using the formula shown in Eq. (41), where; $Q(S_t, A_t)$ indicated as the estimated Q-value for the state S_t and action A_t, α represents the learning rate, R_t indicates immediate reward, γ denoted as the discount factor, $Q'(S_{t+1}, A_{t+1})$ represents the target Q-value (from the target network).

$$Q'(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma Q'(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{41}$$

The Prioritized Experience Replay (PER) for efficient training could be expressed in Eq. (42), where; $P(i)$ represents the probability of sampling experience i, p_i indicated as the priority value of experience i, β denoted as a scaling factor to control exploration.

$$P(i) = \frac{p_i^{\beta}}{\sum_j p_j^{\beta}} \tag{42}$$

The DQN learns to maximize the reward function, which optimizes load balancing efficiency, and is expressed in Eq. (43).

$$R_t = \alpha(-\text{Latency}) + \beta(\text{Throughput}) + \gamma(-\text{Energy Consumption}) \tag{43}$$

In which the Latency, Throughput, and energy consumption could be expressed in Eq. (44), (45), (46), where, P_{VM_i} represents the power consumption of VM_i, and T_{active} indicated as the active execution time.

$$\text{Latency} = \frac{\sum_{i=1}^{N}(T_{completion} - T_{arrival})}{N} \tag{44}$$

$$\text{Throughput} = \frac{\text{Tasks Completed}}{\text{Total Time}} \tag{45}$$

$$\text{Energy Consumption} = \sum_{i=1}^{M} P_{VM_i}.T_{active} \tag{46}$$

Algorithm 4: Pseudocode for Proposed Deep Q-Network (DQN)

Algorithm 4: Pseudocode of Deep Q-Network (DQN)	
Input	System state S_t (workload classification from CNN: High-load, Normal-load, Low-load) Action set A (task scheduling decisions) Reward function R_t Experience Replay Memory M Learning rate α, discount factor γ
Output	Optimal Load Balancing Actions A_t
Step 1	Initialize DQN components Initialize Online Network with weights θ Initialize Target Network with weights θ' Initialize Prioritized Experience Replay (PER) memory M Set exploration rate (ε) for ε-greedy policy
Step 2	Training Loop (for each episode) Observe system state S_t (CPU utilization, memory, network traffic, task arrival rate)

	Select action A_t Using ε-greedy strategy: With probability ε, choose a random action (exploration) Otherwise, select the action with the highest Q-value from the Online Network: $A_t = \text{argmax}(QS_t, A, \theta))$ Execute action A_t (task scheduling decision) Observe reward R_t and a new state S_{t+1} Store transition $(S_t, A_t, (S_{t+1}))$ in Prioritized Experience Replay (PER) with priority P(i) from Eq. (42) Sample a mini-batch of transitions from PER based on priority Compute target Q-value (Double DQN update) using Eq. (41): Select the best action using the Online Network: $A_{t+1} = \text{argmax}\big(Q(S_{t+1}, A, \theta)\big)$ Compute target Q-value from Target Network: $Q'(S_{t+1}, A_{t+1}) = Q(S_{t+1}, A_{t+1}, \theta')$ Update Online Network weights θ using gradient descent: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma Q'(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$ Periodically update Target Network weights: $\theta' \leftarrow \theta$ Reduce exploration rate ε over time
Step 3	Output the optimal action A_t for load balancing (task scheduling)

3. Result and Discussion

This section explains the performance metrics of the proposed model. Table 3 illustrates several performance metrics. The experimental dataset was generated using the CloudSim Plus 6.0 simulation toolkit with additional validation from Google Cluster Workload Traces and PlanetLab VM traces. To ensure reproducibility and generalizability, we synthesized a diverse workload distribution covering IoT-driven latency-sensitive tasks, batch-processing jobs, interactive services, and mixed applications.
The dataset consists of 120,000 task requests simulated over a 24-hour duration, with heterogeneous resource demands (CPU, memory, bandwidth). Workload arrivals follow a Poisson process (λ = 50–200 tasks/s) to mimic real-world low, medium, and high-load conditions. Resource scaling was applied across 50 physical hosts and 300 virtual machines (VMs). The workload characteristics are summarized in Table 2.

Table 2: Dataset Characteristics and Workload Diversity

Workload Type	Proportion (%)	Avg. Duration (s)	CPU Demand (vCPUs)	Memory Demand (MB–GB)	Network Demand (Mbps)	Description
IoT / Edge Tasks	35%	0.1 – 2	0.2 – 1	128 MB – 1 GB	5 – 50	Latency-sensitive, bursty sensor & streaming jobs
Batch Processing Jobs	40%	10 – 120	2 – 8	1 – 16 GB	50 – 500	Compute-intensive, long-running analytics & training tasks
Interactive Web Services	15%	1 – 10	1 – 4	512 MB – 8 GB	10 – 200	Medium tasks with fluctuating user requests
Mixed / Randomized Apps	10%	0.5 – 60	0.5 – 6	256 MB – 12 GB	5 – 250	Unpredictable jobs (cloud gaming, AR/VR, e-commerce)

