Received: 27 May 2025 | Revised: 30 September 2025 | Accepted: 16 October 2025 | Published online: 12 December 2025

RESEARCH ARTICLE

Journal of Computational and Cognitive Engineering
2025, Vol. 00(00) 1-17
DOI: 10.47852/bonviewJCCE52026266

Adaptive Fault-Tolerant Scheduling in Dynamic Clouds

Chetankumar Kalaskar[1] and **Thangam Somasundaram**[1,*]

[1] *Amrita School of Computing, Amrita Vishwa Vidyapeetham, India*

Abstract: The transition from product-centric to service-driven paradigms in cloud computing has enabled the on-demand provisioning of scalable resources. However, the extensive operational complexity and interdependent infrastructure of cloud environments make them highly susceptible to failures, posing significant challenges to ensuring robust fault tolerance in dynamic and distributed systems. This study introduces a novel Hybrid Value-Policy Double Deep Q-Learning (DDQL) Scheduling approach, integrating value-based DDQL with policy-based reinforcement learning techniques, which are Proximal Policy Optimization and Soft Actor-Critic, to enhance fault-tolerant scheduling dynamically. Combining the advantages of both learning paradigms, the proposed method improves decision-making efficiency, enhances system resilience against failures, and optimizes scheduling performance in real-time cloud environments. Improved mean time between failures, mean time to recovery, failure recovery time, and fault tolerance success rate are guaranteed by the Hybrid DDQL scheduler's dynamic adjustments to scheduling and resource allocation algorithms. Also, it reduces the ratio of task deadline miss, energy consumption, and response time and maximizes the throughput. These results confirm the flexibility and strength of the model in dynamic and unreliable real-time cloud computing systems.

Keywords: fault tolerance, real-time cloud computing, hybrid reinforcement learning, Double Deep Q-Learning (DDQL), Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC)

1. Introduction

Cloud computing has essentially changed the resource and service allocation of computer products to service-based distribution. With the aid of cloud systems, consumers can acquire elastic, expandable, and affordable computing power without the need to own any infrastructure. Demand for computational efficiency and high availability, and dependability drives this trend, which has accelerated the fast acceptance of cloud solutions across numerous industries, including healthcare, banking, education, and manufacturing [1]. However, the likelihood of system failures rises as the complexity and dynamism of cloud systems develop—particularly in light of heterogeneous, multi-tenant workloads and real-time service delivery. Among cloud systems, the main sources of job interruptions include hardware failures, software problems, network outages, and resource congestion.

These failures could compromise application performance and break service-level agreements (SLAs) in mission-critical systems depending on real-time data processing and decision-making [2]. Fault-tolerant scheduling systems have been investigated extensively in order to solve these problems and guarantee service continuity even in the event of breakdowns. Under both static and somewhat dynamic environments, conventional fault tolerance methods such as redundancy-based solutions, checkpointing, and replication have proven successful. However, because of too high overhead, resource inefficiencies, and limited flexibility, their relevance reduces in highly dynamic, large-scale, real-time cloud systems [3]. More intelligent and flexible fault-tolerant systems are needed, given the increasing interdependence of jobs, varying execution durations, and nondeterministic behaviour of workloads. Reinforcement learning (RL), a subfield of machine learning, has lately become a powerful tool for real-time decision-making in complex environments. RL agents acquire optimal strategies by interaction with their surroundings; hence, RL is ideal for cloud computing's adaptive scheduling and resource allocation [4]. Common in real-world cloud systems, classic RL approaches such as Q-Learning or Deep Q-Networks (DQNs) generally suffer from sluggish convergence, large volatility, and inferior performance in continuous action environments. This work suggests a hybrid value-policy Double Deep Q-Learning (DDQL) model to solve these challenges by combining the capabilities of value-based RL (e.g., DDQL) with policy-based techniques such Soft Actor-Critic (SAC) and Proximal Policy Optimization (PPO). The combination of these paradigms will ensure robust, elastic and smart scheduling through flexibility to real-time variations in workload and infrastructure state. Fault tolerance in cloud system deals with the entire

lifetime of fault prediction, isolation, mitigating, and adaptive rescheduling, as opposed to seeing and fixing faults. There are a number of issues preventing successful use of fault-tolerant scheduling in dynamic clouds:

Cloud platforms typically handle compute-intensive, latency-sensitive, and data-heavy tasks, each with different QoS requirements and fault sensitivity. Scheduling in such an environment must be responsive to workload dynamics, user priorities, and resource availability in real time, such as follows [5]:

1) The stochastic nature of cloud failures complicates prediction and diagnosis. Failure types (e.g., transient and permanent), durations, and root causes often remain unknown until service degradation occurs. Delayed or inaccurate failure detection may result in deadline violations or data loss [6].
2) Conventional fault tolerance strategies like checkpointing or replication introduce computational and storage overhead. In real-time systems with tight latency constraints, such overheads may violate performance goals or inflate operational costs [7].

