# Adaptive Load Balancing in Multi-Cloud Systems Using ResFedLB for Load Balancing and RL-COSCoati with Fuzzy Fault Management

Damodhar MummiReddy
Department of Computer Science and Engineering, Sri Venkateswara University, India
mdr.damodhar@gmail.com

Choda Durga Venkata Subba Rao
Department of Computer Science and Engineering, Sri Venkateswara University, India
subbarao_chdv@hotmail.com

**Abstract:** *The growing use of cloud services in the contemporary society has put unprecedented pressure on scalability, responsiveness, and fault resilience, especially in multi-cloud environments that combine heterogeneous resources across providers. Such systems have been difficult to balance their loads effectively because of workload variability, varying Service-Level Agreements (SLAs), and the necessity to allocate tasks efficiently with energy constraints of Ultra-Reliable Low-Latency Communication (URLLC). Traditional approaches, such as heuristic schedulers, metaheuristic optimizers, and Reinforcement Learning (RL)-based solutions, have provided partial solutions but still have limitations in the form of slower convergence, high task latency, high migration overhead, and poor fault-tolerant behavior. In order to address these problems, a new framework is presented. The model incorporates four significant innovations. To facilitate privacy-preserving workload forecasting, Resource-Aware Federated Learning to Load Balancing (ResFedLB) is first used. Second, CNN -LSTM -Attention with Forecasting Transformer Network (CLAFT-Net) is the specialized local training module that is used to learn temporal-spatial workload dependencies. Third, Reinforcement Learning with Coordination and Opposition Strategy in Coati (RL-COSCoati) is an adaptive and low-latency task scheduling. Lastly, CloudGuard is a fuzzy inference-based module, which guarantees proactive fault detection and resilient VM management. The framework is proven to be effective in experimental assessments with a forecasting accuracy of 99.38 and an F1-score of 98.32, and task allocation performance of 97% Central Processing Unit (CPU) utilization and 98.20% task success rate. These findings prove the hypothesis that the suggested solution provides a scalable, energy-efficient, and fault-tolerant solution to managing dynamic workloads in multi-cloud ecosystems.*

**Keywords:** *Multi-cloud load balancing, federated learning, ResFedLB, workload forecasting, task scheduling, RL-COSCoati, fault identification.*

*Received September 01, 2025*; *accepted January 11, 2026*
*https://doi.org/10.34028/iajit/23/5/11*

## 1. Introduction

The fast-paced use of cloud computing has altered the manner in which organizations implement and operate applications, moving away the conventional centralized infrastructures to the highly distributed multi-clouds. Multi-cloud setup is the combination of services of two or more cloud providers, including Amazon Web Services (AWS), Microsoft Azure, and Google Cloud, to increase flexibility, decrease vendor dependence, and maximize performance and cost efficiency Bal *et al*. [3] and Rjoub *et al*. [16]. Resource allocation and task scheduling are two important issues in such heterogeneous environments, which determine the efficient execution of computing tasks using a variety of dynamically varying resources Kruekaew and Kimpan [12] and Siddesha *et al*. [20]. The challenge is to choose the appropriate resources at the appropriate time, especially in situations of varying workloads, application requirements, and Service-Level Agreements (SLAs) Belgacem *et al*. [4].

In order to overcome these issues, scientists have turned to the possibility of Artificial Intelligence (AI) and Machine Learning (ML) to develop adaptive scheduling
and load balancing systems Swarup *et al*. [21]. Reinforcement Learning (RL) has become one of the promising paradigms in this category because it can learn the most optimal strategies by interacting with the environment and improving decisions based on trial-and-error feedback Jayanetti *et al*. [11]. Within the framework of multi-cloud systems, RL has demonstrated the ability to enhance the use of resources, lower the cost of operation, and improve the overall responsiveness and reliability through the dynamically adjusted response to the current conditions of the live systems Asghari *et al*. [2].

Traditional methods of multi-cloud task scheduling (heuristic scheduling (round-robin, first-come-first-serve), metaheuristic scheduling (genetic algorithms, particle swarm optimization, ant colony optimization), and rule-based scheduling engines) have only offered partial efficiency Zhao *et al*. [25] and Zhao *et al*. [26]. Although these approaches are easy and can be computationally cheap, they do not have the flexibility

to adapt to the dynamic and uncertain characteristics of large-scale cloud ecosystems Monon *et al*. [15]. Their inflexible decision-making processes tend to lead to inefficient use of resources, increased task latency, and reduced Quality of Service (QoS) Yan *et al*. [23].

Over the past few years, more sophisticated AI-based approaches, such as Deep Reinforcement Learning (DRL), Deep Q-Networks (DQN), and actor-critic models, have been suggested to address these deficiencies Sanjalawe *et al*. [17]. These methods have shown a better decision-making process through learning system behavior and real-time update of allocation strategies Houssein *et al*. [9]. Nevertheless, even with this promise, these approaches continue to be limited with respect to slower convergence rates, lower accuracy with rapidly changing workloads, decision-making latency, and sometimes misclassification of task priorities or resource capabilities, which can impair system performance Zhang [24].

Inspired by these constraints, this work has sought to develop a dynamic and smart load balancing system that uses the latest AI methods to deal with the dynamic nature of multi-cloud systems. The proposed solution has been designed in a way that it guarantees low-latency decision-making, effective workload distribution, energy efficiency, and improved system resilience. Although the specifics of the methodology are covered in the following sections, the general idea of this work is to introduce a lightweight, scalable, and fault-tolerant workload forecasting, task allocation, and fault-aware scheduling solution in multi-cloud ecosystems.

### 1.1. Key Contributions

- A Resource-Aware Federated Learning to Load Balancing (ResFedLB) is introduced to enable distributed workload forecasting while preserving data privacy across heterogeneous multi-cloud environments.
- A specialized local training module, (CLAFT-Net) CNN-LSTM-Attention with Forecasting Transformer Network, is developed to capture both temporal and spatial workload dependencies with high accuracy.
- A novel task scheduling strategy, RL-COSCoati (Reinforcement Learning with Coordination and Opposition Strategy in Coati), is designed to ensure adaptive, low-latency, and energy-efficient task allocation.
- A fuzzy inference-based CloudGuard fault identification and Virtual Machine (VM) management mechanism is integrated to proactively detect, classify, and mitigate faults, enhancing system resilience.

### 1.2. The organization of this work

The literature review based on the current work is found in Section 2. The system model, comprising the framework and key elements, is presented in Section 3. The result and comparison analysis are shown in Section 4. Finally, the work comes to a close in the Section 5 with the conclusion.

## 2. Literature survey

In 2023 Shiva Rama Krishna and Mangalampalli [19] determined the VM work priority based on the cost of electricity per unit, and the scheduler receives these rankings. When the proposed Deep Reinforcement Fault-Tolerant Task Scheduling Algorithm (DRFTSA) is compared to current state-of-the-art methods, such as Ant Colony Optimization (ACO), Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) algorithms, the results show that it reduces makespan by 35.1%, 37.12% and 30.97% failure rates by 44.13%, 46.19% and 39.4% and energy consumption by 23.07%, 28.8% and 18.81% respectively, for both large and regular datasets for both varied and fixed VMs.

In 2023 Danino *et al*. [7] proposed a novel method for allocating containers to cloud servers that satisfies the needs for communication and processing power while reducing the total time needed to complete tasks. The effectiveness of memory-augmented and Long Short-Term Memory (LSTM)-based agents in resolving the difficult container assignment problem is investigated. According to experimental data, the execution runtime was improved by up to 28% when compared to the popular Kubernetes industrial tool and current bin-packing algorithms.

In 2024 Chen *et al*. [6] suggested a joint optimization algorithm based on RL. Lastly, the simulation results show that the suggested algorithm enhances the system's overall energy efficiency by 16.23%, 86.29%, and 5.11%, respectively, under a variety of conditions (such as varying noise power levels, GPF bandwidth, and GPF quantities), in comparison to the three baseline algorithms (random path, average transmit power, and random device association).

In 2024 Lahza *et al*. [13] established a Multi-Objective Resource Optimization (MORO) scheduler for various urban setups, the deficiencies in edge-cloud work are addressed. The simulation results show that MORO Over-Reliability Enhancement Strategies for Workflow Scheduling (RESWS) and Multi-Objective Priority Workflow Scheduling (MOPWS) for Searching in Parallel Helps to Find Targets (SIPHT) workflow consistently improve the workflow, with average makespan enhancements of 31.6% and 70.09%, average cost reductions of 62.64% and 73.24%, and average energy consumption reductions of 25.02% and 17.77%, respectively.

In 2025 Alatawi [1] suggested an adaptive resource allocation paradigm based on RL to deal with these problems. With a 50% decrease in latency (from 250 ms to 120 ms), a 38.9% increase in throughput (from 180 tasks/s to 250 tasks/s), and a 35% gain in energy

efficiency, the suggested model outperforms heuristic methods.

These findings highlight the promise of RL-based approaches to dynamic workload management, paving the path to serverless multitenancy systems that are more scalable, economical, and long-lasting.

In 2025 Sharma and Patel [18] soughted to investigate clever algorithms that adapt effectively towards changing workloads in order to increase task scheduling efficiency. The outcomes demonstrate how effective weighted-based and RL techniques perform. With a 44% decrease in average slowness and a 50% improvement in work completion time, RL saw the most improvements. Under actual cloud conditions, both RL and weighted-based approaches demonstrated better consistent and adaptive scheduling when compared to Fengcun Li's stated Tetris and First-Fit performance.

In 2025 Wang and Yang [22] suggested a clever resource allocation technique that uses RL Deep Q-Network (DQN) for dynamic scheduling and deep learning LSTM for demand prediction. The suggested solution improves resource usage by 32.5%, decreases average reaction time by 43.3%, and lowers operating expenses by 26.6% by precisely predicting computing resource demands and allowing for real-time modifications. Results from experiments conducted in a live cloud environment verify that the technique maintains high service quality while greatly increasing efficiency.

In 2021 Chen *et al*. [5] suggested a DRL-based adaptive and effective cloud resource allocation system. Chen used actual data from Google cloud datacenters to perform comprehensive simulation analysis. The findings demonstrate that, when compared to two sophisticated DRL-based and five traditional cloud resource allocation techniques, our approach can achieve the highest QoS in terms of latency and job dismissing rate with improved energy efficiency.

In 2023 Li *et al*. [14] suggested TapFinger, a distributed scheduler for edge clusters that uses fine-grained multi-resource allocation and task placement co-optimization to reduce the overall time needed to complete ML tasks. Our design can quickly converge to the best scheduling solutions while mitigating the enlarged decision space. In comparison to state-of-the-art schedulers, TapFinger can reduce the average task completion time by up to 54.9% and enhance resource efficiency, according to extensive experiments utilizing synthetic and test-bed ML task traces.

In 2025 Elrod *et al*. [8] suggested DRL-GNN, an innovative framework for improved multi-agent cooperation and collective task execution that combines transformer-based processes, DRL, and Graph Neural Networks (GNNs). In comparison to benchmark approaches like PSO, greedy algorithms, and DQN, experimental results show higher performance, with 90% service provisioning and 100% grid coverage node discovery while lowering the average steps per episode to 200 from 600.

### 2.1. Problem Statement

The recent works in the field of multi-cloud task scheduling and resource allocation have shown that heuristic, metaheuristic and AI-based methods have the potential to enhance the performance of a system. Conventional algorithms like Genetic Algorithm (GA), PSO and Ant Colony Optimization (ACO) have been useful in minimizing makespan and energy usage but are still restricted in their flexibility especially in dynamic workloads and heterogeneous resource conditions [19]. Memory-augmented agent-based and LSTM-based container assignment approaches have enhanced execution runtime, but are limited by scalability concerns and reliance on workload patterns [7]. RL-based systems, such as joint optimization, DQN, actor-critic, and DRL-GNN systems, have provided substantial throughput, latency reduction, and resource efficiency benefits [6, 8]. Nevertheless, they tend to be slower to converge, have long decision latency, sensitive to workload variations, and have high migration overheads, making them impractical to deploy in ultra-reliable Ultra-Reliable Low-Latency Communication (URLLC) environments.

Nevertheless, the literature shows that the current models cannot simultaneously provide high forecasting accuracy, fault tolerance, energy efficiency, and low-latency scheduling in multi-cloud environments. This restriction generates an acute necessity of a new solution that is not only dynamically adjusted to the changes in the workload but also includes fault-conscious decision-making, energy-saving, and federated learning features. Inspired by these loopholes, the current work presents a lightweight, resilient, and intelligent framework that combines forecasting, adaptive scheduling, and fault detection to guarantee efficient, scalable, and SLA-compliant execution of tasks in heterogeneous multi-cloud ecosystems.

## 3. System Model

In this section, the general system model of the proposed multi-cloud load balancing framework is provided. It starts with a formal definition of the problem and its formulation (Section 3.1), then the description of the main components, such as workload forecasting, federated learning-based resource prediction, task scheduling, execution management, VM consolidation, cross-cloud communication, and fault identification using CloudGuard. All the stages are aimed at solving the problem of heterogeneous multi-cloud environments by providing the correct forecasting, adaptive scheduling, energy-efficiency, and fault resilience. Lastly, the section is concluded with task completion (Section 3.10), which is a combination of all the mechanisms mentioned above into a single pipeline to ensure reliable, low-latency, and SLA-compliant task

completion. Figure 1 represents the overall system model structure.

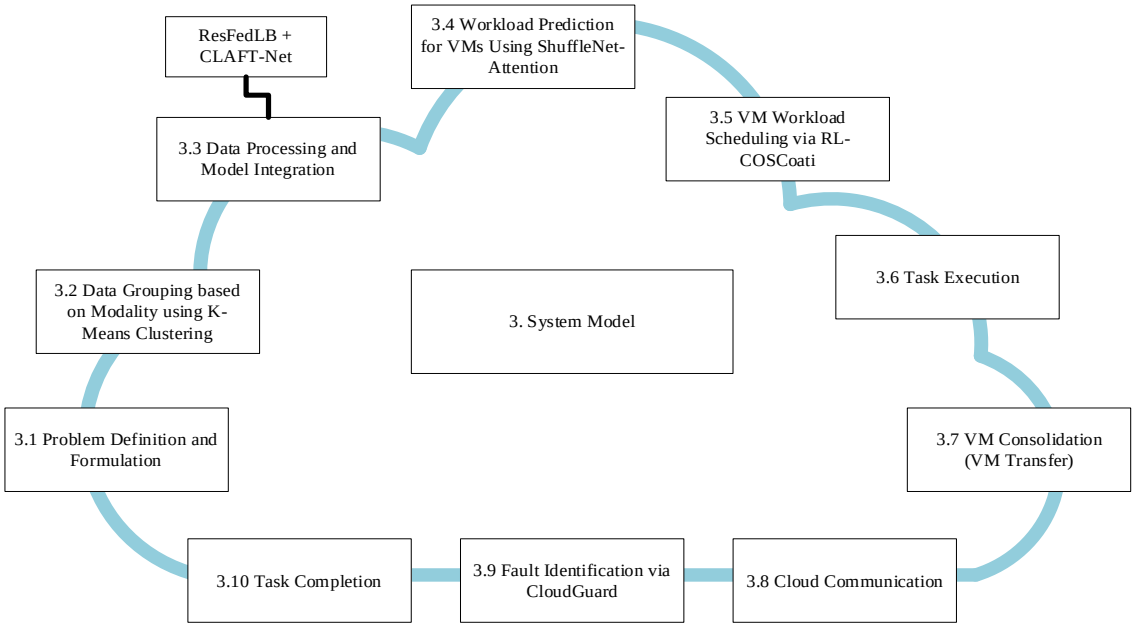


Figure 1. System model structure.

## 3.1. Problem Definition and Formulation

Load balancing is one of the key features of multi-cloud environments to address the issues of heterogeneous task management, optimal resource use, and the ability to comply with the requirements of Ultra-Reliable URLLC. In contrast to single-cloud deployments, multi-cloud platforms are required to coordinate the various SLAs, variable resource capacities, and dynamic user demands across multiple providers simultaneously.

### 3.1.1. Cloud

A cloud is a virtualized computing platform that offers on-demand services including computing, storage and networking via the internet. In this work, a multi-cloud infrastructure, where workloads are spread across cloud platforms (AWS, Azure, Google Cloud). Every cloud is composed of physical resources that are abstracted into virtualized objects and tasks are processed without any issues of the underlying hardware.