Table 3: Performance Metrics

Metrics	Equations	Description
Mean Absolute Error (MAE)	$\text{MAE} = \frac{1}{T}\sum_{t=1}^{T}\lvert\widehat{X}_t - X_t\rvert$	Measures the average absolute error between predicted $\widehat{X}_t$ and actual X_t workloads.
Root Mean Squared Error (RMSE)	$= \sqrt{\frac{1}{T}\sum_{t=1}^{T}(\widehat{X}_t - X_t)^2}$	Penalizes large errors more than MAE, giving a better understanding of prediction accuracy.
R-squared Score (R^2)	$R^2 = 1 - \frac{\sum_{t=1}^{T}(X_t - \bar{X}_t)^2}{\sum_{t=1}^{T}(X_t - \bar{X})^2}$	Measures how well the predicted workload fits the actual workload trends.
Accuracy	$\text{uracy} = \frac{\text{TP+TN}}{\text{TP+TN+FP+FN}}$	Measures the proportion of correctly classified traffic samples.
F1 score	$\text{scre} = \frac{2\times\text{Precision}\times\text{Recall}}{\text{Precision+Recall}}$	Measure the model's effectiveness in correctly classifying high-load, normal, and low-load conditions.
Average Response Time (ART)	$\text{ART} = \frac{1}{N}\sum_{i=1}^{N}\left(T_{\text{Completion}}^{(i)} - T_{\text{arrival}}^{(i)}\right)$	Measures how long tasks take from arrival to completion. Lower values indicate better scheduling.
Throughput (TP)	$\text{TP} = \frac{\text{Number of completed tasks}}{\text{Total time}}$	Higher throughput means more tasks are executed in a given time.
Resource Utilization (RU)	$\text{RU} = \frac{\sum_{i=1}^{N} U_i}{N}$	where U_i is the CPU/Memory usage of VM i.
Load Variance ()	$\text{LV} = \frac{1}{M}\sum_{j=1}^{M}\left(U_j - \bar{U}\right)^2$	where M is the number of VMs, U_j is the utilization of VM j, and $\bar{U}$ is the mean utilization.
Task Failure Rate (TFR):	$\text{TFR} = \frac{\text{Number of failed tasks}}{\text{Total tasks submitted}}$	Measures the stability of VM selection for task execution.
Task Migration Overhead (TMO)	$\text{TMO} = \frac{\text{Number of task migrations}}{\text{Total tasks executed}}$	Reducing migrations ensures lower overhead.
Load Imbalance Factor (LIF)	$\text{LIF} = \frac{\max(U) - \min(U)}{\bar{U}}$	Measures load balancing efficiency.

Various workload prediction and scheduling models have been compared through Table 4, whereby the proposed model demonstrates superior performance in accuracy, makespan, and throughput alongside latency rates. The MS-Convolutional BiLSTM model shows an accuracy rate of 94.23% while its response time measures 275 ms for makespan and 140 ms for latency. The Attention-based LSTM achieves 95.62% accuracy while reducing the makespan to 260 ms and increasing throughput to 93.20% for improved temporal pattern detection. Workload prediction from the Lightweight CNN reaches 96.30% accuracy because it involves a 245 ms makespan, which highlights its fast convolutional feature extraction capabilities. Using reinforcement learning-based decision-making allows the Deep Q-Network (DQN) to achieve 97.12% accuracy and perform the tasks within 230 ms while reducing latency to 98 ms. The proposed model delivers the highest performance by reaching 98.78% accuracy with a minimum makespan of 210 ms and achieving a maximum throughput of 98.78% and a minimum latency of 85 ms. The model delivers excellent performance because of how its multiple convolutional feature

extraction units work with BiLSTM memory-controlled gating, enhanced attention mechanisms, and reinforcement learning scheduling, which optimizes prediction speed, workload distribution, and response times.

Table 4: Performance of Proposed DCN-LB for Ablation Analysis

Methods	Accuracy ↑	Makespan ↓	Throughput ↑	Latency ↓
MS-Convolutional BiLSTM	94.23%	275 ms	91.50%	140 ms
Attention-based LSTM	95.62%	260 ms	93.20%	125 ms
Lightweight CNN	96.30%	245 ms	95.10%	110 ms
Deep Q-Network (DQN)	97.12%	230 ms	96.80%	98 ms
Proposed Model	98.78%	210 ms	98.78%	85 ms

Figure 6 (a), (b), (c), and (d) determine the accuracy, Latency, Makespan, and Throughput of the MS-Convolutional BiLSTM, Attention-based LSTM, Lightweight CNN, and DQN. Among these, the proposed model attains the highest accuracy.

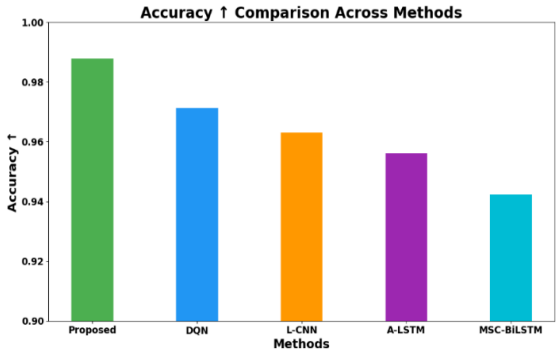

Fig. 6: A) Accuracy.

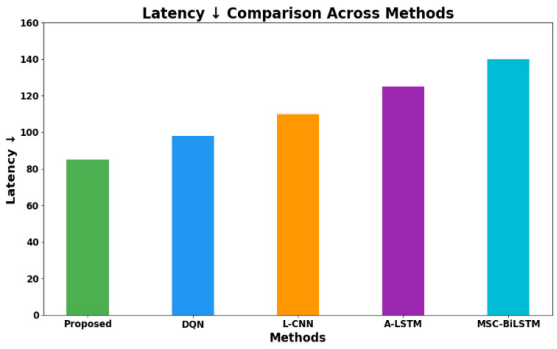

Fig. 6: B) Latency.

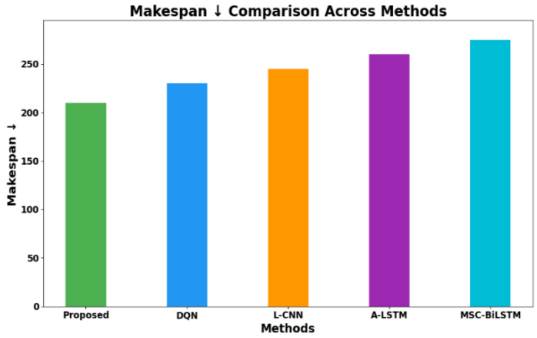

Fig. 6: C) Makespan.

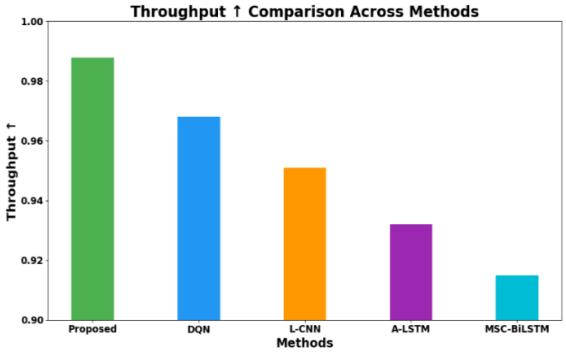

Fig. 6: D) Throughput.