***Corresponding author:** Thangam Somasundaram, Amrita School of Computing, Amrita Vishwa Vidyapeetham, India. Email: s_thangam@blr.amrita.edu

3) Deterministic scheduling is necessary for real-time applications (such as online gaming, video conferencing, and autonomous control) to fulfill deadlines. Any delay caused by task re-execution, recovery, or rescheduling must be minimal to avoid SLA penalties and user dissatisfaction [8].
4) Cloud environments are nonstationary, with unpredictable network conditions, user demand, and node health changes. Fixed or rule-based scheduling algorithms cannot effectively adapt to these fluctuations, leading to inefficient resource use and poor fault resilience [9].
5) The state-action space expands exponentially with the number of tasks and virtual machines (VMs). Many RL approaches suffer from scalability issues or fail to generalize learned policies to new or unseen system configurations [10].

To address these multifaceted challenges, a hybrid RL approach leverages DDQL's decision efficiency, PPO's stability and sample efficiency, and SAC's continuous control capabilities. The hybrid method dynamically learns from the environment, reduces overhead, and supports robust scheduling even under high variability and failure conditions.

This article presents a new hybrid RL architecture to address these issues and significantly improve fault-tolerant scheduling in real-time cloud computing settings. The key contributions are as follows:

1) A new Hybrid Value-Policy model is developed that integrates DDQL, PPO, and SAC. This fusion addresses the limitations of individual RL models by combining the stable convergence of policy-based methods with the precision of value-based strategies, enabling more adaptive and robust scheduling decisions [11].
2) The study develops a scheduling architecture based on real-time input, system conditions, and failure signals that dynamically reallots resources and modifies execution plans. By learning ideal policies through constant environment interaction, the framework provides quick recovery and proactive fault handling.
3) In many important criteria, including fault tolerance success rate (FTSR), mean time to recovery (MTTR), mean time between failures (MTBF), task deadline miss ratio (DMR), energy consumption (EC), and system throughput, comprehensive experimental evaluation shows that the proposed hybrid scheduler beats conventional scheduling techniques. These developments point to increased resilience and dependability of services in dynamic changes in workload and infrastructure flaws.
4) The proposed approach is simulated in a real-time environment, simulated on the CloudSim and OpenStack benchmark datasets and technologies, and performed in a simulated cloud environment with real-time tasks. The baseline models, such as First-Come-First-Serve (FCFS), Round Robin, and standard DDQL, have been found to be scalable and effective, as comparative studies help validate the scalability and effectiveness of the hybrid strategy.
5) Apart from the empirical improvements, the hybrid approach presents a learning-based decision model that spans several cloud workloads and system conditions. This qualifies for incorporation into real-world cloud management systems trying to reduce service outages and improve operational continuity.

Here is the outline for the rest of the paper: Section 2 looks at some of the more established methods for fault-tolerant scheduling and more modern hybrid techniques, including RL for cloud resource management. Section 3 formulates the scheduling problem and outlines key system objectives. Section 4 details the proposed hybrid scheduling methodology, including integrating the DDQL model and PPO and SAC techniques. Section 5 describes the system architecture and workflow. Section 6 presents the experimental setup, simulation tools, and performance metrics. Section 7 discusses the results and comparative analysis. The study and its implications for future research are summarized in Section 8.

2. Related Works

Efficient resource allocation, robust fault tolerance, and adequate load balancing are crucial for optimizing resource utilization in distributed computing environments. The degree to which fault tolerance impacts resource usage largely depends on the specific allocation strategy employed. By applying well-structured resource management techniques, systems can achieve higher performance and reliability. Most of the current fault tolerance solutions, however, tend to compromise resource availability and system efficiency when they fail unexpectedly. Other parameters, such as the mean use of resources and throughput, play a great role in the performance of the entire system. The dynamic nature of cloud systems and their unreliable nature of resources may make operations unstable, thus exposing them to faults, and can even frustrate load balancing and application execution. Fault tolerance mechanisms are supposed to continue service delivery despite faults or failures of a system. It involves the early detection of a defect to minimize risks, system enableability to enable it to continue with the activities to completion. Although different methods have been suggested to deal with these issues, more efficient and improved measures can still be developed to improve the use of resources after failure recovery.