A Cloud Service Provider (CSP) provides infrastructure and services that allow users to implement and run tasks on the cloud. Every CSP possesses its pricing schemes, service policies, and resource settings, which add to the heterogeneity of the multi-cloud ecosystem. Load balancing in our problem environment should not only allocate tasks efficiently, but also consider these differences between CSPs to achieve cost-effective and QoS-compliant execution.

### 3.1.2. Input Tasks

End-users, denoted as $U_i$, generate heterogeneous input tasks that must be processed in the multi-cloud environment. These tasks include:

- Audio data-speech recognition, acoustic signal processing.
- Video data-real-time video streaming, surveillance analytics.
- Electroencephalography (EEG)-biomedical signals requiring precise temporal processing.
- Numerical data-tabular datasets for analytics, financial modeling, or IoT logs.
- Image Data-medical imaging, object recognition, computer vision workloads.
- Electrocardiography (ECG)-biomedical signals for patient monitoring and anomaly detection.

The set of tasks submitted by a user can be represented as Equation. (1):

$$\mathcal{U}_i = \{\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_j\} \tag{1}$$

Each task $T_j$ is defined by its resource demand vector as Equation. (2):

$$\mathcal{T}_j = (\alpha_j, \beta_j, \gamma_j, \delta_j) \tag{2}$$

Where, $a_j$ denotes Central Processing Unit (CPU) cycles, $B_j$ denotes memory requirements, $y_j$ denotes bandwidth needs, and $\delta_j$ represents the latency/deadline constraint.

### 3.1.3. Physical Machines (PMs)

A Physical Machine (PM) is the hardware core of a cloud provider. Each PM is limited to a certain computational power, memory, and network capacity. The multi-cloud system can be described as the set of physical machines, which is formally expressed as Equation. (3):

$$\mathcal{C} = \{PM_1, PM_2, \dots, PM_n\} \tag{3}$$

Depending on their hardware configurations, each PM can carry several VMs. The VMs count per PM is

subject to resource partitioning policies and elastic scaling mechanisms set by the CSP.

### 3.1.4. Virtual Machines (VMs)

A VM is basically a division of a PM done logically through the aid of virtualization technologies (Xen, Kernel-based Virtual Machine (KVM), VMware). As a result, each VM becomes an individually isolated execution environment having its own CPU, memory, storage, and network bandwidth. For the $n^{th}$ physical machine, the set of hosted VMs can be represented as Equation. (4):

$$PM_n = \{VM_1, VM_2, \ldots, VM_m\} \quad (4)$$

VMs act as the smallest resource units in load balancing, which are responsible for the execution of user tasks. A proper distribution of the user load among VMs results in a scenario, where the machines are not overburdened, and the available infrastructure is utilized to a high extent. The suggested load balancing model is depicted in Figure 2.

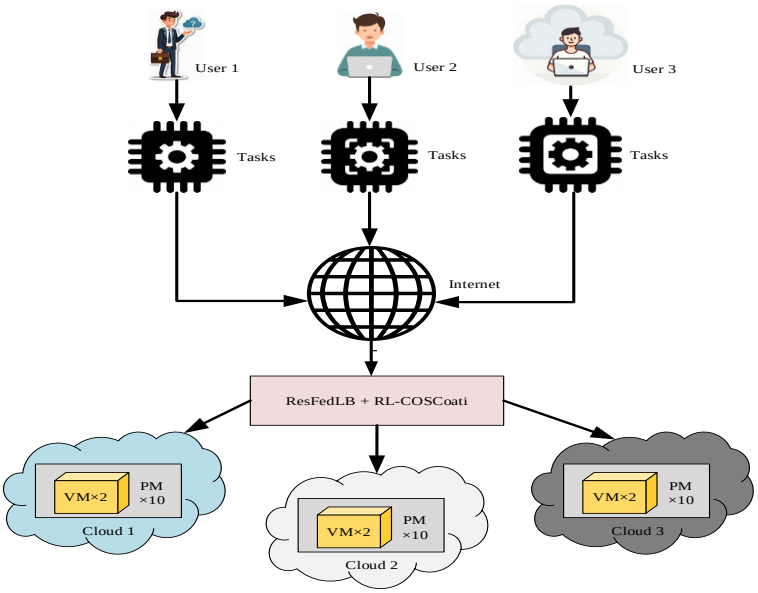


Figure 2. Developed load balancing model.

### 3.1.5. Problem Formulation

The core challenge is to allocate tasks from multiple users to VMs across multi-cloud environments while satisfying the following objectives:

- Minimize energy consumption-by reducing idle resource usage.
- Minimize operational costs-by leveraging CSP-specific pricing models.
- Maximize resource utilization-ensuring efficient distribution across VMs.
- Ensure QoS compliance-particularly for URLLC-driven workloads such as real-time video or biomedical signals.
- Maintain load balance across clouds-avoiding resource hotspots and bottlenecks.

Without a robust load balancing mechanism, the multi-cloud environment suffers from execution delays, energy inefficiencies, cost inflation, and degraded QoS guarantees.

## 3.2. Data Grouping based on Modality using K-Means Clustering

The initial stage of the suggested model is about gathering and preprocessing the unprocessed data that comes from various kinds of sources. Data in a multi-cloud environment is usually coming from different areas such as audio, video, EEG, ECG, numerical/tabular logs, and image-based applications. Because these inputs are inherently different, handling them directly without some sort of pre-processing will lead to the inefficient use of cloud resources.

To tackle this problem, a K-means clustering method is used by the system to categorize the incoming data based on the modality and relevance. Consequently, the tasks associated with the same modality are grouped together, which makes it possible to use them more effectively for the subsequent training and prediction phases. Let the dataset be denoted as Equation. (5):

$$D = \{d_1, d_2, \ldots d_n\} \quad (5)$$

Where, each data instance $d_i$ is characterized by feature vectors extracted from its modality (spectral features for audio, pixel intensities for images, temporal series for EEG/ECG). The K-means algorithm partitions the dataset into $K$ modality-specific clusters as Equation. (6):

$$D \rightarrow \{C_1, C_2, \ldots C_K\}, C_k \cap C_{k'} = \emptyset, \forall k \neq k' \quad (6)$$

Here, $C_k$ represents the cluster corresponding to one modality $C_1$= audio tasks, $C_2$= image tasks, etc.

The objective function of K-means minimizes the sum of squared distances between each data point and the centroid of its assigned cluster as $\min \sum_{k=1}^{K} \sum_{d_i \in C_k} \| d_i - \mu_k \|^2$. Where $\mu_k$ is the centroid of cluster $C_k$.

The output of this stage is a set of modality clusters that are prepared for distributed learning in the federated framework. These clusters act as structured inputs to subsequent steps, ensuring that resource-intensive workloads are mapped according to their computational signatures. Figure 3 represents the number of cloud and data utilized.

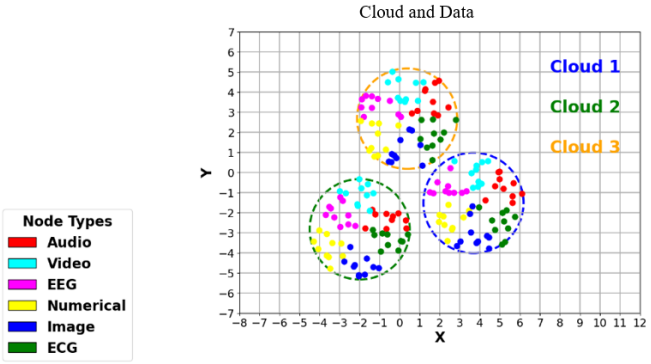


Figure 3. Number of cloud and data.

The grouping of data is graphically represented in Figure 4.

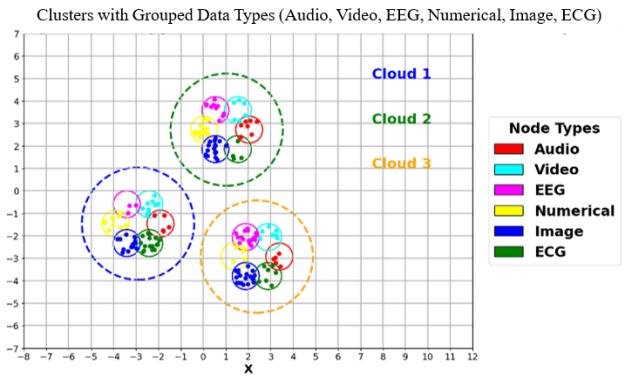


Figure 4. Grouping of data.

## 3.3. Data Processing and Model Integration

The grouped tasks $\{C_1, C_2, \ldots C_K\}$ serve as structured inputs for resource forecasting and workload balancing following the modality clustering stage described in Section 3.2. Each cluster corresponds to a specific task modality (audio, video, EEG, image, ECG, numerical), enabling modality-aware learning. At this stage, introduces the core novelty of this work: A Federated Learning-based multi-cloud processing framework called ResFedLB, with a specialized local training module termed CLAFT-Net.

The primary goal of this stage is to predict resource utilization patterns associated with each task, defined as Equation. (7):

$$\Phi(\mathcal{T}_j) = \{CPU(\mathcal{T}_j), RAM(\mathcal{T}_j), E(\mathcal{T}_j), M(\mathcal{T}_j), C(\mathcal{T}_j), R(\mathcal{T}_j), L(\mathcal{T}_j)\} \quad (7)$$

Where, $CPU(T_j)$ denotes the expected CPU utilization for task $T_j$, $RAM(T_j)$ signifies the expected memory consumption, $E(T_j)$ represents the energy consumption, $M(T_j)$ reflects the makespan completion time, $C(T_j)$ denotes the cost incurred, $R(T_j)$signifies the response time, $L(T_j)$ presents the latency constraint. These forecasted metrics form the foundation for intelligent load balancing decisions in later stages. The number of data mention in cloud clusters is depicted in Figure 5.

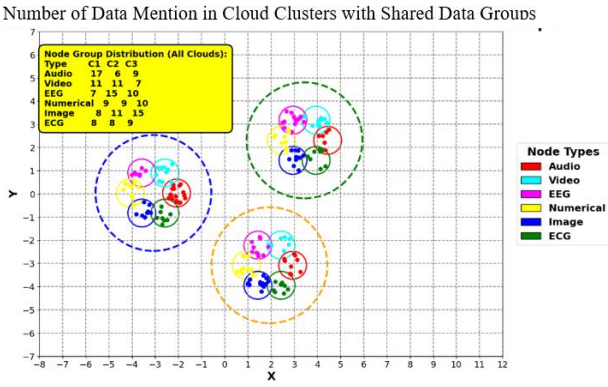


Figure 5. Number of data mention in cloud clusters.

### 3.3.1. ResFedLB: Resource-Aware Federated Learning Framework

Conventional cloud load balancing mechanisms are based on a centralized model training which leads to increased data transfer costs, raises privacy concerns, and results in high communication latency. On the other hand, the ResFedLB framework that is introduced carries out distributed learning among several CSPs without any need for raw data transfer between nodes.

In ResFedLB, each CSP (denoted as $CSP_i$) trains a local model $M_i$ using its own modality-specific task clusters, and only the model parameters (weights) are shared with the global aggregator. The global model $M_g$ is updated iteratively as Equation. (8):

$$M_g^{(t+1)} = \sum_{i=1}^{N} \frac{|D_i|}{\sum_{j=1}^{N} |D_i|} M_i^{(t)} \quad (8)$$

Where, $|D_i|$ represents the size of the dataset at CSP $i$, and $N$ signifies the total number of participating CSPs. This ensures fairness in aggregation and avoids bias toward larger datasets.

The key novelty of ResFedLB lies in its resource-awareness, meaning that the global model is optimized not just for prediction accuracy, but also for forecasting task-resource mappings $\Phi(T_j)$ in multi-cloud environments. The architecture of ResFedLB based load forecasting model is represented in Figure 6.

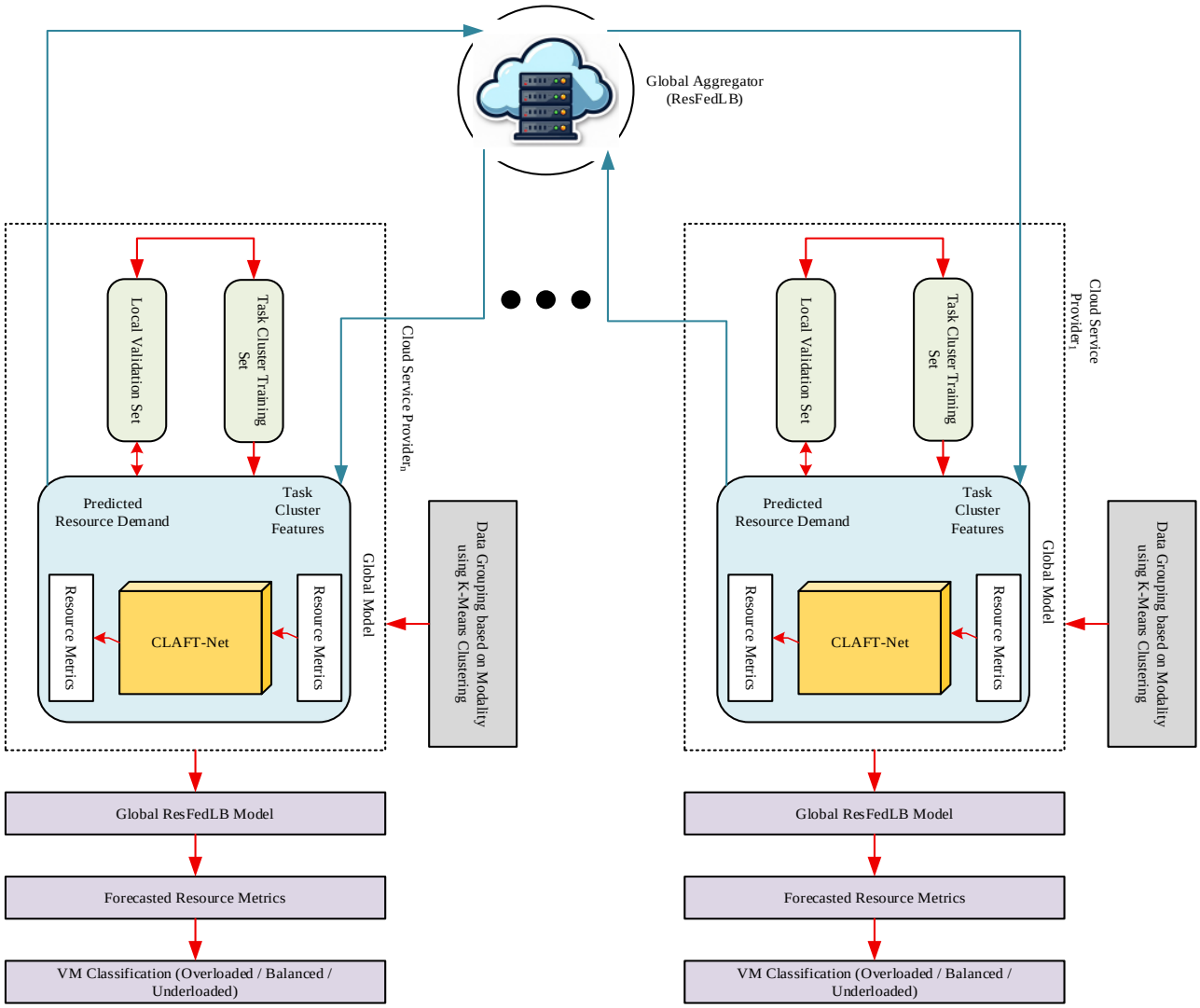


Figure 6. Architecture of ResFedLB based load forecasting model.

### 3.3.2. CLAFT-Net: Local Training Module

Each CSP trains its local model using the proposed CLAFT-Net, which integrates the strengths of multiple deep learning paradigms:

### 3.3.3. Convolutional Neural Network (CNN) Layer

The CNN layer constitutes the first critical stage within the CLAFT-Net framework, responsible for extracting low- to mid-level spatial representations from high-dimensional multimodal data such as video frames, EEG signal topographies, ECG waveform structures, and medical image textures. CNNs are particularly effective in identifying localized patterns through convolution operations, which act as learnable filters that slide across the input space, which is defined as Equation. (9).

$$f_{CNN} = \sigma_{na}(W_{CNN} * d_i + b_{CNN}) \quad (9)$$

The input data instance $d_i$ (a segment of an EEG time-series mapped into 2D space or a frame from surveillance footage) is convolved with a kernel $W_{CNN}$, where*denotes the convolution operator. The convolution emphasizes local correlations such as edges, textures, or waveform transitions, which are frequently too faint for dense feedforward layers to detect directly. The bias term $b_{CNN}$ adds versatility to the activation threshold, whereas the non-linear activation function $\sigma_{na}(\cdot)$ GELU guarantees that the learned representation is capable of modeling complicated, non-linear dependencies.