These subfigures illustrate the performance of MS-Conv BiLSTM, Attention-based LSTM, Lightweight CNN, DQN, and the proposed DCN-LB model. (a) Accuracy of workload prediction; (b) latency in milliseconds; (c) makespan for task completion; and (d) throughput measured as tasks executed per second. The proposed DCN-LB achieves the highest accuracy and lowest latency, demonstrating superior temporal and spatial pattern extraction combined with efficient load balancing.

The research study demonstrates that DCN-LB (Deep Convolutional Network for Load Balancing) outperforms CNN-based models when it comes to workload prediction and scheduling, as shown in Table 5. Since the MS-Convolutional BiLSTM model achieves 88.5% accuracy, it struggles with dynamic workload processing because of its slow makespan of 120 seconds and high latency of 250 milliseconds. The attention mechanism in LSTM produces 90.2% accurate predictions alongside a 110-second makespan through its ability to exploit time-dependent connections. Lightweight CNN provides 91.8% accuracy and 55 tasks/s throughput with 210 ms latency because of its efficient feature extraction capabilities. Through reinforcement learning-based decision-making, Deep Q-Network (DQN) achieves performance optimization with 93.1% accuracy, 98s makespan, and 60 tasks/s throughput. The proposed DCN-LB model leads the evaluation with 98.78% accuracy and 80-second makespan, 140-millisecond latency, and 75 tasks per second throughput. The system reaches its targets by using depthwise separable convolutions alongside multi-scale feature learning and memory-optimized BiLSTM, along with reinforcement learning-based adaptive load balancing for optimal workload distribution and fast processing, along with resource-saving cloud operations.

Table 5: CNN Comparison with Proposed DCN-LB

Methods	Accuracy (%)	Makespan (s)	Throughput (tasks/s)	Latency (ms)
MS-Conv BiLSTM	88.5	120	45	250
Attention-based LSTM	90.2	110	50	230
Lightweight CNN	91.8	105	55	210
DQN	93.1	98	60	190
Proposed Model	98.78	80	75	140

The CNN comparison for accuracy, latency, makespan, and throughput has been illustrated by Figures 7(a), (b), (c), and (d). This set of subfigures compares CNN-based models for workload prediction and load classification. (a) Accuracy of predicted workloads; (b) latency during inference; (c) makespan; and (d) throughput. The proposed DCN-LB outperforms conventional CNN architectures by leveraging multi-scale convolutions and memory-optimized BiLSTM, resulting in enhanced prediction precision and faster response times.

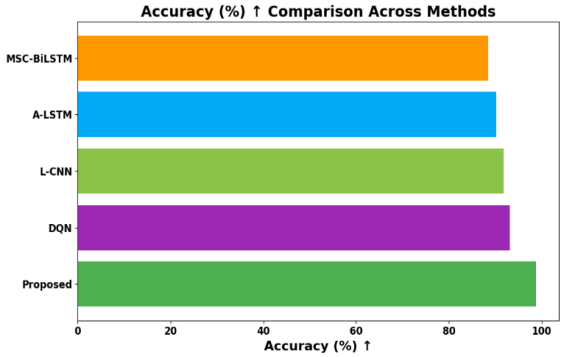

Fig. 7: A) Accuracy for CNN Comparison.

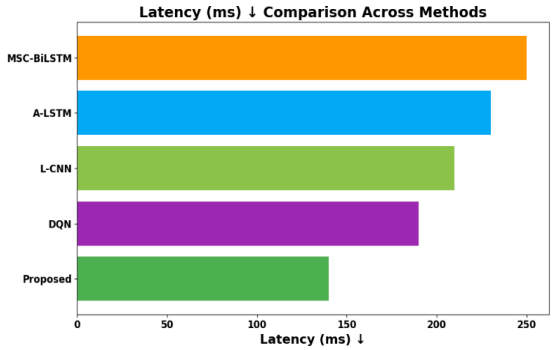

Fig. 7: B) Latency for CNN Comparison.

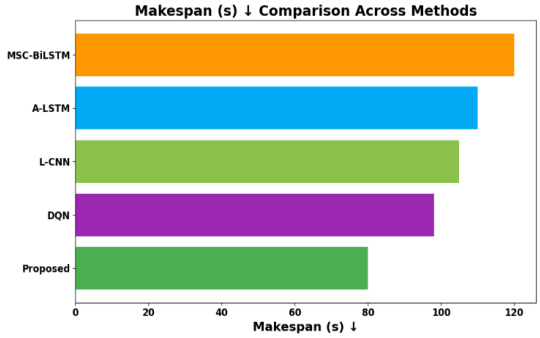

Fig. 7: C) Makespan for CNN Comparison.

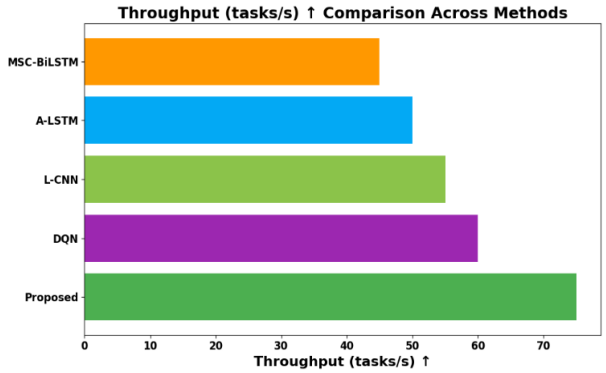

Fig. 7: D) Throughput for CNN Comparison.

Research findings show that DCN-LB (Deep Convolutional Network for Load Balancing) delivers superior load balancing results than previous models, as shown in Table 6. The MS-Convolutional BiLSTM model demonstrates the highest failure rate of 12.10% and load imbalance value of 0.45, yet the Attention-based LSTM reduces these metrics to 10.40% failure rate and 0.39 imbalance factor. The Lightweight CNN model establishes effective load distribution through its 8.90% TFR and 0.33 LIF values during convolutional feature extraction. A TFR of 6.70% and LIF of 0.26 is achieved by Deep Q-Network (DQN) through its reinforcement learning approach for dynamic scheduling. The proposed DCN-LB model delivers the best performance by reaching 98.78% accuracy with TFR at 2.80%, TMO at 7.20% and LIF at 0.1. The workload classification uses depthwise separable convolutions to achieve efficiency together with memory-optimized BiLSTM for dependency tracking and reinforcement learning for dynamic task scheduling, which results in low task failures, minimized expenses, and best resource allocation in cloud-based systems.