A number of time management techniques have been suggested to enhance performance in the cloud environment. A multi-objective model using an enhanced differential evolution algorithm was introduced to optimize time and cost in cloud systems, though it lacks the adaptability to dynamic task variations. Their framework demonstrated that RL can effectively adapt task allocation strategies to minimize failures under varying workload conditions. However, the method primarily relied on a single value-based learning paradigm [12]. Framework applied a GNN-based approach for fault tolerance and load balancing in cloud computing. By modelling task dependencies as graph structures, DMQOS effectively improved resource utilization and reduced task failures under dynamic workloads [13]. FCFS remains a basic approach [14] but lacks job elimination mechanisms. More advanced methods include heuristic-based scheduling [15], combining the Modified Analytic Hierarchy Process, bandwidth-aware scheduling, and divide-and-conquer strategies. Although these methods aim to optimize allocation, resource efficiency is not fully addressed. The QL-Heterogeneous Earliest Finish Time (HEFT) model, which integrates Q-learning and the HEFT algorithm [16], seeks to reduce makespan but neglects resource utilization. Algorithms, such as Minimum Completion Time, propose how cloud approaches can hybridize for efficient data storage and security [17, 18]. Using precedence graphs and Johnson's algorithm, a dynamic three-step sequencing method was also proposed to reduce makespan [19]. Cloud platforms such as AWS have supported task scheduling strategies such as the Periodic Min-Max Algorithm for multi-robot applications [20]. More recently, scheduling improvements have included Priority-Based Fair Scheduling (PBFS) [21], which uses the Earliest Gap Shortest Job First strategy, and its variant, SG-PBFS [22], designed to optimize scheduling by targeting available time gaps in cloud systems.

Reactive, proactive, and resilient fault-tolerance methods are the three main categories used to describe them in the literature. Proactive approaches proactively monitor for mistakes and reroute execution pathways to avoid failures, in contrast to reactive methods that handle faults after they have occurred. Resilient approaches go further by incorporating intelligent learning mechanisms to avoid faults proactively. Task resubmission, a reactive strategy, was employed in the article by Yao et al. [23], but repeated reassignments led to inefficient resource usage. Software rejuvenation, involving periodic system restarts, was suggested in the article by Rezaei Kalantari et al. [24] as a proactive measure. Resilient strategies that adaptively monitor and address faults were applied in the article by Shahid et al. [25] and extended in the article by Liu et al. [26] using system response-based assertions. A fault-aware scheduling technique, the checkpointed League Championship Algorithm (CPLCA), was introduced in the article by Liakath et al. [27], enabling task migration and checkpointing, though lacking load-balancing capabilities. In the article by Chen et al. [28], the DBSA heuristic was proposed for job scheduling in heterogeneous environments. The Dynamic Clustering League Championship Algorithm was introduced in the article by Sheeba and Uma Maheswari [29] to reduce premature task failure. Additionally, Mushtaq et al. [30] proposed a reservation method that pre-allocates VMs to tasks for a fixed time window to ensure completion. Key challenges in cloud computing include balancing workloads across VMs and optimizing makespan and resource utilization. Several strategies have been developed to address these issues. HBLBA, a heuristic-based load-balancing algorithm for IaaS clouds, organizes servers based on task size and count for efficient VM selection. A clustering and Bayes theorem-based heuristic model was introduced in the article by Negi et al. [31], though resource utilization remains a limitation. In the article by Mani and Kaur [32], a cloud partitioning method using game theory was proposed for public cloud environments. HEFT was enhanced into E-HEFT and LB-HEFT to incorporate load balancing for improved scheduling. The PTAL algorithm was introduced in the article by Shiekh et al. [33] to enable better resource allocation and QoS, while WAMLB in the article by Singh and Prakash [34] used active monitoring to boost utilization. However, both lacked dynamic adaptability to changing tasks and VM conditions. OLB maintains load equilibrium but struggles with making a span when handling multiple objectives. MELISA, an improvement of ELISA, addresses resource heterogeneity and migration cost, making it suitable for large-scale systems. More recently, LDRA was proposed to optimize resource allocation and reduce task execution time in distributed settings.