The CNN layer in the ResFedLB+CLAFT-Net pipeline acts as the first extractor, merging multimodal input into small embedding's that keep the structural aspects of the data. This operation guarantees that unnecessary noise is filtered out and that the spatial/temporal order is maintained, thus giving proper descriptors for the following attention.

### 3.3.4. LSTM Layer

The LSTM Layer extends the features extracted by the CNN by looking at the temporal side of the data. Although CNNs are good at capturing localized spatial patterns, they cannot model sequential dynamics, which are very important in multimodal IoT and edge environments. As an example, physiological signals like

EEG and ECG are totally dependent on time and subtle changes in the temporal windows indicate abnormalities. Likewise, in network traffic streams or sensor data, anomalies be going under the radar in the face of single observations but they could be the evolving trends of several time steps. To address this, the LSTM unit is integrated as Equation. (10).

$$h_t = LSTM(f_{CNN}, h_{t-1}, c_{t-1}) \tag{10}$$

Here, $f_{CNN}$ represents the spatially compressed features obtained from the CNN, $h_{t-1}$ signifies the previous hidden state, and $c_{t-1}$ denotes the previous cell state. The LSTM uses this triplet to change the current hidden state $h_t$, thus it is able to capture long-range temporal dependencies and at the same time it does not suffer from the vanishing gradient problem that is usually the case with traditional Recurrent Neural Networks (RNNs).

The LSTM layer in ResFedLB+CLAFT-Net pipeline enhances the features that are extracted by CNN through understanding temporal changes, managing variable sequence lengths, and connecting short- and long-term dependencies. As a result, the system can identify the changing patterns that are essential for accurate time-dependent tasks.

### 3.3.5. Attention Mechanism

The attention layer allows the sequential representation to be more detailed by choosing which features are the most important from the time or modality dimensions. In general, this is the key factor in a task such as Ultra-Reliable URLLC, where the most significant identification of latency-sensitive features leads to more precise prediction and faster decision-making.

At each timestep *t*, the attention score $e_t$ is computed as Equation. (11), a compatibility function between the current hidden state $h_t$ and the previously aggregated context $s_{t-1}$.

$$e_t = \alpha_v^{\mathrm{T}} \tan h(W_h\, h_t + W_s s_{t-1}) \tag{11}$$

Where, $W_h$ and $W_s$ represents learnable weight matrices for hidden state and context alignment, respectively. $a_v$ denotes the attention vector that projects the compatibility score into a scalar, and *tan h*(·). The normalized attention weight $a_w$ is then obtained through a softmax operation as Equation. (12):

$$\alpha_w = \frac{\exp(e_t)}{\sum_{k=1}^{T} \exp(e_k)} \tag{12}$$

This is done to make sure that all the attention weights over the sequence add up to 1, thereby showing the relative importance of each timestep. In the end, the context vector $a_v$ is obtained by the weighted sum of hidden states, which is mathematically expressed as Equation. (13).

$$c_v = \sum_{k=1}^{T} \alpha_k h_k \tag{13}$$

This context vector $c_v$ is like a sophisticated representation that highlights the most important signals for the task (anomalies in EEG, packet delays in IoT streams, or latency hotspots in 5G URLLC). The attention mechanism, which is to a large extent responsible for the dynamic reweighting of temporal dependencies, works together with the LSTM layer and the overall system becomes more adaptable and interpretable when dealing with sequences.

### 3.3.6. Forecasting Transformer Layer

Forecasting Transformer Layer makes the prediction more accurate by being able to capture relations over a long range and dependencies between different modalities. The attention mechanism basically figures out which features of the task are most relevant at the given time step, but the transformer layer is even further by actually being able to model global interactions not only across temporal dimensions but also across heterogeneous modalities (Network States, QoS Metrics, and Application-Layer Signals). The combination of these two capabilities basically allows short-term fluctuations Sudden Traffic Bursts (STB) and long-term dependencies Periodic Workload Patterns (PWP) to be taken into account jointly in the forecasting process. Formally, the forecasting representation at time step *t* is expressed as Equation. (14):

$$z_t = Transformer(h_t, \alpha_w) \tag{14}$$

Here, $h_t$ represents the hidden feature state derived from the encoder, while $a_W$ signifies the attention weight vector obtained in Equation. (12). The transformer block combines these inputs via a Multi-Head Self-Attention (MHSA) mechanism, which calculates several parallel attention distributions in order to capture different dependency structures across the sequence. Each head learns a different representation space; thus, the model is able to simultaneously capture correlations such as:

- Intra-modality dependencies (packet arrival rate vs. latency trends), and
- Cross-modality dependencies (the influence of mobility patterns on communication reliability).

The self-attention operation within the transformer is defined as Equation. (15):

$$SA(qu, ke, va) = Softmax\left(\frac{quke^{\mathrm{T}}}{\sqrt{d_{ke}}}\right) va \tag{15}$$

Where, query $q_u$, key $ke$, and value $v_a$ are linear projections of [$h_t$, $a_W$], and $d_k$ denotes the dimension of the key vector. This formulation enables the model to re-weight contextual relationships dynamically, rather than relying solely on static temporal dependencies.

In order to make the learning process more stable and keep the order of tasks, positional encodings are incorporated into the input sequence preceding the attention calculation. These encodings offer a smooth

representation of the positions of the time-steps, thus allowing the transformer to distinguish between the features coming from the first and the last stages of the task evolution.

Eventually, the results of all attention heads are combined and given to a feed-forward projection layer through residual connections and layer normalization. This makes strong feature propagation possible and also vanishing gradients are alleviated. So, the final vector $z_t$ stands for a deep contextual embedding that integrates both attention-guided local feature importance and transformer-captured global dependencies.

The final output of CLAFT-Net for task $T_j$ is the predicted resource demand vector, which is expressed as Equation. (16):

$$\hat{\Phi}(\mathcal{T}_j) = f_{CLAFT}(d_i) \tag{16}$$

In the ResFedLB+CLAFT-Net workflow, modality-based clusters generated in Step 3.2 are first assigned to CSPs. Each CSP then performs local training using CLAFT-Net on its respective task clusters, producing forecasts denoted as $\hat{\Phi}(\mathcal{T}_j)$. The parameters trained locally are later combined via the ResFedLB framework, refer to Equation (4), to build a single global model that reflects the workload distribution patterns over several clouds. Afterward, the global model is used to predict the essential resource metrics such as CPU, RAM, energy, cost, makespan, response time, and latency. At last, these estimations are turned into the next step of the pipeline, i.e., workload classification (Step 3.4), which makes it possible to assign the virtual machines into three classes: overloaded, underloaded, and balanced ones. The resource forecasting is graphically provided in Figure 7.

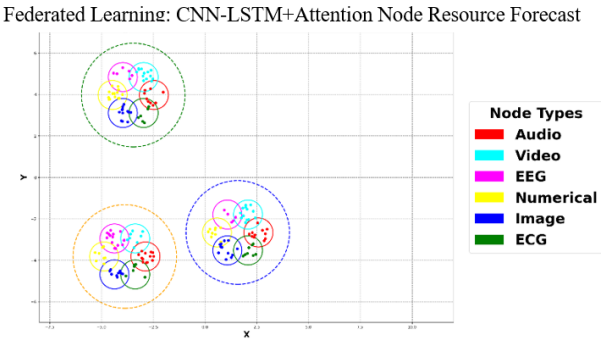


Figure 7. Resource forecasting.

## 3.4. Workload Prediction for Virtual Machines (VMs) Using ShuffleNet-Attention

After the local training of CLAFT-Net and the globally coordinated resource allocation strategy powered by ResFedLB, the very next step is to look at provisioning and scaling the VMs based on the anticipated workload. Although Section 3.3 briefly described a federated learning-based optimization framework for workload allocation, the current section concentrates on workload forecasting, which is, in fact, an indispensable component of the deep learning-based approach utilizing ShuffleNet with channel-spatial attention mechanisms for precise workload trend analysis.

ResFedLB's output generates resource usage distributions for various tasks, which include CPU utilization, RAM consumption, energy cost, makespan, and latency per task. But these resource allocations require time-series forecasting to be able to predict workload surges or drops in advance. Through forecasting workloads for each VM individually, the system is able to categorize VMs into three operational clusters:

- Overloaded VMs (high demand, risk of SLA violations)
- Underloaded VMs (low utilization, wasted resources)
- Balanced VMs (optimal resource usage)

This clustering ensures dynamic scaling of CPU, memory, and storage to maintain both efficiency and cost-effectiveness. The Figure 8 represents three clouds with PM and VM (VM numbers shown).

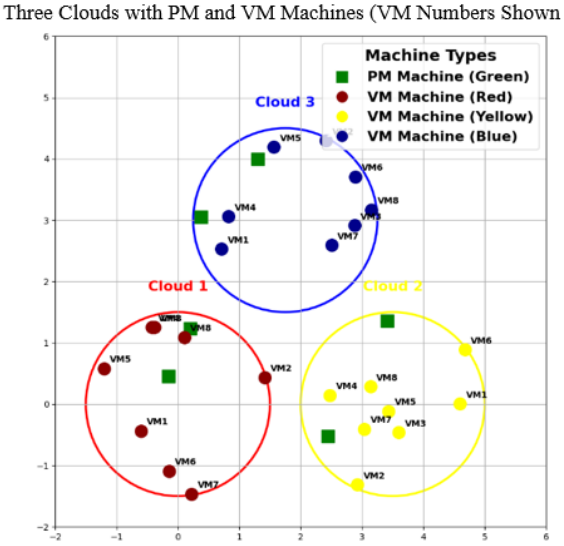


Figure 8. Three clouds with PM and VM.

### 3.4.1. ShuffleNet-Attention Framework

Combine a ShuffleNet backbone with an attention refinement module to get precise and efficient workload forecasting. The reason why ShuffleNet is selected is its minimalistic nature and grouped convolution method, which lowers the computational complexity by a large factor without losing the representational power. This feature is what makes it a perfect fit for extensive multi-cloud scenarios, where models need to be scalable and still have low latency while a large number of VM workload data is dealt with.

Let the historical workload sequence of a VM be represented as $W_{VM} = \{w_1, w_2, \ldots, w_t\}$, where $w_t$ denotes the observed workload at time step $t$ (CPU utilization, memory consumption, or task arrival rate). Each workload input is first transformed into an embedding

through ShuffleNet feature extraction blocks, expressed as Equation. (17):

$$F_t = ShuffleNet(w_t; \theta_s) \tag{17}$$

Here, $\theta_s$ represents the ShuffleNet parameters, and $F_t$ signifies the compact feature embedding corresponding to time step $t$. ShuffleNet makes sure that features are both computation efficient and semantically rich by using pointwise group convolutions and channel shuffling, thus capturing subtle workload variations over time.

To elevate these features further, use a dual attention mechanism that includes both channel attention and spatial attention. With this, the model gets a chance to selectively highlight the most informative temporal changes while wiping out the less useful noise. The refined feature signal for each time step is calculated by Equation. (18):

$$\hat{F}_t = \alpha_c \cdot F_t + \alpha_s F_t \tag{18}$$

Where, $a_c$ and $a_s$ represents the adaptive channel and spatial attention weights, respectively. The attention module varies these weights dynamically during training, thereby allowing the model to concentrate on essential changes in the workload, for instance, a sudden increase in CPU usage or an unusual memory spike, which are reliable signs of the forthcoming load imbalance.

Finally, the refined features are passed through a fully connected forecasting layer, which outputs the predicted workload for the next time step, which is defined as Equation. (19):

$$\hat{w}_{t+1} = f_p(\hat{F}_t; \theta_p) \tag{19}$$

Here, $f_p$ denotes the regression layer parameterized by $\theta_p$, responsible for mapping attention-refined embedding's into workload forecasts. The predictions made as a result offer a lead indication of the resource consumption trends through which the system is enabled to segregate the VMs as overloaded, underloaded, or balanced ones in the following stages of the workflow.

By leveraging the efficiency of ShuffleNet and the discriminative power of attention, this framework is able to strike a compromise between a lightweight scalable solution and extremely accurate forecasting, thereby turning it into a very potent tool for the real-time management of the workload in a multi-cloud environment.

### 3.4.2. VM Clustering Based on Predicted Workload

Once the workloads are predicted, VMs are categorized into three clusters, which is expressed in Equation. (20):

$$C(VM) = \begin{cases} Overloaded, & \hat{w}_{t+1} > \tau_{high} \\ Underloaded, & \hat{w}_{t+1} < \tau_{low} \\ Balanced, & \tau_{low} \le \hat{w}_{t+1} \le \tau_{high} \end{cases} \tag{20}$$

Here, $\tau_{low}$ and $\tau_{high}$ are empirically determined thresholds. The workload prediction of VM is graphically provided in Figure 9.

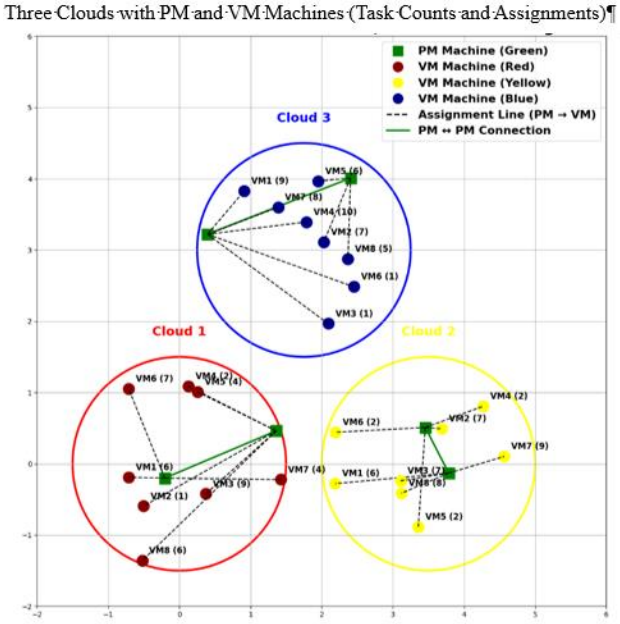


Figure 9. Workload prediction of VM.

## 3.5. VM Workload Scheduling via RL-COSCoati (Reinforcement Learning with Coordination and Opposition Strategy in Coati)

Efficient workload scheduling is a fundamental constituent in both cloud and edge environments, whose challenges reside in dynamic resource demands, heterogeneous VMs, and fluctuating workloads, which may substantially degrade system performance. This work proposes RL-COSCoati, a hybrid scheduling framework that integrates RL-based coordination with an opposition-learning coati optimization strategy. Accordingly, RL-COSCoati empowers adaptive decision-making by leveraging RL to learn scheduling policies from prior system states and feedback, whereas Opp-Coati offers enhanced global exploration and fast convergence towards near-optimal solutions. All these together facilitate balanced workload distribution, reduced execution latency, improved energy efficiency, and cost-aware resource allocation.

The detailed stepwise workflow of RL-COSCoati begins with input formulation and problem formulation, then is followed by the RL-based coordination, optimization via opp-coati, and finally the generation of scheduling decisions.

### 3.5.1. Input State Acquisition

Continuing from Section 3.4, once the VMs have been forecasted and categorized into three clusters based on their predicted workloads as Equation. (20). Where, $\tau_{low}$ and $\tau_{high}$ are empirically determined thresholds, the input state for the RL-COSCoati scheduler is constructed.

The state vector $S_t$ at time step $t$ for each VM

incorporates the metrics as expressed in Equation. (21):

$$S_t^{(VM)} = \begin{bmatrix} CPU_t, RAM_t, Latency_t, \\ QueueLength_t, Energy_t \end{bmatrix} \quad (21)$$

Where, $CPU_t$ and $RAM_t$ represent the current utilization of the VM, $Latency_t$ captures the response time of the VM, $QueueLength_t$ represents the number of pending tasks in the VM, and $Energy_t$ denotes the cumulative energy consumed by the VM.

This input state acquisition is a key factor to the RL agent to have a whole picture of VM performance and load status, thus the most suitable scheduling decisions. The VMs that are overloaded have been noted as those that cannot be given new tasks, the underloaded VMs as the first ones to get new tasks, and the balanced VMs as those which can be used depending on the condition. After that, this organized input state is being used by the Parallel Policy Generation Stage, which is the main part of RL-COSCoati's scheduling method.