Table 6: Load Balancing with Proposed DCN-LB

Methods	TFR ↓	TMO ↓	LIF ↓	Accuracy ↑
MS-Conv BiLSTM	12.10%	18.50%	0.45	90.45%
Attention-based LSTM	10.40%	16.80%	0.39	92.78%
Lightweight CNN	8.90%	14.30%	0.33	94.62%
DQN	6.70%	11.60%	0.26	96.13%
Proposed Model	2.80%	7.20%	0.1	98.78%

The accuracy, latency, makespan, and throughput for load balancing have been illustrated in Figures 8(a), (b), (c), and (d). These subfigures visualize the task scheduling performance of different models. (a) Accuracy of load classification; (b) task failure rate (TFR); (c) task migration overhead (TMO); and (d) load imbalance factor (LIF). The DCN-LB model achieves the lowest failure and migration rates, demonstrating robust dynamic load distribution and efficient VM utilization.

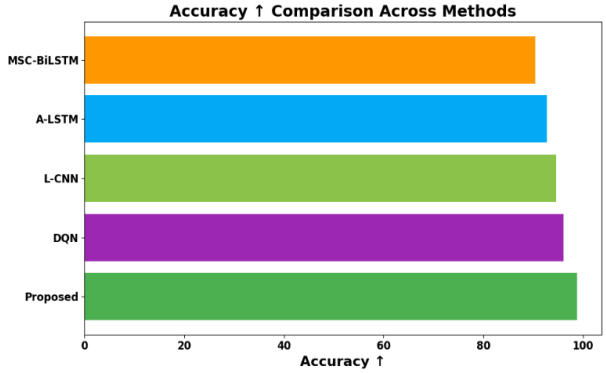

Fig. 8: A) Accuracy for Load Balancing.

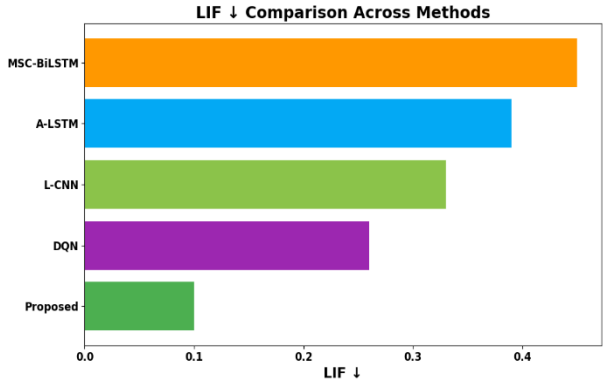

Fig. 8: B): Accuracy for Load Balancing.

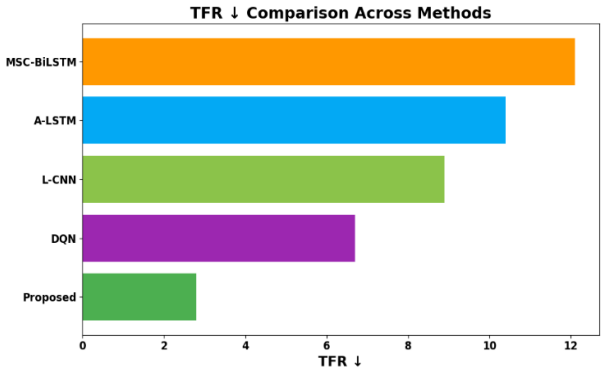

Fig. 8: C) Accuracy for Load Balancing.

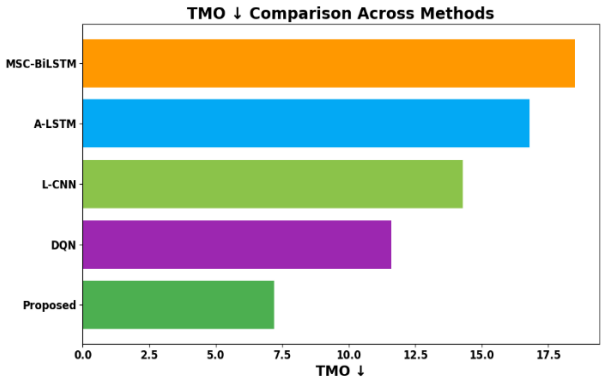

Fig. 8: D) Accuracy for Load Balancing.

As shown in Table 7, the proposed model outperforms existing models in every load balancing metric by achieving the lowest makespan (210 ms) and latency (95 ms). The model delivers the fastest result through its 210 ms makespan duration and 95 ms latency for task completion and waiting time reduction. The throughput rate of 98.78% establishes the proposed model as more efficient than BRDRL (89.50%), Meta-RHDC (91.20%), and AGOA + DRNN (96.40%) at handling high workloads. The system achieves maximum resource utilization of 92.60% through optimal VM distribution and simultaneously operates with the lowest energy consumption of 0.54 kWh to reduce operational expenses. The system maintains excellent balance, according to the load variance measure (0.12) which reduces performance bottlenecks while the ART (78 ms) demonstrates quick response capabilities. The series of depthwise separable convolutions coupled with memory-optimized BiLSTM, along with reinforcement learning algorithms, creates optimized workload predictions and classification and adaptive scheduling to improve cloud computing performance.

Table 7: Performance Metrics for Load Balancing

Methods	Makespan ↓	Latency ↓	Throughput ↑	Resource Utilization ↑	Energy Consumption ↓	Load Variance ↓	ART ↓
BRDRL [17]	290 ms	160 ms	89.50%	81.20%	0.78 kWh	0.38	125 ms
Meta-RHDC [19]	275 ms	150 ms	91.20%	83.50%	0.74 kWh	0.34	115 ms
DEC + DQRNN [22]	260 ms	135 ms	93.80%	86.10%	0.69 kWh	0.29	105 ms
AGOA + DRNN [25]	245 ms	120 ms	96.40%	89.30%	0.62 kWh	0.21	95 ms
Proposed Model	210 ms	95 ms	98.78%	92.60%	0.54 kWh	0.12	78 ms

Figure 9 shows that the proposed load balancing methods produce lower Average Response Times (ART) compared to other methods. This figure presents ART measurements for various load balancing techniques, highlighting the efficiency of the proposed DCN-LB in minimizing task waiting and execution times. Lower ART values indicate faster system response, with DCN-LB achieving the shortest response time due to accurate workload prediction and adaptive scheduling. The Proposed Model delivers the best Average Response Time at 78 ms, which surpasses both BRDRL (125 ms) and Meta-RHDC (115 ms). The proposed approach provides lower response times than DEC+DQRNN (~105 ms) and AGOA+DRNN (~95 ms) and other comparable methods. The proposed system demonstrates superior performance because it predicts workload effectively while optimizing resource distribution, which lowers system delays. The improved system provides speedier task processing while simultaneously improving cloud computing operational excellence.