### 3.5.2. Parallel Policy Generation Stage

Once the input state $S_t$ for each VM is acquired (Section 3.5.1), the RL-COSCoati scheduler generates candidate task scheduling policies through a dual-branch mechanism to balance exploitation and exploration. This stage consists of RL coordination policy and opposition-based learning.

Branch 1: RL Coordination Policy

In the first branch, a Markov Decision Process (MDP)-based RL agent generates a candidate scheduling vector $X_{RL}$ representing proposed task-to-VM allocations, which is defined as Equation. (22):

$$X_{RL} = f_{RL}(S_t; \theta_{RL}) \quad (22)$$

Where, $\theta_{RL}$ denotes the parameters of the RL agent. The RL agent considers:

- VM cluster status (Underloaded /Balanced/ Overloaded)
- Real-time resource metrics (CPU, RAM, latency, energy, queue length)
- Task-specific constraints (SLA, latency requirements)

The coordination mechanism ensures that candidate allocations are globally consistent across multiple CSPs. This prevents scenarios where multiple CSPs over-allocate tasks to the same VMs, thereby avoiding bottlenecks and SLA violations.

Branch 2: Opposition-Based Learning

To enhance exploration and prevent premature convergence to local optima, the second branch generates an opposite candidate vector $\tilde{X}_{RL}$ as expressed in Equation. (23):

$$\tilde{X}_{RL} = Opposite(X_{RL}) \quad (23)$$

The reverse vector corresponds to a different scheduling setup which essentially flips the allocation decisions (assigning tasks to VMs that were previously less preferred). By traveling through this branch, the possible solutions are very different from each other, thus the probability of finding the best scheduling policies becomes higher.

The candidates from both branches ($X_{RL}$ and $\tilde{X}_{RL}$) are evaluated using a reward function $R(\cdot)$ as Equation. (24) that captures multiple objectives.

$$R(X) = \varphi_1 \frac{1}{Latency(X)} + \varphi_2 \frac{1}{Energy(X)} + \quad \varphi_3 \frac{1}{Cost(X)} + \varphi_4 \cdot SLACompliance(X) \quad (24)$$

Where, $\varphi_1$, $\varphi_2$, $\varphi_3$, $\varphi_4$ denotes the weight factors representing the relative importance of latency, energy, operational cost, and SLA compliance. The candidate with the higher reward score is retained for the next stage.

This parallel policy generation allows the scheduler to simultaneously utilize the trained RL strategies and investigate new candidate mappings, thereby keeping a balance between the best task allocation and the wide-ranging solution search.

### 3.5.3. Coati-Inspired Search Dynamics

Following the generation and evaluation of candidate scheduling policies ($X_{RL}$ and $\tilde{X}_{RL}$) in Section 3.5.2, the RL-COSCoati framework initiates a Coati-based metaheuristic search process to fine-tune task-VM allocations. The method is based on the foraging and social behavior of coatis (which are raccoon family members), which combine individual scouting exploration with group cooperation when locating food sources.

#### 3.5.3.1. In the context of VM scheduling:

- Exploration (scouting): Mimics coatis dispersing widely in search of new resources, corresponding to diverse task allocation trials across the multi-cloud environment.
- Exploitation (cooperation): Reflects coatis regrouping once a promising food source is found, corresponding to converging on high-quality scheduling policies.

#### 3.5.3.2. Initialization of Scheduling Population

At the start of this stage, an initial scheduling population $\mathcal{P}^{(0)}$ is created by combining the retained candidate vector (Equation 24) with additional randomized mappings as Equation. (25):

$$\mathcal{P}^{(0)} = \{X_{RL}, \tilde{X}_{RL}, X_1, X_2, \dots, X_p\} \quad (25)$$

Where, $p$ is the population size, and each $X_k$ represents a distinct task-VM scheduling configuration. This parallels the way coatis form groups with multiple individuals exploring different areas simultaneously.

#### 3.5.3.3. Fitness Function for Scheduling

Each candidate schedule $X_k$ is evaluated using a multi-objective fitness function that balances latency, energy,

cost, and SLA compliance, which is expressed in Equation. (26):

$$F(X_{\hbar}) = \beta_1 \frac{1}{Latency(X_{\hbar})} + \beta_2 \frac{1}{Energy(X_{\hbar})} + \beta_3 \frac{1}{Cost(X_{\hbar})} + \beta_4 \cdot SLA.Compliance(X_{\hbar}) \quad (26)$$

These mimics coatis evaluating food quality and accessibility, with the scheduler prioritizing solutions that maximize efficiency and reliability.

#### 3.5.3.4. Exploration (Scouting Phase)

During the exploration phase, candidate solutions are spread out over the scheduling search space in a manner similar to coatis spreading in forests to find new food patches. In this way, the algorithm looks at a wide range of tasks-VM mappings and does not concentrate on a single region too early. The way the candidates are updated is random, hence each candidate can change its position in relation to another peer chosen at random, which is given by Equation. (27):

$$X_{\hbar}^{(t+1)} = X_{\hbar}^{(t)} + \mathfrak{r} \cdot \left(X_{rand} - X_{\hbar}^{(t)}\right), \quad \mathfrak{r} \in [0,1] \quad (27)$$

Here, $X_k^{(t)}$ denotes the scheduling configuration of candidate $k$ at iteration $t$, $X_{rand}$ is a randomly sampled candidate from the population, and $r$ is a random exploration factor that controls the step size of scouting movement.

To strengthen this mechanism, adopts refinements inspired by the Improved Coati Optimization Algorithm (ICOA), which incorporates sound transmission and social scattering behaviors of coatis:

#### 3.5.3.5. Sound Transmission (Indirect Awareness):

Coatis use vocalizations to signal the presence of food. In the scheduling context, this can be modeled by directing some candidates toward promising peers which is mathematically expressed as Equation. (28).

$$X_{\hbar}^{(t+1)} = X_{\hbar}^{(t)} + \alpha \cdot \left(X_{best}^{(t)} - X_{\hbar}^{(t)}\right) + \beta \cdot \left(X_{rand} - X_{\hbar}^{(t)}\right) \quad (28)$$

Where, $X_{best}^{(t)}$ is the global best solution at iteration $t$, $\alpha \in (0,1)$ controls the attraction toward high-quality solutions (awareness), and $\beta \in (0,1)$ preserves stochastic scouting.

#### 3.5.3.6. Social Scattering (Broad Coverage):

To prevent overcrowding around certain solutions, ICOA scatters individuals by amplifying the difference between randomly selected candidates, which is mathematically expressed as Equation. (29):

$$X_{\hbar}^{(t+1)} = X_{\hbar}^{(t)} + \gamma \cdot \left(X_j^{(t)} - X_l^{(t)}\right), j \neq l \quad (29)$$

Where, $\gamma \sim U(0,1)$ is a random coefficient, and $X_j^{(t)}$, $X_l^{(t)}$ are randomly chosen candidates. This term separates the individuals, thus making sure that the task-VM scheduling area is explored more broadly. Such a hybrid exploration provides a good balance of search diversity and getting closer to high-quality solutions, which makes the population ready for the next phase of Exploitation Cooperation Phase where the best scheduling policies are refined.

#### 3.5.3.7. Exploitation (Cooperation Phase)

During the exploitation phase, candidate solutions merge the best scheduling policy found up to that point, much like coatis converging on a food-rich area after a successful scout has located resources. The main idea of this phase is to highlight the concept of intensification, which essentially means that the search will be refined to focus on promising allocations rather than going further with broad exploration. The basic cooperation update is expressed as Equation. (30).

$$X_{\hbar}^{(t+1)} = X_{\hbar}^{(t)} + \lambda_{coo} \cdot \left(X_{best} - X_{\hbar}^{(t)}\right), \lambda_{coo} \in [0,1] \quad (30)$$

Where, $X_k^{(t)}$ represents the scheduling configuration of candidate $k$ at iteration $t$, $X_{best}$ is the current global best solution, and $\lambda_{coo}$ is the cooperation factor that regulates the speed of convergence. A higher $\lambda_{coo}$ accelerates convergence but risks premature stagnation, while a smaller value encourages gradual refinement. To strengthen exploitation while avoiding local optima, ICOA-inspired refinements are incorporated:

#### 3.5.3.8. Weighted Cooperation (Adaptive Attraction):

Instead of uniformly moving toward the global best, each candidate considers both the best-known solution and elite peers, which is given by Equation. (31).

$$X_{\hbar}^{(t+1)} = X_{\hbar}^{(t)} + \lambda_{coo1} \cdot \left(X_{best} - X_{\hbar}^{(t)}\right) + \lambda_{coo2} \cdot \left(X_{elite} - X_{\hbar}^{(t)}\right) \quad (31)$$

Where, $X_{elite}$ presents a high-performing candidate (but not necessarily the global best), and $\lambda_{coo1}$, $\lambda_{coo2} \in (0,1)$ are adaptive cooperation weights. This dual attraction prevents over-exploitation of a single solution.

#### 3.5.3.9. Adaptive Convergence Control:

To dynamically balance exploration and exploitation, $\lambda_{coo}$ can be made iteration-dependent as Equation. (32).

$$\lambda_{coo}(t) = \lambda_{coo(min)} + \left(\lambda_{coo(max)} - \lambda_{coo(min)}\right) \cdot \frac{t}{t_{max}} \quad (32)$$

Where, $t$ signifies the current iteration, $t_{max}$ is the maximum number of iterations, and $\lambda_{coo(min)}$, $\lambda_{coo(max)}$ define the range of cooperation strength. Early iterations favor slower convergence (larger search space coverage), while later iterations accelerate convergence toward the global optimum.

Through the addition of weighted attraction and adaptive convergence control to the cooperation phase, RL-COSCoati makes sure that exploitation is not only deepened but also variable-candidates get closer to the global best solution while still having enough diversity to be resistant.

By alternating between exploration and exploitation,

RL-COSCoati maintains a balance between:

- Diversity (exploration of novel task-VM mappings to avoid local optima).
- Intensification (exploitation around the global best allocation to achieve stability).

The outcome of this phase is the refined scheduling vector $X^*$, which is defined as Equation. (33).

$$X^* = \arg \max_{X_k \in \mathcal{P}^{(t)}} F(X_k) \quad (33)$$

which represents the globally optimal task-to-VM allocation policy, derived from coati-inspired search behavior.

#### 3.5.3.10. Termination Condition

The Coati-inspired search process continues iteratively, alternating between exploration (scouting) and exploitation (cooperation), until one of the termination criteria is met, which is Stop if ($|F(X^{*(t)})-F(X^{*(t-1)})| < \epsilon$) ∨ ($t \geq t_{max}$). Here, $F(X^{*(t)})$ signifies the best fitness value at iteration $t$, and $\epsilon$ denotes a small threshold for improvement tolerance. If neither condition is satisfied, the algorithm continues the search in the next iteration.

### 3.5.4. Scheduling Execution

After obtaining the refined scheduling vector $X^*$ through the Coati-style search dynamics (Section 3.5.3), the RL-COSCoati system commits the execution of the local task-VM allocation policy in the multi-cloud environment. In other words, this step is about the implementation of the optimized decisions to carry out the practical scheduling actions.

#### 3.5.4.1. Task Assignment Rules

The execution process adheres to the VM categories from Section 3.4 (Overloaded, Underloaded, Balanced):

- Underloaded VMs ($\hat{w}_{t+1} < \tau_{low}$): Assigned priority for new tasks, increasing utilization and balancing the workload distribution.
- Balanced VMs ($\tau_{low} \leq \hat{w}_{t+1} \leq \tau_{high}$): Retain current workloads to maintain stability. New tasks may be assigned only if resource limits allow.
- Overloaded VMs ($\hat{w}_{t+1} > \tau_{high}$): Excluded from new allocations to prevent SLA violations and excessive latency.

#### 3.5.4.2. Scheduling Decision Function

The mapping of a task $T_j$ to a VM is guided by a multi-objective optimization function which is defined as Equation. (34).

$$\pi(T_j) = \arg \min_{VM_i \in \{Underloaded \cup Balanced\}} (\varphi_1 \cdot Latency(VM_i) + \varphi_2 \cdot Energy(VM_i) + \varphi_3\ Cost(VM_i)) \quad (34)$$

Where, $\pi(T_j)$ selects the optimal VM for $T_j$, $\varphi_1$, $\varphi_2$, $\varphi_3$ are weights representing system priorities (latency-sensitive vs. cost-aware scheduling). This ensures that tasks are mapped to VMs offering the best trade-off between performance, cost, and energy.

#### 3.5.4.3. Execution Monitoring and Embedded Feedback

Once scheduling is enforced, the system continuously monitors:

- Throughput (tasks processed per unit time),
- Average latency,
- Energy consumption,
- SLA compliance.

Besides evaluating runtime performance, these metrics also serve as a feedback signal to the RL-COSCoati agent, which enables the adaptive refinement of scheduling policies in the next iterations. Figure 10 shows the RL-COSCoati optimized VM.

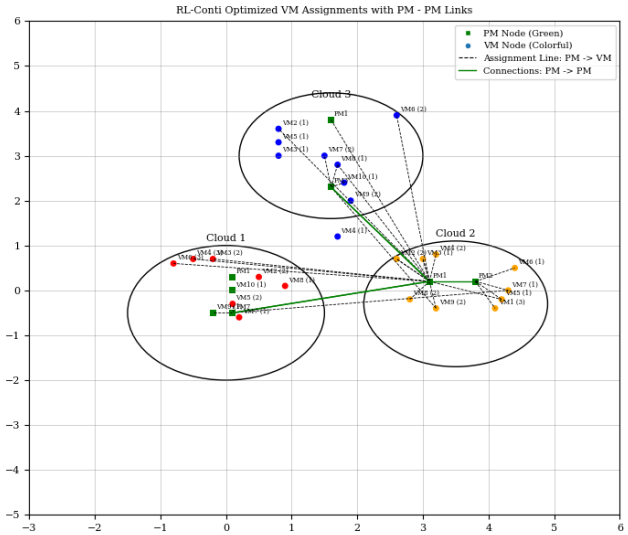


Figure 10. RL-COSCoati optimized VM.

## 3.6. Task Execution

The execution phase is the one that actually carries out the task-VM mappings that are decided by the RL-COSCoati framework. Here, the system goes down from policy-level decisions to the ground-level practicals, where tasks are handed over to their best-suited virtual machines (VMs) and are run on the spot.

### 3.6.1. Execution of Tasks on Allocated VMs

Let the refined scheduling decision for task $T_j$ be represented as $\pi(T_j)=VM_i^*$, where $VM_i^*$ is the VM selected based on Equation (32) in Section 3.5.4. Once this decision is finalized, task execution proceeds as follows:

- Task Dispatching→ Each task $T_j$ is transmitted to its designated VM, ensuring that input data (audio, video, EEG, image, ECG, or numerical signals) are properly mapped to the VM's allocated compute, memory, and bandwidth resources.
- Task Processing→ The VM executes the task using

its available resources (*CPU*, *RAM*, *Bandwidth*, *Storage*), where processing time depends on both task complexity and VM capacity. The expected execution time of task $T_j$ on VM $VM_i^*$ can be modeled as Equation. (35):

$$ET(\mathcal{T}_j, VM_i^*) = \frac{\mathbb{W}(\mathcal{T}_j)}{MIPS(VM_i^*)} \quad (35)$$

Where, $W(T_j)$ is the workload size of task $T_j$ (in million instructions), $MIPS(VM_i^*)$ is the processing speed of the selected VM (Million instructions per second). This ensures that the chosen VM provides an efficient balance between throughput, latency, and energy.

### 3.6.2. Real-Time Performance Monitoring

During execution, the system continuously monitors runtime performance across multiple dimensions to validate that the scheduling decisions achieve the desired objectives:

- Latency Monitoring: The response time for each executed task is computed as Equation. (36):

$$RT(\mathcal{T}_j) = ET(\mathcal{T}_j, VM_i^*) + QD(VM_i^*) \quad (36)$$

Where, $QD(VM_i^*)$ represents the queuing delay at VM *i*.

- Energy Consumption: The instantaneous power draw of VM $VM_i^*$ during execution is measured and aggregated into task-level energy consumption as Equation. (37):

$$\mathcal{E}(\mathcal{T}_j, VM_i^*) = P(VM_i^*) \times ET(\mathcal{T}_j, VM_i^*) \quad (37)$$

Where, $P(VM_i^*)$ is the average power usage of the VM.

- Resource Utilization: Utilization ratios for CPU, memory, and bandwidth are tracked to ensure resources are neither underutilized nor pushed beyond safe thresholds.
- SLA Compliance: Execution results are checked against SLA-defined thresholds for latency, availability, and error rate. A violation triggers adaptive re-scheduling signals to the RL-COSCoati agent.

### 3.6.3. Embedded Feedback Loop

The execution phase is closely integrated with the reinforcement learning cycle and is, therefore, not an isolated process. The metrics that are observed include latency, energy consumption, resource utilization, and SLA compliance-are returned to the RL-COSCoati reward function, which is given mathematically in Equation. (38):

$$\mathcal{R} = \gamma_1 \cdot \frac{1}{RT} + \gamma_2 \cdot \frac{1}{\mathcal{E}} + \gamma_3 \cdot SLA\_Success \quad (38)$$

Where, $\gamma_1$, $\gamma_2$, $\gamma_3$ are reward weights. This ensures that real execution outcomes inform the policy for subsequent scheduling decisions, enabling adaptive learning under dynamic workloads. The task execution is graphically represented in Figure 11.

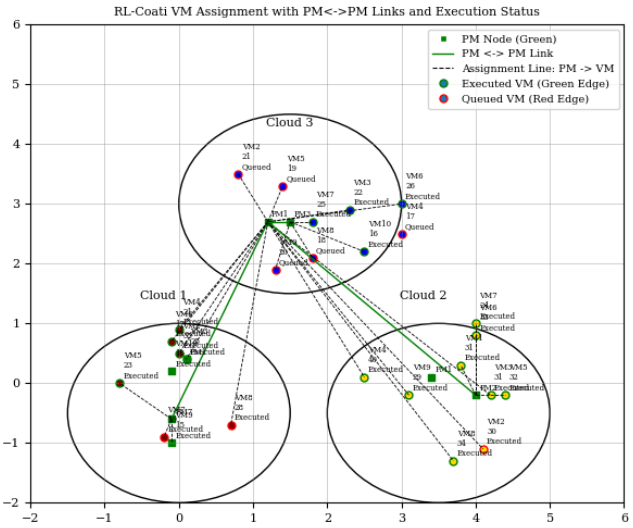


Figure 11. Task execution.

During this stage, tasks are carried out on their respective VMs, however, execution monitoring might still show that there are situations of PMs being persistently underutilized or overloaded. Such PMs imbalances push the necessity of VM consolidation and migration that is elaborated in Section 3.7.

## 3.7. VM Consolidation (VM Transfer)

The execution phase of Section 3.6 is about efficiently processing tasks on their allocated VMs. But live monitoring frequently shows that the usage of resources over PMs is still not balanced-for example a few PMs might be running VMs that are underutilized, whereas others may be almost fully loaded. If these imbalances are not compensated for, they will result in the waste of electric energy and a decline in the total performance of the system.

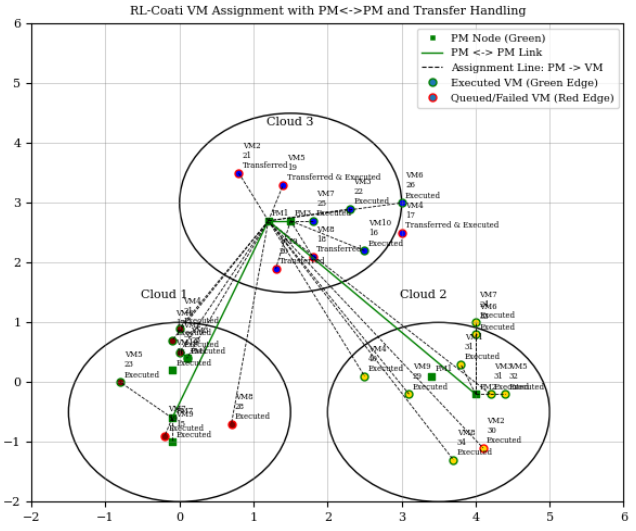


Figure 12. VM transfer.

The framework uses VM consolidation as a solution to this issue, which means VMs are moved from one PM

to another in a dynamic manner. In addition to achieving a more balanced workload distribution across the multi-cloud infrastructure, this operation also results in lower energy consumption as the PMs that are left without a load can be turned off. The VM transfer is depicted in Figure 12.

The primary objective of VM consolidation is to minimize the number of active PMs while ensuring SLA-compliant task execution. This can be mathematically expressed as min $\sum^{n}_{p=1} Active(PM_p)$, which is subjected to $\sum_{(VM\_i \in PM\_p)} U_{cpu}(VM_i) \leq Cap_{cpu}(PM_p)$, $\sum_{(VM\_i \in PM\_p)} U_{mem}(VM_i) \leq Cap_{mem}(PM_p)$. Here, $Active(PM_p) = 1$ if PM $p$ hosts at least one VM, otherwise 0, $U_{cpu}(VM_i)$ and $U_{mem}(VM_i)$ are CPU and memory utilization of VM $i$, and $Cap_{cpu}(PM_p)$ and $Cap_{mem}(PM_p)$ are the total CPU and memory capacities of PM $p$.

### 3.7.1. VM Transfer Mechanism

The consolidation process proceeds in three stages:

- Identification of Candidate VMs for Migration: From real-time monitoring, underloaded VMs ($\hat{w}_{t+1} < \tau_{low}$) are marked as potential candidates for migration. If a PM hosts only underloaded VMs, it is flagged for shutdown consolidation.
- VM Selection for Migration: The candidate VMs are chosen using a heuristic such as Minimum Migration Time (MMT), defined as Equation. (39):

$$MMT(VM_i) = \frac{Mem(VM_i)}{BW(PM_p)} \tag{39}$$

Where, $Mem(VM_i)$ denotes the memory footprint of VM $i$, and $BW(PM_p)$ represents the available bandwidth of PM $p$. VMs with lower migration costs are prioritized to minimize service disruption.

- VM Reallocation to Target PMs: Selected VMs are moved to balanced or lightly loaded PMs with which the capacity limitations are not exceeded. When migration leads to a complete consolidation of a PM, the source PM is switched off to save energy.

### 3.7.2. Seamless Migration and Service Continuity

To avoid service interruption during migration:

- Live Migration Techniques are employed, allowing VMs to continue execution while their memory state is progressively copied to the destination PM.
- Downtime is minimized by transferring the majority of VM memory in iterative pre-copy rounds, with only the remaining delta copied during a very short pause.

The downtime $D_{VM}$ can be modeled as Equation. (40):

$$\mathcal{D}_{VM} = \frac{R_{mem}(VM_i)}{BW(PM_p)} \tag{40}$$

Where, $R_{mem}(VM_i)$ represents the residual memory at the final copy stage. Although virtual machine consolidation maximizes resource use within a single cloud; multi-cloud setups need supplementary methods to make sure that work can be handed over to different clouds as well for overall balancing. This move thus brings us to Section 3.8 Cloud Communication that deals with the communication between clouds for scheduling of tasks and exchanging data in a secure manner.

## 3.8. Cloud Communication (Multi-Cloud Load Balancing and Task Offloading)

Though VM consolidation (Section 3.7) is aimed at efficiency within a single cloud, the reality is that most applications these days require cross-cloud operations to be able to respond to spikes in the workload, comply with SLA requirements, or take advantage of resources that are specialized. Load balancing is taken by the system to the multi-cloud tier in this stage of the work where jobs can be moved from a CSP to a different one via secure communication channels Ishtaiwi *et al*. [26].

The goal of cloud-to-cloud communication is to minimize global latency and energy consumption while preventing overload in any single CSP. This can be expressed as min $(\sum_{(c=1 \text{ to } C)} [\rho_1 \cdot L_c + \rho_2 \cdot E_c + \rho_3 \cdot C_c])$ which is subjected to $U_c^{total} \leq Cap_c, \forall_c \in C$. Where, $L_c$, $E_c$ and $C_c$ represent latency, energy consumption, and cost in cloud $c$, $\rho_1$, $\rho_2$, $\rho_3$ are priority weights, $U_c^{total}$ denotes the the aggregated resource utilization in cloud $c$, and $Cap_c$ signifies the maximum capacity of cloud $c$. This ensures that offloading decisions are made with a global optimization perspective across all participating CSPs. The cloud-to-cloud communication is graphically shown in Figure 13.

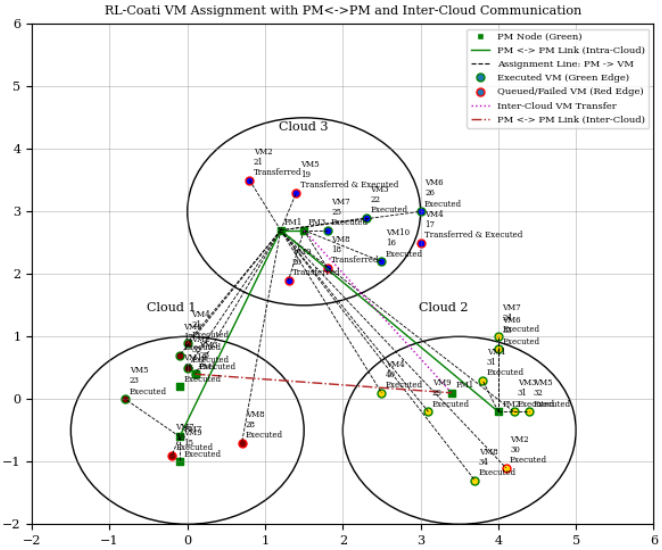


Figure 13. Cloud Communication.

### 3.8.1. Secure Data and Control Exchange

To enable reliable inter-cloud communication, the framework integrates both data-plane and control-plane

interactions:

- Data-Plane Communication: Transfers actual task payloads (video, EEG, image, numerical data) between CSPs. Offloaded data is encrypted using Elliptic Curve Cryptography (ECC) to ensure confidentiality during transit.
- Control-Plane Communication: Exchanges scheduling metadata, VM status, and workload predictions across CSPs. Control messages are lightweight but secured with Transport Layer Security (TLS 1.3) for authentication and integrity.

### 3.8.2. Task Offloading Mechanism

The decision to offload tasks between clouds proceeds in three steps:

- Detection of Overloaded CSPs: From global monitoring, if a CSP's average utilization exceeds a high threshold (($\tau\_{cloud}$) ^high), it is flagged as overloaded.
- Selection of Target CSPs: Candidate CSPs are identified based on availability and lower utilization ($U_c$ < (τ_cloud) ^high). Target selection is guided by a multi-objective offloading score, which is defined as Equation. (41):

$$Score(c) = \varphi_1 \cdot \frac{1}{Latency_c} + \varphi_2 \cdot \frac{1}{Cost_c} \cdot \varphi_3 \cdot Trust_c \qquad (41)$$

Where, $Trust_c$ measures historical SLA compliance of cloud $c$.

- Offloading Execution: Tasks from overloaded CSPs are migrated to selected underloaded CSPs. Load is distributed such that global SLA violations and resource wastage are minimized.

While cloud communication allows for global task load balancing and offloading, cross-cloud interactions additionally escalate the possibility of errors such as a failure of a node, loss of packets, or breach of SLA. Hence, the system features a fault recognition module that is described in detail in Section 3.9.

## 3.9. Fault Identification via CloudGuard

The expansion of load balancing to a multi-cloud setting (Section 3.8) enables better scalability and more efficient task distribution, however, it also escalates the possibility of local faults. Considering the existence of several CSPs, heterogeneous workloads, and complicated VM migrations, the system still has to be robust against VM crashes, resource exhaustion, network failures, and performance degradation. To tackle the problem, introduces CloudGuard, an innovative fault identification module using fuzzy logic that is designed to work closely with the load-balancing process.

CloudGuard remains active in checking system health, comprehending diverse signals, and making decisions aware of faults to the federated load balancer. Its operations are spread over seven phases, beginning with VM monitoring and concluding with the return of information to the load balancer. The general flow of proposed CloudGuard is depicted in Figure 14.

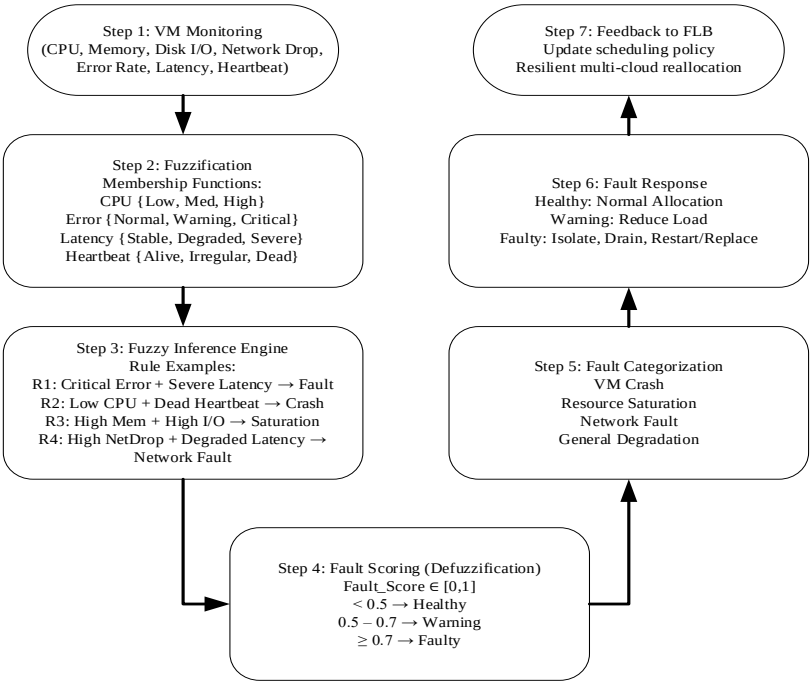


Figure 14. Flow of cloudguard.

### 3.9.1. Step 1: Input Metrics (VM Monitoring)

CloudGuard's core is built around in-depth runtime monitoring of VMs as well as the cloud components linked to them. After the assignment and execution of tasks (Section 3.6) and their consolidation in PMs (Section 3.7), monitoring probes of a light nature are deployed in every VM to record performance and

reliability metrics.

The monitored parameters are:

- CPU utilization ($U_{cpu}$): Indicates the proportion of the computational resources utilized by a virtual machine. A high long-run utilization can be a sign of resource contention or overload, while an extremely low utilization in a situation of an active workload may be a silent execution of the task.
- Memory usage ($U_{mem}$): Tracks consumed vs. allocated memory. Continuous high memory usage can indicate memory leaks or saturation conditions.
- Disk I/O wait ($IO_{wait}$): Captures delays in disk operations. Prolonged I/O wait can degrade task performance, especially for data-intensive applications (video or EEG inputs).
- Network packet drop ($Drop_{net}$): Reflects the ratio of lost packets to transmitted packets. High packet drop directly affects cloud communication efficiency (linking back to Section 3.8).
- Request error rate ($E_{req}$): Monitors failed requests per unit time. Rising error rates are early indicators of application or service-level failures.
- Response latency ($L_{resp}$): Measures end-to-end task completion time as perceived by the user. Latency spikes often precede SLA violations.
- Heartbeat interval ($HB_{int}$): Represents periodic VM health signals. Missed or irregular heartbeats typically indicate VM crash or network isolation.

#### 3.9.1.1. Mathematical Representation of Monitoring State

At each monitoring interval $I$, the health state of VM $VM_i$ can be represented as Equation. (42):

$$\mathcal{H}(VM_i,\mathcal{I}) = \{\mathbb{U}_{cpu}(\mathcal{I}), \mathbb{U}_{mem}(\mathcal{I}), \mathbb{IO}_{wait}(\mathcal{I}), Drop_{net}(\mathcal{I}), \mathbb{E}_{req}(\mathcal{I}), \mathbb{L}_{resp}(\mathcal{I}), \mathbb{HB}_{int}(\mathcal{I})\} \quad (42)$$

This vectorized health profile $H$ ($VM_i$, $I$) serves as the input state for the subsequent fuzzification stage (Step 2).

CloudGuard is making sure that no anomaly of a single nature is neglected by combining these diverse performance indicators. But in fact, the raw monitoring data is frequently noisy and it is not straightforward for a person to comprehend it. In order to cope with uncertainty and inaccuracy, CloudGuard performs fuzzification, which associates numerical values with linguistic categories like Low, Medium, High or Normal, Warning, Critical. This change is explained in Step 2: Fuzzification (Membership Functions).