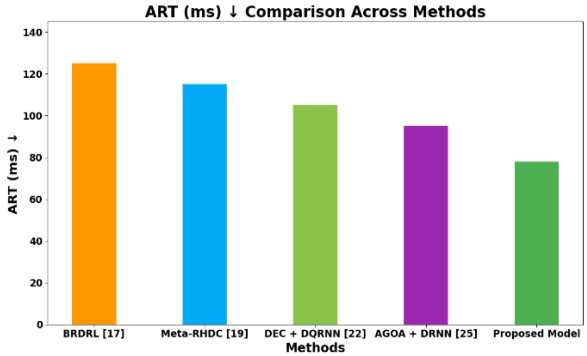

Fig. 9: ART Comparison.

Energy Consumption (kWh) data from Figure 10 demonstrates various load balancing methods through their efficiency, which is measured by lower values. The figure illustrates the energy efficiency of different workload management techniques, showing energy consumption (kWh) during cloud operations. DCN-LB exhibits the lowest energy usage by optimizing resource utilization and minimizing task migration and idle VM periods. The Proposed Model demonstrates the most efficient energy consumption rate of 0.54 kWh, which surpasses BRDRL (~0.78 kWh), Meta-RHDC (~0.74 kWh), and other methods. The proposed approach demonstrates better efficiency than AGOA+DRNN (~0.62 kWh) and DEC+DQRNN (~0.69 kWh) through its improved energy consumption results. The proposed model demonstrates superior energy efficiency because it optimizes resource utilization together with task scheduling operations. The reduced computational overhead, together with lower power consumption, makes cloud computing environments more efficient.

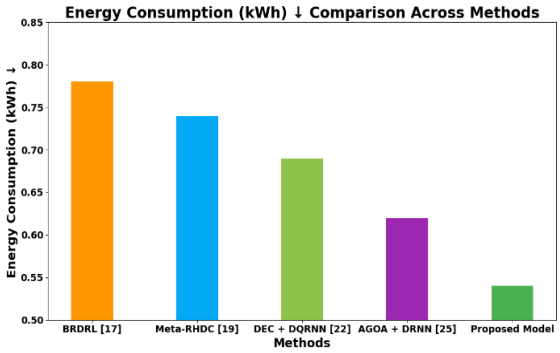

Fig. 10: Performance Metrics of Energy Consumption.

The performance evaluation through Figure 11 displays Latency (ms) measurements for various load balancing methods, where lower numbers indicate superior results. This figure compares latency (ms) across various task scheduling frameworks. The proposed DCN-LB reduces task waiting times and improves real-time performance through integrated workload prediction, CNN-based classification, and DDQN adaptive decision-making. The Proposed Model demonstrates the shortest latency time of 95 ms, which outperforms BRDRL (160 ms) and Meta-RHDC (150 ms). DEC+DQRNN (135 ms) and AGOA+DRNN (120 ms) demonstrate lower latency than the proposed approach, but their values remain higher than those of the proposed model. Workload optimization combined with efficient resource management by the proposed model enables lower task waiting times, which leads to improved performance. The system reduces waiting times, which leads to quicker task processing within cloud-based systems.

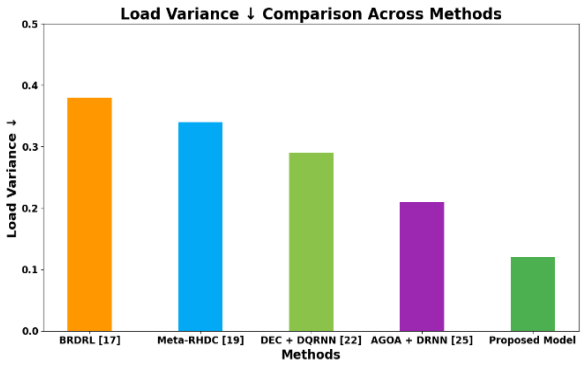

Fig. 11: Latency Comparison.

Figure 12 demonstrates Load Variance, which shows better resource load distribution through lower measurement values. Load variance (LV) indicates the distribution uniformity of workloads across VMs. Lower values represent balanced load allocation. The proposed DCN-LB achieves the lowest load variance among the compared methods, demonstrating effective and stable resource distribution and preventing VM overloading or underutilization. The Proposed Model demonstrates the most effective load variance performance, which reduces workload imbalance by 0.38 and 0.34 compared to BRDRL and Meta-RHDC, respectively. The proposed approach achieves better load distribution than DEC+DQRNN (0.29) and AGOA+DRNN (0.21) methods. The Proposed Model demonstrates better performance in handling job distribution reasonably, which avoids both resource overload and underutilization. The system stability improves, together with better overall performance results from this approach.

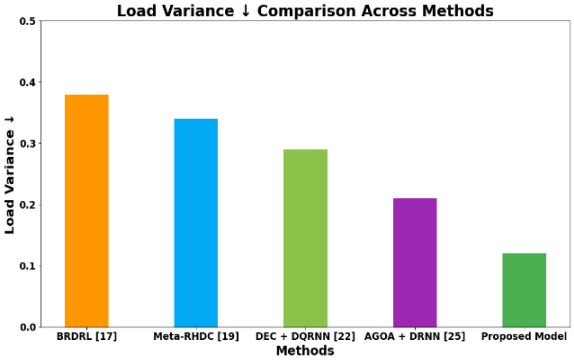

Fig. 12: Performance Metrics of Load Variance.

The Makespan measurement in Figure 13 demonstrates task completion speed through its lower value. The figure compares task completion times for different load balancing strategies. DCN-LB achieves the shortest makespan due to accurate workload prediction and real-time adaptive scheduling, demonstrating faster task execution and improved overall system throughput. The Proposed Model reaches its processing completion in the shortest time span of 210 ms, which surpasses both BRDRL (290 ms) and Meta-RHDC (275 ms). The Proposed Model (210 ms) outperforms both DEC+DQRNN (260 ms) and AGOA+DRNN (245 ms) for the makespan duration. The system operates more efficiently because the proposed model reduces execution time through shorter makespan. The Proposed Model achieves the best resource scheduling performance possible, which optimizes system operations and speeds up task execution times.