### 3.9.2. Step 2: Fuzzification (Membership Functions)

The raw monitoring state vector $H$ ($VM_i$, $I$) (Equation. 43 from Step 1) provides detailed performance readings of each VM. Nevertheless, such values stand as continuous, noisy, and uncertain; thus, it is quite hard to make a direct inference if a VM is healthy or faulty. To deal with this problem, CloudGuard uses fuzzification, which is a method that converts crisp numerical inputs into linguistic variables by means of Membership Functions (MFs).

This fuzzy representation captures the degree of abnormality rather than binary states, enabling more flexible and intelligent fault detection.

#### 3.9.2.1. Defining Linguistic Variables and Membership Functions

Each monitored metric from Step 1 is mapped into linguistic categories using fuzzy sets. which include:

- For CPU Utilization ($U_{cpu}$): Linguistic values are low, medium, high and membership function (triangular) is defined as Equation. (43):

$$\mu_{Low}(\mathbb{U}_{cpu}) = \max\left(0, \frac{\tau_{mid} - \mathbb{U}_{cpu}}{\tau_{mid} - \tau_{low}}\right), \quad \mu_{High}(\mathbb{U}_{cpu}) = \max\left(0, \frac{\mathbb{U}_{cpu} - \tau_{mid}}{\tau_{high} - \tau_{mid}}\right) \quad (43)$$

Where, $\tau_{low}$, $\tau_{mid}$, $\tau_{high}$ are threshold points for CPU utilization.

- For Request Error Rate ($E_{req}$): Linguistic values are normal, warning, critical and membership function (trapezoidal) is given by Equation. (44):

$$\mu_{critical}(\mathbb{E}_{req}) = \begin{cases} 0, & \mathbb{E}_{req} < \alpha_1 \\ \frac{\mathbb{E}_{req} - \alpha_1}{\alpha_2 - \alpha_1}, & \alpha_1 \le \mathbb{E}_{req} \le \alpha_2 \\ 1, & \mathbb{E}_{req} > \alpha_2 \end{cases} \quad (44)$$

- For Response Latency ($L_{resp}$): Linguistic values are stable, degraded, severe and membership function (Gaussian) is defined as Equation. (45):

$$\mu_{Severe}(\mathbb{L}_{\text{resp}}) = \exp\left(-\frac{(\mathbb{L}_{\text{resp}} - \mu)^2}{2\sigma^2}\right) \quad (45)$$

Where, $\mu$ and $\sigma$ define the severity center and spread.

- For Heartbeat Interval ($HB_{int}$): Linguistic values are alive, irregular, dead and membership function (piecewise linear) is given by Equation. (46):

$$\mu_{Dead}(\mathbb{HB}_{int}) = \begin{cases} 0, & \mathbb{HB}_{int} \le \delta_{low} \\ \frac{\mathbb{HB}_{int} - \delta_{low}}{\delta_{high} - \delta_{low}}, & \delta_{low} < \mathbb{HB}_{int} < \delta_{high} \\ 1, & \mathbb{HB}_{int} \ge \delta_{high} \end{cases} \quad (46)$$

#### 3.9.2.2. Example Fuzzy Mapping

Consider a VM with the following metrics:

- $U_{cpu}$=85%,
- $E_{req}$=0.12,
- $L_{resp}$=400ms,
- $HB_{int}$=8s.

The fuzzification process might yield:

- $\mu_{High}$ ($U_{cpu}$) =0.9→CPU is strongly *High*.
- $\mu_{critical}$ ($E_{req}$) =0.6→Error rate is partly *Critical*.
- $\mu_{Severe}$ ($L_{resp}$) =0.75→Latency is significantly *Severe*.
- $\mu_{Dead}$ ($HB_{int}$) =0.2→Heartbeat is slightly drifting but not *Dead*.

Therefore, the VM state is not represented by sharp thresholds but rather as different levels of abnormality, which makes the fault detection more nuanced. The fuzzy state thus obtained serves as the input for the fuzzy inference engine Step 3 that uses the predefined if-then rules to combine several fuzzy conditions in order to decide whether a VM is faulty, healthy, or degraded.

### 3.9.3. Step 3: Fuzzy Inference Engine (Rule Evaluation)

From Step 2 (Fuzzification), each VM's monitoring metrics have been transformed into fuzzy linguistic variables (Low, High, Critical, Severe). The next stage of CloudGuard is the Fuzzy Inference Engine (FIE), which evaluates these fuzzy inputs against a set of if-then rules to infer potential faults.

The FIE uses a Mamdani-style inference mechanism, where the antecedent conditions are matched with the fuzzified metrics, and the output is a fault likelihood expressed as a fuzzy set.

#### 3.9.3.1. Rule Base

The fuzzy rules are designed to capture common fault patterns in multi-cloud VM operations. Some representative rules are:

- **R1**: IF error rate is critical and latency is severe→ Fault =True
- **R2**: IF CPU is low and heartbeat is dead→Fault =VM Crash
- **R3**: IF memory is high and disk I/O wait is high→ Fault =Resource Saturation
- **R4**: IF network packet drop is high and latency is degraded→ Fault =Network Fault

#### 3.9.3.2. Mathematical Formulation of Rule Evaluation

For a general fuzzy rule of the form is $R_k$: IF $x_1$ is $A_1^k$ and $x_2$ is $A_2^k$ ⇒ y is $B^k$. The firing strength of rule $R_k$ is computed as Equation. (47):

$$\omega_k = \min\left(\mu_{A_1^k}(x_1), \mu_{A_2^k}(x_2), \dots\right) \tag{47}$$

Where, $\mu_{A_j^k}(x_j)$ is the membership degree of input $x_j$ in fuzzy set $A_j^k$. The fault output set for rule $R_k$ is then scaled by its firing strength as Equation. (48):

$$\mu'_{B^k}(y) = \omega_k \cdot \mu_{B^k}(y) \tag{48}$$

Finally, all rule outputs are aggregated using the max operator, which is given by Equation. (49):

$$\mu_{fault}(y) = \max_k\left(\mu'_{B^k}(y)\right) \tag{49}$$

#### 3.9.3.3. Example Rule Evaluation

Suppose a VM exhibits as $\mu_{critical}$ ($E_{req}$)=0.6 and $\mu_{Severe}$ ($L_{resp}$)=0.75
For R1:
$\omega_{R1}$=min (0.6, 0.75) =0.6
Thus, the fault set from R1 contributes with a 0.6 strength to the global fault likelihood. The aggregated fuzzy fault likelihood $\mu_{fault}(y)$ is still a fuzzy value. To convert this into a crisp, actionable Fault Score, CloudGuard proceeds to Step 4: Fault Scoring (Defuzzification).

### 3.9.4. Step 4: Fault Scoring (Defuzzification)

From Step 3, the output of the FIE is an aggregated fuzzy fault likelihood, denoted as $\mu_{fault}(y)$. Although this depiction succeeds in reflecting the ambiguity of the VM's condition, the load balancer cannot use it directly. In order to make it possible for the decisions to be taken, a defuzzification step is performed that changes the fuzzy result into a definite numerical Fault Score that lies within the range [0,1].

#### 3.9.4.1. Centroid Method of Defuzzification

Among available defuzzification techniques (max-membership, weighted average, centroid), this work employs the centroid center of gravity method due to its stability and ability to reflect the weighted influence of multiple fuzzy rules. Formally, the Fault Score for VM *i* at time *t* is computed as Equation. (50):

$$Fault\_Score_{\mathrm{i}}(\mathbb{t}) = \frac{\int \mu \cdot \mu_{fault}(y)\,d\mu}{\int \mu_{fault}(y)\,d\mu} \tag{50}$$

Where, $\mu$ denotes the output domain of fault likelihood, $\mu_{fault}(y)$ signifies the membership function aggregated in Step 3, the numerator captures the weighted contribution of each fault likelihood, and the denominator ensures normalization into the range [0,1].

#### 3.9.4.2. Decision Thresholds for VM Health

The crisp fault score is then mapped to discrete health states using predefined thresholds Equation. (51):

$$State(VM_i) = \begin{cases} Healthy, & Fault\_Score_{\mathrm{i}} < 0.5 \\ Warning(Degraded), & 0.5 \le Fault\_Score_{\mathrm{i}} < 0.7 \\ Faulty, & Fault\_Score_{\mathrm{i}} \ge 0.7 \end{cases} \tag{51}$$

This categorization allows the system to differentiate between minor degradations (which may be recoverable without migration) and severe faults (which demand immediate VM isolation).

#### 3.9.4.3. Example Defuzzification

Suppose the aggregated fault membership function from Step 3 yields:

- Fault likelihood centered at 0.8 with high weight (severe latency and critical error rate),
- Fault likelihood around 0.3 with low weight (mild CPU imbalance).

The centroid-based fault score is computed as *Fault_Score* =0.72. According to the decision

thresholds, this VM is classified as Faulty, triggering corrective measures in Step 6.
Therefore, the defuzzification step gives a quantitative health metric that mediates fuzzy inference and deterministic scheduling policies, so that VM fault states can always be interpreted by the federated load balancer.

### 3.9.5. Step 5: Fault Categorization

Once the crisp Fault Score has been calculated in Step 4, CloudGuard goes ahead to categorize the identified anomaly into particular faults. This classification is important in that the presence of a fault is not only detected, but also the cause of the fault, which allows the system to initiate the most suitable recovery process.

#### 3.9.5.1. Fault Categories

Based on monitored metrics (CPU utilization, memory, I/O wait, packet loss, error rates, latency, heartbeat signals), faults are categorized into four primary classes: VM Crash-complete unresponsiveness due to halted execution or lost heartbeat signals.

- Indicator metrics: $U_{cpu} \approx 0$, missed heartbeats ($HB_{int}=0$), rising error rates.

Resource Saturation-VM remains active but exhibits excessive consumption of resources.

- Indicator metrics: sustained $CPU > 90\%$, $U_{mem} > 90\%$, prolonged I/O wait.

Network Fault-degradation in data exchange due to communication issues.

- Indicator metrics: high packet drop ($Drop_{net}$), increased response latency, degraded throughput.

General Degradation-intermediate faults not fitting the above categories, often early signs of instability.

- Indicator metrics: fluctuating latency, medium error rate, inconsistent resource usage.

#### 3.9.5.2. Categorization Function

The mapping of Fault Score and associated monitoring metrics into categories can be expressed as Equation. (52):

$$Category(VM_i) = \begin{cases} VM\ Crash, & (\mathbb{HB}_{int} = 0 \wedge \mathbb{U}_{cpu} \approx 0) \\ Resource\ Saturation, & (\mathbb{U}_{cpu} > \theta_{cpu} \vee \mathbb{U}_{mem} > \theta_{mem}) \\ Network\ Fault, & (Drop_{net} > \theta_{net} \wedge \mathbb{L}_{resp} > \theta_{Lat}) \\ General\ Degradation, & otherwise\ if\ 0.5 \leq Fault\_Score_i < 0.7 \end{cases} \quad (52)$$

Where, $\theta_{cpu}$, $\theta_{mem}$, $\theta_{net}$, $\theta_{Lat}$ are empirically determined thresholds for CPU, memory, packet drop, and latency, respectively. The categorization function leverages both the Fault Score from Step 4 and specific metric thresholds to ensure accurate fault attribution.

#### 3.9.5.3. Example Categorization

- If a VM reports $U_{cpu}$ = 95%, $U_{mem}$ = 92%, and Fault Score= 0.68, the system classifies it as Resource Saturation.
- If $HB_{int}$= 0 and $U_{cpu}$= 0, the VM is categorized as VM Crash.
- If $Drop_{net}$= 20% and Latency= 500 ms, the VM is flagged as Network Fault.

Through this categorization process, CloudGuard distinguishes the nature of failure, which becomes the foundation for the targeted fault recovery mechanisms introduced in Step 6.

### 3.9.6. Step 6: Fault Response Mechanism

After establishing the fault state and category of a VM Step 5, CloudGuard triggers an adaptive Fault Response Mechanism. The main goal is to provide continuous service delivery by dynamically managing faulty or degraded VMs with minimal SLA breach, resource wastage, and latency as perceived by users.
The response mechanism operates as a policy-driven state machine, where each VM health state triggers predefined corrective actions:

#### 3.9.6.1. Healthy State (*Fault_Score* < 0.5)

- Action: VM remains fully active in the federated load balancing pool.
- Tasks continue to be allocated normally via the RL-COSCoati scheduler.
- No intervention is required beyond standard monitoring.

#### 3.9.6.2. Warning / Degraded State (0.5 ≤ Fault_Score < 0.7)

- Action: VM is still functional but at risk of fault escalation.
- Corrective responses:
- Gradual reduction of task inflow to the VM.
- Prioritization of lightweight tasks over resource-intensive ones.
- Continuous re-monitoring at finer intervals ($\Delta t_{warn} < \Delta t_{normal}$).
- Objective: stabilize the VM, prevent escalation to a hard fault, and preserve SLA compliance.

#### 3.9.6.3. Faulty State (Fault_Score ≥ 0.7)

- Action: VM is considered non-operational and immediately isolated.
- The corrective actions include isolation (take the VM out of the load balancing pool to avoid additional task assignments), drain Phase (drain the existing tasks (where possible) to avoid abrupt termination), recovery (autoscaler starts the VM again or launches a new instance in the multi-cloud environment), and reallocation (pending or drained tasks are reallocated to underutilized or balanced VMs in the multi-cloud environment).

#### 3.9.6.4. Formal Response Function

The overall response policy can be summarized as Equation. (53):

$$Response(VM_i) = \begin{cases} Continue\ Allocation, & State(VM_i) = Healthly \\ Reduce\ Traffic + Monitor, & State(VM_i) = Warning \\ Isolate + Drain + Replace, & State(VM_i) = Faulty \end{cases} \quad (53)$$

#### 3.9.6.5. Example Response Execution

- A VM categorized as Warning due to high latency (Fault Score=0.65) receives only short-lived, low-resource tasks while being closely monitored for further changes.
- A VM categorized as Faulty (Fault Score = 0.81, heartbeat loss) is immediately isolated, ongoing tasks are migrated to stable VMs, and an autoscaler provisions a replacement.

This multi-layered response mechanism allows CloudGuard to provide fault-tolerant continuity in scheduling, so that localized VM failures do not spread to large-scale service failures. Step 7 is followed by forwarding the refined response outputs to the load balancer to adapt the system-wide.

### 3.9.7. Step 7: Feedback to Load Balancer (Fuzzy Load Balancer-FLB)

The last step in the fault identification pipeline of CloudGuard is to send actionable feedback to the load balancing layer of the system. The step will make sure that the fault awareness that has been attained in Steps 1-6 is successfully translated into adaptive scheduling decisions throughout the multi-cloud environment.

#### 3.9.7.1. Integration with Fuzzy Load Balancer (FLB)

The fault score and fault category of every VM is constantly sent to the Fuzzy Load Balancer (FLB). The FLB, a policy maker that controls the distribution of tasks between VMs and clouds, modifies its decision-making policies according to these inputs to guarantee:

- Faulty VMs are excluded from future scheduling pools.
- Warning VMs are cautiously scheduled with reduced workload inflow.
- Healthy VMs are prioritized for incoming tasks.

This dynamic reconfiguration allows the FLB to maintain system reliability while adapting to real-time VM health changes.

#### 3.9.7.2. Feedback-Control Policy

Formally, the updated scheduling policy $\pi_{FLB}$ is expressed as Equation. (54):

$$\pi_{FLB}(\mathcal{T}_j) = \begin{cases} Assign(\mathcal{T}_j, VM_i), & State(VM_i) = Healthly \\ Assign(\mathcal{T}_j, VM_i)\ with\ reduced\ weight, & State(VM_i) = Warning \\ Exclude(VM_i), & State(VM_i) = Faulty \end{cases} \quad (54)$$

Here, *Assign* ($T_j$, $VM_i$) denotes the mapping of task *j* to VM *i*. Reduced weight reflects probabilistic down-scaling of allocations to degraded VMs. *Exclude*($VM_i$) ensures that faulty VMs are entirely isolated from the scheduling pool.

#### 3.9.7.3. System-Wide Reallocation

Once the FLB incorporates fault-aware feedback, the federated scheduling framework (ResFedLB+RL-COSCoati) is updated to redistribute pending workloads across the multi-cloud ecosystem. This includes:

- Diverting new tasks away from degraded/faulty VMs.
- Prioritizing underloaded and balanced VMs for reassigned workloads.
- Triggering cross-cloud offloading (Section 3.8) if intra-cloud resources are insufficient.

#### 3.9.7.4. Example Feedback Scenario

- A VM identified as Faulty (VM Crash) is removed from the allocation pool. The FLB diverts tasks originally scheduled for this VM to a set of underloaded VMs.
- A VM in a Warning state (Resource Saturation) remains active but is assigned a smaller fraction of new tasks, while being continuously monitored.
- Only Healthy VMs retain unrestricted task allocations, ensuring overall SLA compliance.

The framework provides a closed-loop fault-tolerant scheduling system by integrating the health insights of CloudGuard into the FLB. The load balancer is not reactive to faults, but proactive in adjusting to predicted and observed faults, which guarantees high availability, low downtime, and efficient resource utilization in multi-cloud environments. Figure 15 illustrates the graphical representation of fault identification.

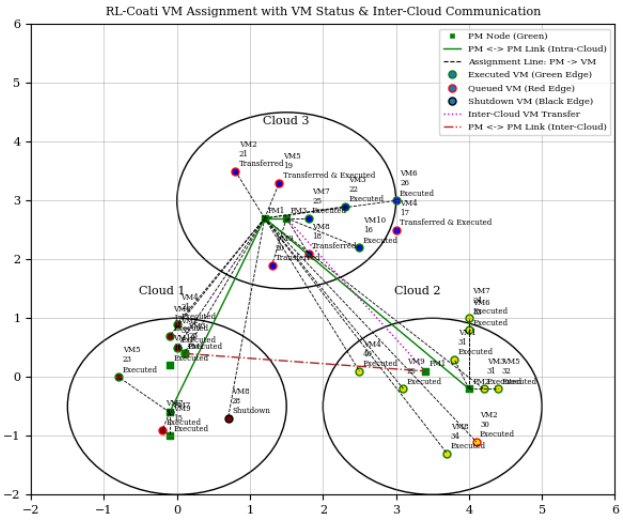


Figure 15. Fault identification.

## 3.10. Task Completion

The task completion stage is the final stage of the

suggested multi-cloud load balancing framework. At this point, the sum of all the above-mentioned mechanisms, including data grouping, federated learning-based resource forecasting, VM workload prediction, RL-COSCoati scheduling, task execution, VM consolidation, cross-cloud communication, and CloudGuard fault identification, are all combined in order to provide a smooth processing of tasks.

The tasks are implemented on their respective VMs and continuously monitored with real-time feedback loops that allow the dynamic reallocation of the tasks in case faults, overloads, or SLA violations are detected. The framework ensures that tasks are delivered with high reliability, low latency, less energy use, and maintained service quality by combining fault-tolerant scheduling, resource-aware forecasting, and adaptive consolidation.

Therefore, the task completion phase confirms the usefulness of the proposed system, so that despite the changing workloads, resource heterogeneity, and unforeseen faults, multi-cloud environments can provide efficient, resilient, and SLA-compliant task execution.

## 4. Result and Discussion

This part provides the experimental analysis of the suggested multi-cloud load balancing model. The aim is to evaluate its performance in the aspects of workload forecasting accuracy and task allocation performance, and compare it with the current state-of-the-art methods.

### 4.1. Experimental Setup

The suggested framework is implemented in Python and its performance is strictly tested in controlled experimental conditions. To ensure reproducibility and to validate the robustness of the proposed ResFedLB-CLAFT-Net-RL-COSCoati-CloudGuard framework, all experiments were conducted on a controlled multi-cloud simulation testbed. The evaluation included workload forecasting, task scheduling, fault detection accuracy, and resource utilization analysis. The following subsections describe the hardware configuration, software environment, datasets, and benchmarking methodology.

In the case of the Workload Forecasting Analysis, the most important evaluation measures have been accuracy, precision, recall, F1-score, task completion time (s) and energy consumption (J). The proposed model has been compared to the performance of four existing methods, including: Random Forest (RF), CNN-LSTM, FL-CNN, and C-CNN-LSTM-A that have been extensively used in previous cloud resource forecasting work. In the analysis on task allocation performance, the analysis has been based on a wider range of system-level measures, such as average task completion time (s), average task wait time (s), task success rate (percent), CPU utilization (percent), memory utilization (percent), energy consumption (J), migration overhead, RL cumulative reward, and throughput. The suggested RL-COSCoati scheduling method has been contrasted with the known baselines like heuristic scheduling, RL-GA-Opt, Q-Learning, and RL-ACO. These metrics and baseline comparisons are chosen to fully justify the predictive accuracy of workload forecasting as well as the operational efficiency of task allocation in heterogeneous multi-cloud settings.

#### 4.1.1. Hardware and Runtime Environment

The experiments were executed using a hybrid setup consisting of local servers and simulated multi-cloud nodes.

Table 1. Hardware configuration.

| Component | Specification |
|---|---|
| **CPU** | Intel xeon gold 6230R (20 cores @ 2.10 GHz) |
| **GPU** | NVIDIA RTX A6000 (48 GB VRAM) |
| **RAM** | 128 GB DDR4 |
| **Storage** | 2 TB NVMe SSD |
| **Network** | 10 Gbps Virtual internal network |
| **Multi-cloud nodes** | 12 VM instances (4 AWS-type, 4 GCP-type, 4 Azure-type equivalents) |

Table 2. Software/libraries.

| Component | Version |
|---|---|
| **OS** | Ubuntu server 22.04 LTS |
| **Python** | 3.11 |
| **TensorFlow** | 2.15 |
| **PyTorch** | 2.2 |
| **Federated learning framework** | Flower FL 1.6 |
| **RL toolkit** | Stable-saselines3+custom COS-coati engine |
| **Fuzzy logic library** | scikit-fuzzy 0.4 |
| **Cloud simulation** | CloudSim plus 7.3 |
| **Containerization** | Docker 24.0 |

#### 4.1.2. Execution Modes:

- Federated Training: 20 communication rounds, 12 clients, non-IID workload distribution
- Scheduling Simulation: 50,000 cloudlets (tasks), mixed URLLC+delay-tolerant workloads

- Fault Injection: 50+fault types (thermal overload, VM stragglers, network delay spikes, soft faults)

## 4.2. Workload Forecasting Analysis: Proposed vs. Existing

The results of the workload forecasting have been provided in Table 3 and also demonstrated in Figure 16 where the proposed framework has performed better in all the evaluation metrics than the existing methods. This high performance has been mainly achieved through the combination of ResFedLB framework and CLAFT-Net (CNN LSTM Attention with Forecasting Transformer Network) local training module. This has facilitated precise modality-conscious learning and sound resource usage prediction in multi-cloud settings.

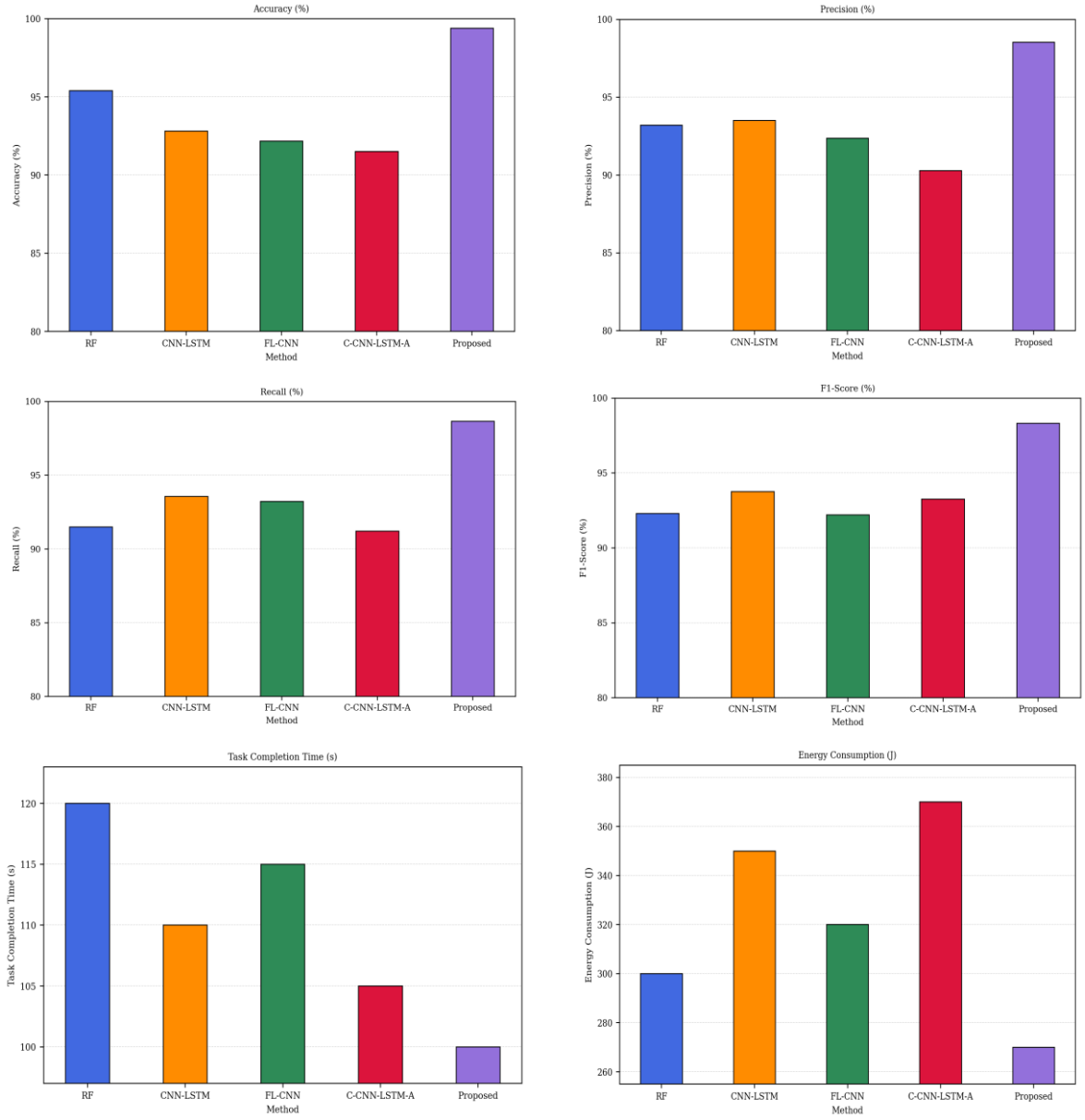


Figure 16. Workload forecasting analysis between proposed and existing.

The proposed model has a high accuracy of 99.38, which is much higher than RF (95.4%), CNN-LSTM (92.81%), FL-CNN (92.17%), and C-CNN-LSTM-A (91.5%). Federated aggregation in ResFedLB has enabled the global model to use distributed data without centralizing raw inputs, which results in better generalization than centralized baselines.

Table 3. Workload forecasting comparison.

| Method | Accuracy (%) | Precision (%) | Recall (%) | F1-score (%) | Task completion time (s) | Energy consumption (J) |
|---|---|---|---|---|---|---|
| **RF** | 95.4 | 93.2 | 91.5 | 92.3 | 120 | 300 |
| **CNN-LSTM** | 92.81 | 93.51 | 93.57 | 93.76 | 110 | 350 |
| **FL-CNN** | 92.17 | 92.38 | 93.23 | 92.21 | 115 | 320 |
| **C-CNN-LSTM-A** | 91.5 | 90.29 | 91.22 | 93.26 | 105 | 370 |
| **Proposed** | 99.38 | 98.54 | 98.66 | 98.32 | 100 | 270 |

To be more specific, the given approach has achieved 98.54, whereas RF, CNN-LSTM, FL-CNN, and C-CNN-LSTM-A have reached 93.2, 93.51, 92.38, and 90.29, respectively. The increased accuracy has shown that the proposed framework has been capable of reducing false positives in a better way, which guarantees a good identification of workload patterns.

Recall value has also been improved significantly with the proposed model reaching 98.66, which is better than RF (91.5%), CNN-LSTM (93.57%), FL-CNN (93.23%), and C-CNN-LSTM-A (91.22%). This shows that the suggested approach has been more receptive to the capturing of the various workload variations, which is essential in fault-tolerant scheduling.

The proposed model has attained 98.32 in terms of F1-score, compared to 92.3, 93.76, 92.21, and 93.26 of RF, CNN-LSTM, FL-CNN, and C-CNN-LSTM-A, respectively. The equal increase in both accuracy and recall has demonstrated that ResFedLB with CLAFT-Net has offered a more comprehensive prediction performance.

In addition to measures of accuracy, the efficiency of the system level has been confirmed. The proposed framework has also minimized the time of completion of the tasks to 100 seconds as compared to RF (120s), CNN-LSTM (110s), FL-CNN (115s), and C-CNN-LSTM-A (105s). On the same note, the energy use has been reduced to 270 J, which is better than RF (300 J), CNN-LSTM (350 J), FL-CNN (320 J), and C-CNN-LSTM-A (370 J). This has minimized the resource-consciousness of the federated learning framework, guaranteed effective use of energy, and Service-Level Agreements (SLAs).

In general, the comparative analysis in Table 1 and Figure 16 has shown that the proposed ResFedLB + CLAFT-Net framework has always provided better forecasting results. The federated architecture has reduced the overheads of data transfer and maintained privacy, whereas the hybrid CNN-LSTM-Attention-Transformer architecture has learned spatial and temporal dependencies of workloads, which has allowed making accurate, low-latency, and energy-efficient predictions.

### 4.3. Analysis on Task Allocation Performance: Proposed vs. Existing

The results of the task allocation have been summarized in Table 4 and plotted in Figure 17 where the proposed framework has performed better than the current heuristic and reinforcement learning-based schedulers. This enhanced efficiency has been credited to its hybrid scheduling policy that integrates RL coordination policy, opposition-based learning and coati-inspired search dynamics.

Table 4. Comparison on task allocation performance.

| Model | Avg Task Completion Time(s) | Avg Task Wait Time(s) | Task Success Rate (%) | CPU Utilization(%) | Memory Utilization(%) | Energy Consumption(J) | Migration Overhead | RL Cumulative Reward | Throughput |
|---|---|---|---|---|---|---|---|---|---|
| **Proposed** | 12.3 | 3.5 | 98.20% | 97% | 98.60% | 150.2 | 5.1 | 345.7 | 120 |
| **Heuristic** | 15.8 | 5.2 | 92% | 92% | 93% | 180.4 | 7.6 | 280.3 | 100 |
| **RL-GA-Opt** | 18.1 | 6 | 92% | 91.70% | 94% | 190 | 9 | 260 | 90 |
| **Q-Learning** | 20 | 7.1 | 94% | 93% | 92% | 210.5 | 10.5 | 240 | 85 |
| **RL-ACO** | 22.5 | 8.3 | 92% | 95% | 94.20% | 230.1 | 12 | 220 | 80 |

The proposed approach has recorded 12.3 seconds, which is much lower than Heuristic 15.8s, RL-GA-Opt 18.1s, Q-Learning 20s, and RL-ACO 22.5s in terms of average time to complete a task. This has enhanced the fact that workloads have been distributed to underloaded and balanced VMs to process tasks faster.

Mean time to wait in the tasks has also been reduced to 3.5s, as opposed to 5.2s Heuristic, 6s RL-GA-Opt, 7.1s Q-Learning, and 8.3s RL-ACO. This decrease has demonstrated that the proposed model has been proactive in assigning tasks before queues built up thus reducing scheduling latency.

The success rate of the task has been 98.20 which is better than Heuristic 92%, RL-GA-Opt 92%, Q-Learning 94% and RL-ACO 92%. The steady enhancement has drawn attention to the fact that the system can ensure adherence to SLA by preventing failure of tasks and overloading situations.

The model has also been proven to be effective through the analysis of resource utilization. CPU and memory utilization have been recorded as 97 and 98.60 respectively, which are higher than the competing methods: Heuristic 92% CPU, 93% memory, RL-GA-Opt 91.70% CPU, 94% memory, Q-Learning 93% CPU, 92% memory, and RL-ACO 95% CPU, 94.20% memory. Such outcomes have shown that there has been maximum utilization of resources without overloading.

Energy wise, the proposed framework has used only 150.2 J which is significantly less than Heuristic 180.4 J, RL-GA-Opt 190 J, Q-Learning 210.5 J, and RL-ACO 230.1 J. The lower energy footprint has been as a result of efficient avoidance of resource hotspots and overloaded VMs.