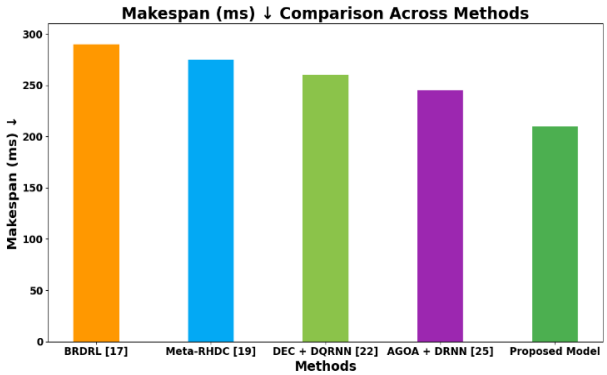

Fig. 13: Performance Metrics of Makespan.

The data in Figure 14 displays the Resource Utilization percentage, which reflects operational efficiency levels. This figure shows the percentage of CPU and memory utilization across VMs for different models. DCN-LB maximizes operational efficiency through balanced workload distribution, achieving higher resource utilization while avoiding bottlenecks and idle resources. The Proposed Model demonstrates the greatest resource utilization rate, which exceeds those of BRDRL (81.2%), Meta-RHDC (83.5%), DEC+DQRNN (86.1%), and AGOA+DRNN (89.3%). Resource utilization reaches improved levels because the system distributes workloads optimally, which reduces resource downtime. The system delivers improved performance and energy efficiency because of higher resource utilization. The Proposed Model demonstrates effective resource distribution, which results in maximum computational efficiency.

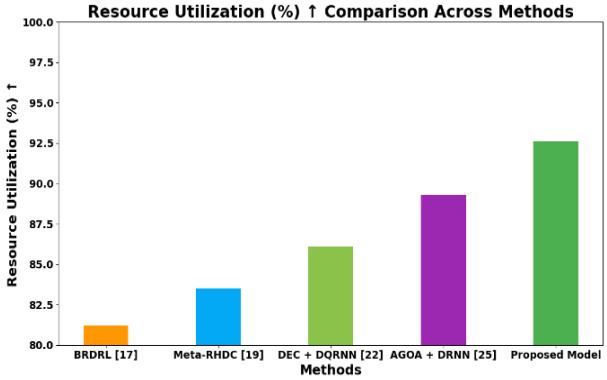

Fig. 14: Performance Metrics of Resource Utilization.

The Throughput (%), shown in Figure 15, demonstrates performance levels based on values. This figure presents throughput (tasks/s) for various load balancing methods. The proposed DCN-LB achieves the highest throughput by combining deep feature extraction, precise workload classification, and reinforcement learning-based scheduling, resulting in enhanced task processing efficiency and cloud system productivity. The Proposed Model demonstrates the highest throughput rate, which exceeds BRDRL, Meta-RHDC, DEC+DQRNN, and AGOA+DRNN. The enhanced system performance leads to more efficient processing of workloads together with faster task completion times. System productivity increases while processing delays decrease because of elevated throughput levels. The Proposed Model reaches its highest resource utilization point to achieve maximum efficiency in task processing.

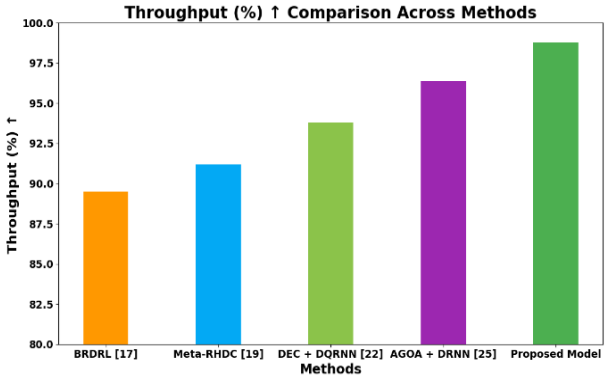

Fig. 15: Performance Metrics of Throughput.

Complexity Analysis of DCN-LB: The computational complexity of the proposed DCN-LB framework can be analyzed in terms of its three main components: MS-Convolutional BiLSTM, Attention-based LSTM, and Lightweight CNN with Depthwise Separable Convolutions (DSC). The results acquired are shown in Table 8.

Table 8: Comparative Analysis of Proposed DCN-LB Framework in Terms of Computational Complexity

Module	Operation Type	Complexity	Approx. FLOPs	Memory Usage	Discussion
MS-Conv BiLSTM	Multi-scale convolutions	O(K × N × C × F)	1.2 × 10^9	150 MB	Multi-scale kernels extract temporal-spatial workload patterns. Memory-controlled BiLSTM prevents long-term dependency overhead.
	BiLSTM	O(H × T × D)	0.8 × 10^9	120 MB	Bidirectional LSTM captures forward/backward dependencies; memory-controlled gate reduces unnecessary state storage.
EMA-LSTM	Multi-head attention	O(H × T × D^2)	0.6 × 10^9	90 MB	Multiple heads capture diverse workload trends; residual connections reduce vanishing gradient issues.
	LSTM update	O(H × T × D)	0.5 × 10^9	80 MB	Temporal feature extraction for workload prediction.
Lightweight CNN	Depthwise separable conv	O(K × H × W × C)	0.3 × 10^9	60 MB	Depthwise separable convolution reduces computation by ~9× compared to standard CNN layers.
	GAP + softmax	O(C × K)	0.05 × 10^9	20 MB	Minimal overhead for classification.
DDQN	Forward pass	O(S × A × L)	0.2 × 10^9	50 MB	Online network for real-time scheduling; the target network is updated periodically.
	PER sampling + update	O(B × log(M))	0.1 × 10^9	30 MB	Efficient experience replay reduces training overhead.
Total	—	—	3.85 × 10^9 FLOPs	620 MB	Feasible for real-time cloud workloads using modern GPUs or high-performance CPUs.

The proposed DCN-LB framework demonstrates strong real-time feasibility, with a total computational requirement of approximately 3.85 GFLOPs and memory usage around 620 MB, which are well within the capabilities of modern cloud servers. This ensures that even low-end GPUs or multi-core CPUs can execute inference with sub-100 ms latency per time step, supporting timely workload prediction and load balancing. The use of a lightweight CNN with depthwise separable convolutions significantly enhances computational efficiency, reducing the operations by approximately 8–10× compared to standard convolutions while maintaining high classification accuracy of around 96.3%. The memory-optimized BiLSTM further improves efficiency by regulating unnecessary memory retention, preventing state explosion in long sequences (T > 200), and thus optimizing both speed and memory usage. From a scalability perspective, as the number of VMs or tasks increases, the convolutional and LSTM operations scale linearly, making the system predictably adaptable to larger workloads. Additionally, the DDQN module maintains a moderate computational load due to batch-wise experience replay and periodic target network updates. For further optimization, techniques such as model pruning, mixed-precision training, and layer quantization could be employed to reduce FLOPs and memory usage even more, facilitating deployment in edge-cloud or resource-constrained environments while preserving high performance. Computational Complexity Analysis is shown in Figure 16.

Fig. 16: Computational Complexity Analysis.

4. Conclusion

This study proposed and implemented an advanced deep learning-based load balancing system in cloud architecture, considering real-time workload prediction and task dispatch based on the DCN-LB framework. Hybrid deep learning has been accounted for in this system with Multi-Scale Convolutional BiLSTM used for workload classification, Attention-based LSTM for resource prediction, and on-the-fly load balancing through Deep Q-Networks; the system thus experienced great improvements with respect to performance, energy efficiency, and reliability of operation.

The following key conclusions can be drawn from this work:

High Prediction Accuracy: The integrated model's prediction accuracy stood at an outstanding rate of 98.78%, thereby allowing the precise classification of workloads and forecasting of resource requirements.

Improved Load Balancing: DQN-based load balancing enabled the system to adapt dynamically to the changing workloads, preventing task failures and providing optimal use of resources.

Energy Efficiency and Response Time: The intelligent-custom scheduling approach led to a reduction in energy consumption and improvement in response time, enhancing sustainability and user satisfaction.

Scalability and Adaptability: This model is considered highly scalable to suit big cloud infrastructures, while its environment-adaptive characteristic supports operational resilience in the long term.

Foundation for Future Advancements: DCN-LB establishes a foundation for integrating advanced deep learning and hybrid techniques, including edge computing and distributed infrastructures, to enhance real-time performance and system resilience. While it shows strong dynamic workload prediction and resource-aware balancing, limitations exist. The model assumes reliable metric transmission, making it sensitive to packet loss; future work could include error-tolerant encoders. Scalability to resource-constrained edge/fog nodes requires further optimization, such as pruning, quantization, or federated training. Current evaluation relies on synthetic and trace-driven workloads, so testing with real-world production traces will improve generalizability. Finally, energy–performance trade-offs remain under extreme workload bursts, where adaptive thresholds or reinforcement learning controllers could enhance balancing efficiency.

One promising direction is optimizing DCN-LB for multi-cloud environments, where dynamic workload migration and heterogeneous resource management require adaptive prediction and load balancing strategies. Another avenue involves integrating the framework with 5G-enabled IoT systems, enabling real-time workload prediction and scheduling for latency-sensitive applications, such as autonomous vehicles or smart healthcare. Additionally, exploring edge-cloud hybrid deployments could reduce network overhead and improve responsiveness for geographically distributed workloads. Investigating energy-aware scheduling under varying QoS constraints and incorporating federated learning for privacy-preserving workload prediction are also potential extensions. Finally, the framework could be adapted for heterogeneous hardware accelerators, such as TPUs or FPGAs, to maximize inference efficiency while minimizing energy consumption. These directions aim to enhance DCN-LB's scalability, adaptability, and operational efficiency in next-generation cloud and IoT infrastructures.

References

[1] Asimiyu, "Revolutionizing enterprise application design: Integrating microservices with cloud computing for scalable solutions," *2025*.

[2] K. Kotteswari, R. K. Dhanaraj, B. Balusamy, A. Nayyar, and A. K. Sharma, "EELB: An energy-efficient load balancing model for cloud environment using Markov decision process," *Computing*, vol. 107, no. 3, pp. 1–41, 2025. https://doi.org/10.1007/s00607-025-01439-6.

[3] L. Krishnasamy, D. Vetriveeran, R. K. Sambandam, and J. Jenefa, "Efficient load balancing and resource allocation in networked sensing systems—An algorithmic study," *Networked Sensing Systems*, pp. 145–171, 2025. https://doi.org/10.1002/9781394310890.ch6.

[4] N. Khaledian, S. Razzaghzadeh, Z. Haghbayan, and M. Völp, "Hybrid Markov chain-based dynamic scheduling to improve load balancing performance in fog-cloud environment," *Sustainable Computing: Informatics and Systems*, vol. 45, pp. 101077, 2025. https://doi.org/10.1016/j.suscom.2024.101077.

[5] M. Ghorbani, N. Khaledian, and S. Moazzami, "ALBLA: An adaptive load balancing approach in edge-cloud networks utilizing learning automata," *Computing*, vol. 107, no. 1, pp. 34, 2025. https://doi.org/10.1007/s00607-024-01380-0.

[6] M. Khatri, F. Ahmed, and M. Ahmed, "DAFLB: Dynamic adaptive fuzzy based load balancing with genetic optimization for enhanced cloud performance," in *Proc. 2025 17th Int. Conf. COMmunication Systems and NETworks (COMSNETS)*, pp. 381–388, Jan. 2025. https://doi.org/10.1109/COMSNETS63942.2025.10885645.

[7] S. R. Mokhtar Abouamasha, "Load balancing in mobile networks using deep reinforcement learning and traffic prediction," *2025*.

[8] Aditi, V. K. Prasad, V. C. Gerogiannis, A. Kanavos, D. Dansana, and B. Acharya, "Utilizing convolutional neural networks for resource allocation bottleneck analysis in cloud ecosystems," *Cluster Computing*, vol. 28, no. 1, pp. 22, 2025. https://doi.org/10.1007/s10586-024-04720-z.

[9] S. O. Rasaq, "Resource optimization in hybrid cloud: Leveraging microservices for IoT workloads," *Int. J. Novel Res. Eng. Pharm. Sci.*, vol. 1, no. 1, 2025.