In terms of migration overhead, the proposed model has recorded 5.1 whereas Heuristic, RL-GA-Opt, Q-Learning, and RL-ACO have recorded 7.6, 9, 10.5, and 12, respectively. The reduced cost of migration has shown that load balance has been achieved with reduced VM transfers.

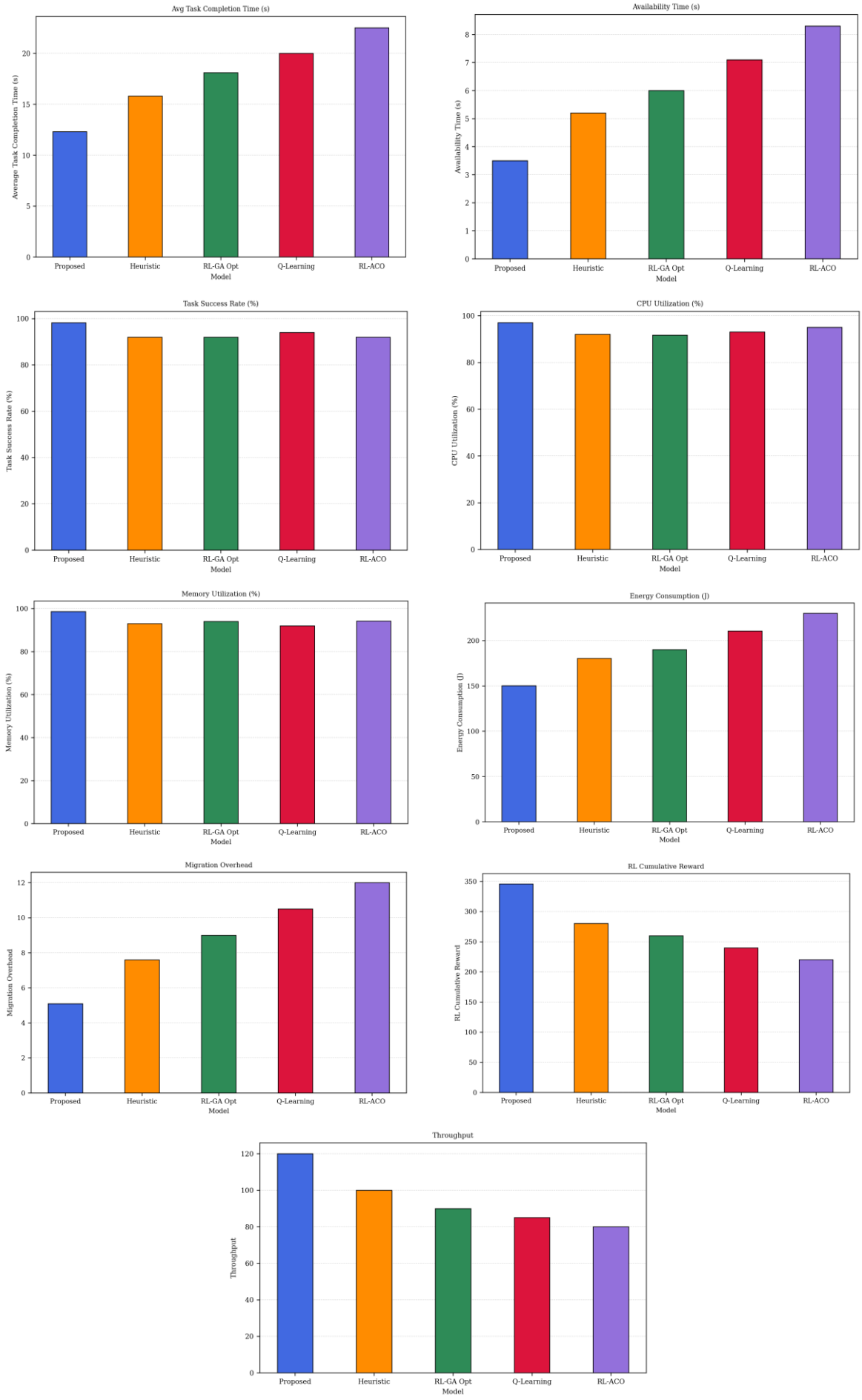


Figure 17. Comparison on task allocation performance between proposed and existing.

The usefulness of the reinforcement learning aspect has also been confirmed by the cumulative reward in which the suggested approach has achieved 345.7, which is much greater than Heuristic 280.3, RL-GA-Opt 260, Q-Learning 240, and RL-ACO 220. This has demonstrated that the scheduling agent has always been drawn to the best policies.

Lastly, the proposed system has achieved throughput of 120 tasks/s, which is higher than Heuristic 100, RL-GA-Opt 90, Q-Learning 85, and RL-ACO 80. This has enhanced the fact that the model has made it possible to execute tasks faster and be more scalable to dynamic workloads.

In general, the comparative analysis in Table 2 and Figure 17 has shown that the proposed framework, driven by the RL-COSCoati scheduling strategy, has provided substantial gains in terms of latency, utilization, energy efficiency, and reliability compared to both heuristic and traditional reinforcement learning-based baselines.

## 5. Conclusions

This work has introduced a smart multi-cloud load balancing model that integrated federated learning, forecasting transformers, reinforcement learning, and fuzzy-based fault management to overcome the issue of dynamic workload scheduling and resilience in heterogeneous settings. The suggested system has incorporated ResFedLB to provide resource-aware federated workload forecasting, CLAFT-Net as the local training module, RL-COSCoati to provide adaptive task scheduling, and CloudGuard to identify faults and manage resilient VM. The whole structure has been done in Python and it is modular and reproducible in case of experimental analysis. The results of the evaluation have shown that it has significantly improved with a forecasting accuracy of 99.38, F1-score of 98.32, 97% CPU usage, and a 98.20% task success rate, which is better than the state-of-the-art heuristic and reinforcement learning-based schedulers. These results have validated the usefulness of the suggested method in offering low-latency, energy-saving, and fault-tolerant execution of tasks. Socially, this work has led to scalable, sustainable, and reliable cloud services, which are essential in supporting real-time applications like healthcare monitoring, financial systems, intelligent transportation, and smart city infrastructures. The framework has offered a route to lowering the operational expenses, enhancing energy usage, and service-level agreements in real-world multi-cloud ecosystems by increasing efficiency and resilience. Nonetheless, the model that has been proposed still exhibits some drawbacks. To begin with, while ResFedLB significantly cuts down on the amount of data sharing, the communication latency between federated nodes could go up in heavily distributed environments. Moreover, the efficacy of RL-COSCoati directly correlates with the reward structure and in extremely diverse multi-cloud infrastructures, it might need further tuning. CloudGuard's fuzzy inference rules are powerful but could be enhanced even more to decipher intricate failure modes. The upgrades for the future will be all about the incorporation of self-evolving federated optimization, creation of cross-cloud intention-aware RL strategies, and joining of explainable AI methods to increase the transparency in the scheduling and fault-detection decisions. The subject of extending the framework to serverless and edge-cloud continuum was also proposed as a way to further test its scalability and adaptability. In addition, implementing the framework on real cloud testbeds and serverless systems would also confirm its flexibility and scalability to the next-generation intelligent cloud systems.

## References

[1] Alatawi M., "Optimizing Multitenancy: Adaptive Resource Allocation in Serverless Cloud Environments Using Reinforcement Learning," *Electronics*, vol. 14, no. 15, pp. 1-25, 2025. https://doi.org/10.3390/electronics14153004

[2] Asghari A., Sohrabi M., and Yaghmaee F., "Task Scheduling, Resource Provisioning, and Load Balancing on Scientific Workflows Using Parallel SARSA Reinforcement Learning Agents and Genetic Algorithm," *Journal of Supercomputing*, vol. 77, no. 3, pp. 1-25, 2021. https://doi.org/10.1007/s11227-020-03364-1

[3] Bal P., Mohapatra S., Das T., Srinivasan K. and Hu Y., "A Joint Resource Allocation, Security with Efficient Task Scheduling in Cloud Computing Using Hybrid Machine Learning Techniques," *Sensors*, vol. 22, no. 3, pp. 1-16, 2022. https://doi.org/10.3390/s22031242

[4] Belgacem A., Mahmoudi S. and Kihl M., "Intelligent Multi-Agent Reinforcement Learning Model for Resources Allocation in Cloud Computing," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 6, pp. 2391-2404, 2022. https://doi.org/10.1016/j.jksuci.2022.03.016

[5] Chen Z., Hu J., Min G., Luo C. and El-Ghazawi T., "Adaptive and Efficient Resource Allocation in Cloud Datacenters Using Actor-Critic Deep Reinforcement Learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 8, pp. 1911-1923, 2021. DOI: 10.1109/TPDS.2021.3132422

[6] Chen Z., Zhou H., Du S., Liu J., and et al., "Reinforcement Learning-Based Resource Allocation and Energy Efficiency Optimization for a Space-Air-Ground-Integrated Network," *Electronics*, vol. 13, no. 9, pp. 1-21, 2024. https://doi.org/10.3390/electronics13091792

[7] Danino T., Ben-Shimol Y., and Greenberg S., "Container Allocation in Cloud Environment Using Multi-Agent Deep Reinforcement Learning," *Electronics*, vol. 12, no. 12, pp. 1-15, 2023. https://doi.org/10.3390/electronics12122614

[8] Elrod M., Mehrabi N., Amin R., Kaur M., and et al., "Graph Based Deep Reinforcement Learning Aided by Transformers for Multi-Agent Cooperation," *arXiv preprint*, arXiv:2504.08195v1, pp. 1-6, 2025. https://doi.org/10.48550/arXiv.2504.08195

[9] Houssein E., Gad A., Wazery Y., and Suganthan P., "Task Scheduling in Cloud Computing Based on Meta-Heuristics: Review, Taxonomy, Open Challenges, and Future Trends," *Swarm and Evolutionary Computation*, vol. 62, pp. 100841, 2021. DOI: 10.1016/j.swevo.2021.100841

[10] Ishtaiwi A., Nabot A., Alzubi O., Ramadan A., and et al., "HECOCP: Hybrid Edge-Cloud Optimistic Concurrency Protocol for Sensor Data Transactional Services," *International Arab Journal of Information Technology*, vol. *22*, no. 4, pp. 756-768, 2025. DOI: https://doi.org/10.34028/iajit/22/4/10

[11] Jayanetti A., Halgamuge S. and Buyya R., "Deep Reinforcement Learning for Energy and Time Optimized Scheduling of Precedence-Constrained Tasks in Edge-Cloud Computing Environments," *Future Generation Computer Systems*, vol. 137, pp. 14-30, 2022. https://doi.org/10.1016/j.future.2022.06.012

[12] Kruekaew B. and Kimpan W., "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

[13] Lahza H., BR S. and Lahza H., "Adaptive Multi-Objective Resource Allocation for Edge-Cloud Workflow Optimization Using Deep Reinforcement Learning," *Modelling*, vol. 5, no. 3, pp. 1298-1313, 2024. https://doi.org/10.3390/modelling5030067

[14] Li Y., Zhang X., Zeng T., Duan J., and et al., "Task Placement and Resource Allocation for Edge Machine Learning: A GNN-Based Multi-Agent Reinforcement Learning Paradigm," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 12, pp. 3073-3089, 2023. DOI: 10.1109/TPDS.2023.3313779

[15] Monon J., Barua D., and Khan M., "Enhancing Heterogeneous Multi-Agent Cooperation in Decentralized MARL via GNN-Driven Intrinsic Rewards," *arXiv preprint*, arXiv:2408.06503v4, pp. 1-9, 2024. https://doi.org/10.48550/arXiv.2408.06503

[16] Rjoub G., Bentahar J., Abdel Wahab O. and Bataineh A., "Deep and Reinforcement Learning for Automated Task Scheduling in Large-Scale Cloud Computing Systems," *Concurrency and Computation Practice and Experience*, vol. 33, no. 23, pp. 1-13, 2021. https://doi.org/10.1002/cpe.5919

[17] Sanjalawe Y., Al-E'mari S., Fraihat S. and Makhadmeh S., "AI-Driven Job Scheduling in Cloud Computing: A Comprehensive Review," *Artificial Intelligence Review*, vol. 58, no. 7, pp. 11-113, 2025. DOI:10.1007/s10462-025-11208-8

[18] Sharma K and Patel N., "Optimizing Cloud Resource Management with Advanced Scheduling Algorithms and Reinforcement Learning," *Indian Journal of Science and Technology*, vol. 18, no. 17, pp. 1356-1364, 2025. DOI:10.17485/IJST/v18i17.434

[19] Shiva Rama Krishna M. and Mangalampalli S., "A Novel Fault-Tolerant Aware Task Scheduler Using Deep Reinforcement Learning in Cloud Computing," *Applied Sciences*, vol. 13, no. 21, pp. 1-26, 2023. https://doi.org/10.3390/app132112015

[20] Siddesha K., Jayaramaiah G., and Singh C., "A Novel Deep Reinforcement Learning Scheme for Task Scheduling in Cloud Computing," *Cluster Computing*, vol. 25, no. 6, pp. 4171-4188, 2022. DOI:10.1007/s10586-022-03630-2

[21] Swarup S., Shakshuki E. and Yasar A., "Task Scheduling in Cloud Using Deep Reinforcement Learning," in *Proceedings of Procedia Computer Science*, vol. 184, pp. 42-51, 2021. https://doi.org/10.1016/j.procs.2021.03.016

[22] Wang Y and Yang X., "Intelligent Resource Allocation Optimization for Cloud Computing via Machine Learning," *arXiv preprint*, arXiv:2504.03682v1, pp. 55-63, 2025. https://doi.org/10.48550/arXiv.2504.03682

[23] Yan Y., Yao S., Wang H. and Gao M., "Index selection for NoSQL database with Deep Reinforcement Learning," *Information Sciences*, vol. 561, pp. 20-30, 2021. https://doi.org/10.1016/j.ins.2021.01.003

[24] Zhang G., "Cloud Computing Convergence: Integrating Computer Applications and Information Management for Enhanced Efficiency," *Frontiers in Big Data*, vol. 8, pp. 1-8, 2025. https://doi.org/10.3389/fdata.2025.1508087

[25] Zhao B., Huo M., Li Z., Feng W., and et al., "Graph-Based Multi-Agent Reinforcement Learning for Collaborative Search and Tracking of Multiple UAVs," *Chinese Journal of Aeronautics*, vol. 38, no. 3, pp. 1-15, 2025. https://doi.org/10.1016/j.cja.2024.08.045

[26] Zhao T., Chen T., and Zhang B., "QMIX-GNN: A Graph Neural Network-Based Heterogeneous Multi-Agent Reinforcement Learning Model for

Improved Collaboration and Decision-Making," *Applied Sciences*, vol. 15, no. 7, pp. 1-18, 2025. https://doi.org/10.3390/app15073794

**Damodhar MummiReddy** is a research scholar from the Department of Computer Science and Engineering, Sri Venkateswara University College of Engineering, Tirupati, India, since 2019. He received his BTech and MTech degrees in Computer Science and Engineering from JNTUA, Anantapuramu. He worked as a faculty member and has more than 8 years of teaching experience in different engineering colleges in and around Tirupati, India. He has published 4 papers in Scopus/SCI indexed conferences and journals. His areas of interest include Distributed Systems, Cloud Computing, Advanced Computing, Machine Learning and Deep Learning.

**Choda Durga Venkata Subba Rao** has been serving as Professor of Computer Science and Engineering at Sri Venkateswara University since 2009. At present, he is the Vice-Principal of the University College of Engineering, Sri Venkateswara University. He also serves as Chairman, Board of Studies in CSE (AI), Sri Venkateswara University, and Chairman, Board of Studies in CSE, Sri Padmavathi Women's University. He worked as a faculty member at NIT Warangal during 1991-92. His areas of interest include Distributed Systems, Wireless Networks, and Advanced Computing. Choda Rao is a Fellow of the IETE and a fellow of the Institution of Engineers (India). He has published 85 papers in international journals. He has guided 46 M.Tech. theses and 8 Ph.D. theses, and the research work of 6 Ph.D. scholars is in progress. He has authored 5 textbooks, 4 book chapters, and holds 5 patents. Choda Rao he visited Austria, Hong Kong, the Netherlands, Belgium, Thailand, Germany, and Singapore on academic assignments. He was conferred the State Best Teacher Award by the Government of Andhra Pradesh, the Teacher of Excellence Award by Sri Venkateswara University, the DTA Excellence Award by the Delhi Telugu Academy, the Rashtriya Gaurav Award by IIFS, and the Best Paper Award at the IAENG International Conference, Hong Kong.