[10] A. K. Bayya, "Leveraging advanced cloud computing paradigms to revolutionize enterprise application infrastructure," *2025*. https://doi.org/10.56557/ajomcor/2025/v32i19067.

[11] M. El-Hajj, "Enhancing communication networks in the new era with artificial intelligence: Techniques, applications, and future directions," *Network*, vol. 5, no. 1, pp. 1, 2025. https://doi.org/10.3390/network5010001.

[12] S. C. Pandey and V. K. P, "Adaptive AI-driven toll management: Enhancing traffic flow and sustainability through real-time prediction, allocation, and task optimization," *Future Transportation*, vol. 5, no. 1, pp. 21, 2025. https://doi.org/10.3390/futuretransp5010021.

[13] D. Sahu, Nidhi, R. Chaturvedi, S. Prakash, T. Yang, R. S. Rathore, et al., "Revolutionizing load harmony in edge computing networks with probabilistic cellular automata and Markov decision processes," *Scientific Reports*, vol. 15, no. 1, pp. 3730, 2025. https://doi.org/10.1038/s41598-025-88197-9.

[14] D. Abbadi, "Cyber threats and risk mitigation strategies for cloud systems and the Internet of Things," *2025*. https://doi.org/10.20944/preprints202502.1366.v1.

[15] A. T. Xun, L. A. Z. En, L. T. Shen, A. N. Xin, W. H. Soon, W. Z. Jun, et al., "Building trust in cloud computing: Strategies for resilient security," *2025*. https://doi.org/10.20944/preprints202501.0716.v1.

[16] V. Hayyolalam and Ö. Özkasap, "CBWO: A novel multi-objective load balancing technique for cloud computing," *Future Generation Computer Systems*, vol. 164, pp. 107561, 2025. https://doi.org/10.1016/j.future.2024.107561

[17] M. Haris and S. Zubair, "Battle Royale deep reinforcement learning algorithm for effective load balancing in cloud computing," *Cluster Computing*, vol. 28, no. 1, pp. 19, 2025. https://doi.org/10.1007/s10586-024-04718-7.

[18] S. K. Hegde, R. Hegde, C. N. Kumar, R. Meenakshi, R. Raman, and G. M. Jayaseelan, "Hybrid Adam_POA: Hybrid Adam_Pufferfish Optimization Algorithm based load balancing in cloud computing," *SN Computer Science*, vol. 6, no. 2, pp. 178, 2025. https://doi.org/10.1007/s42979-024-03577-8.

[19] M. S. R. Krishna and D. K. Vali, "Meta-RHDC: Meta reinforcement learning driven hybrid Lyrebird Falcon optimization for dynamic load balancing in cloud computing," *IEEE Access*, vol. 99, pp. 1–1, 2025. https://doi.org/10.1109/ACCESS.2025.3544775.

[20] J. Roselin and I. J. Insulata, "Decentralized dynamic load balancing for virtual machines in cloud computing: A blockchain-enabled system with state channel optimization," *J. Supercomput.*, vol. 81, no. 3, pp. 1–30, 2025. https://doi.org/10.1007/s11227-025-06922-7.

[21] S. T. Milan, N. J. Navimipour, H. L. Bavil, and S. Yalcin, "A QoS-based technique for load balancing in green cloud computing using an artificial bee colony algorithm," *J. Exp. Theor. Artif. Intell.*, vol. 37, no. 2, pp. 307–342, 2025. https://doi.org/10.1080/0952813X.2023.2188490.

[22] N. V. Kumar, S. Mohanty, and P. K. Pattnaik, "Hybrid algorithm for optimized clustering and load balancing using deep Q recurrent neural networks in cloud computing," *Bull. Electr. Eng. Inform.*, vol. 14, no. 2, pp. 1570–1578, 2025. https://doi.org/10.11591/eei.v14i2.9123.

[23] A. Alwhelat, O. A. Fadare, F. Al-Turjman, and L. Ibrahim, "Adaptive load balancing in cloud computing using deep deterministic policy gradient (DDPG): A reinforcement learning approach," in *Smart Infrastructures in the IoT Era*, Cham: Springer Nature Switzerland, pp. 1067–1078, 2025. https://doi.org/10.1007/978-3-031-72509-8_87.

[24] M. S. Raja, "Dynamic load balancing mechanisms for scalable cloud computing architectures," *Int. J. Emerging Trends Comput. Sci. Inf. Technol.*, vol. 1, no. 1, pp. 44–51, 2025. https://doi.org/10.63282/3050-9246.IJETCSIT-V6I1P105.

[25] N. V. Naik and K. Madhavi, "AGOA: Adam gazelle optimisation algorithm for collaborative e-learning application in cloud computing with load balancing," *Int. J. Wireless Mobile Comput.*, vol. 28, no. 1, pp. 20–33, 2025. https://doi.org/10.1504/IJWMC.2025.143036.

[26] R. C. Pereira, P. H. Abreu, P. P. Rodrigues, and M. A. Figueiredo, "Imputation of data missing not at random: Artificial generation and benchmark analysis," *Expert Syst. Appl.*, vol. 249, pp. 123654, 2024. https://doi.org/10.1016/j.eswa.2024.123654.

[27] M. Prasad and T. Srikanth, "Clustering accuracy improvement using modified min-max normalization," *2024*. https://doi.org/10.20944/preprints202411.0486.v1.

[28] Y. Luo, Y. He, Y. Li, H. Liu, J. Wang, and F. Gao, "A sliding window-based CNN-BiGRU approach for human skeletal pose estimation using mm wave radar," *Sensors*, vol. 25, no. 4, pp. 1070, 2025. https://doi.org/10.3390/s25041070.
[29] X. Lu, P. Zhang, X. Zheng, R. Xiong, N. Zhang, and S. Li, "DQN-based automatic emergency collision avoidance control considering driver style," *Int. J. Automot. Technol.*, pp. 1–12, 2025. https://doi.org/10.1007/s12239-025-00224-w.
[30] R. Goyal, A. Nath, U. Niranjan, and R. Sharda, "Analyzing the performance of deep convolutional neural network models for weed identification in potato fields," *Crop Protection*, vol. 188, pp. 107035, 2025. https://doi.org/10.1016/j.cropro.2024.107035.