In the article by Radojević and Žagar [35], a centralized load-balancing decision model was developed for cloud systems use. This method eliminates the need for human administrators to oversee scheduling by automating the process. However, the model does not effectively address detecting or managing system faults. In unexpected failures, existing fault tolerance strategies often compromise system reliability and reduce resource management efficiency. The inherently dynamic nature of VMs in cloud environments contributes to instability and fault-proneness, potentially leading to interrupted task execution and uneven workload distribution. Even when faults or system anomalies are detected, ensuring the timely delivery of services is essential to avoid premature task termination, which defines the core goal of fault-tolerant cloud systems. Comprehensive surveys have been conducted to evaluate scheduling techniques, load balancing algorithms, and fault tolerance mechanisms, as compiled in the article by Poola et al. [36]. They investigated the integration of machine learning with fault-tolerance strategies in cloud infrastructures. Their study highlighted how predictive models can identify potential system failures in advance and enable proactive fault-handling mechanisms. Kalaskar and Thangam [37] examined different fault-tolerant strategies. It was found that these techniques are crucial for optimizing overall cloud infrastructure performance. These insights have inspired continued research into fault-tolerant task scheduling and load-balancing strategies. A model named FTHRM was proposed in the article by Sheikh et al. [38], which introduced the concept of reservation-based fault tolerance. Although the concept of reservation was outlined, it was not implemented in the model. For problems when some nodes are heavily loaded and others are idle or weakly loaded, the Proactive and Reactive Fault Tolerance Framework was introduced in the article by Tamilvizhi and Parvathavarthini [39] and integrated with the Elastic Cloud Balancer. Like PLBFT, which suggests in the article by Attallah et al. [40], proactive fault handling combined with efficient load balancing, there is also PLBFT. To improve availability, Mohmmed and Abdalrahman [41] presented a hybrid strategy integrating load balancing and fault tolerance strategies. Moreover, Shahid et al. [42] introduced a robust fault-tolerant load balancing model called FTLB, which directly includes fault tolerance into the load balancing mechanism, thereby allowing the system to dynamically manage problems throughout the load distribution process.

Although numerous heuristic, RL, and hybrid RL approaches have been proposed for task scheduling and fault tolerance in cloud systems, critical gaps remain unaddressed. Traditional heuristic and rule-based models are simple but fail to adapt effectively under dynamic workloads or unpredictable failures [16, 24, 34, 35]. Single-agent RL methods such as DQL, PPO, and SAC [19], Mnih et al. [43], Schulman et al. [44] and Haarnoja et al. [45] improve adaptability by learning from experience; however, they generally optimize only one performance metric, leading to inefficiencies in balancing throughput, latency, energy, and fault tolerance simultaneously. Hybrid RL approaches attempt to combine the strengths of value-based and policy-based methods, yet most suffer from limited scalability, high computational overhead, or incomplete recovery mechanisms under dynamic failure conditions [12, 14, 23, 37]. These limitations highlight the absence of a comprehensive solution that simultaneously ensures adaptability, efficiency, and resilience in large-scale cloud systems. To bridge this gap, the proposed Hybrid DDQL scheduler integrates value-based and policy-based learning to provide dynamic fault tolerance, robust multi-objective optimization, and efficient resource utilization, thereby advancing beyond the capabilities of existing methods.

3. Problem Formulation

This section formulates the scheduling problem addressed in this study, particularly fault-tolerant scheduling in real-time cloud systems. The problem involves dynamically allocating resources and scheduling tasks to ensure high availability, minimize the impact of failures, and maximize overall system performance.

3.1. A representation of the planned system

Take into consideration an N-task cloud system $T = \{t_1, t_2, \ldots, t_N\}$ that needs to be scheduled across M available VMs $V = \{v_1, v_2, \ldots, v_M\}$. Each task has the following attributes:

1) **Arrival time** at: Time at which the task arrives in the system.
2) **Execution time** e_i: Time required to complete the task.
3) **Deadline** d_i: Time by which the task must be completed.
4) **Resource requirements** r_i : A vector representing the resources (e.g., CPU, memory, and bandwidth) required by the task.
5) **Priority** p_i: A numerical value indicating the importance of the task relative to others.

Similarly, each VM v_j has the following attributes:

1) **Processing capacity** c_j: Number of million instructions per second (MIPS) that a VM can execute.
2) **Available resources** r_j: A vector representing the current availability of resources on the VM.
3) **Failure probability** f_j: The likelihood of failure occurring for each VM over a given period.

3.2. Objectives and constraints

Dynamically allocating work to VMs while retaining good system performance and tolerance to failures is the core aim of the fault-tolerant scheduling problem in real-time cloud computing systems. The following key objectives are considered:

1) Maximize FTSR by completing failure-affected tasks through effective recovery and reassignment.
2) Maximize MTBF to improve system reliability by extending the interval between consecutive failures.
3) Minimize MTTR to reduce the duration required for system restoration following a failure.
4) Minimize DMR by reducing the number of tasks that miss their specified deadlines.
5) Minimize EC by decreasing the total power usage of active VMs during execution.

Raise the amount of work done per unit of time to improve system efficiency, which is known as throughput (TH). To ensure that the scheduling algorithm operates effectively and within system limitations, the following constraints are enforced:

1) **Resource constraints:** Each task must be assigned to a VM with sufficient available resources. The total amount of resources allotted must not exceed the VM's limit.

$$R_{ij} \leq R_j^{\text{available}} \tag{1}$$

2) **Task dependency constraints:** If task t_2 depends on the completion of task t_1, the scheduler must ensure that t_1 is completed before t_2 begins.

$$\text{Start}(t_2) \geq \text{End}(t_1) \tag{2}$$

3) **Failure resilience constraints:** If VM v_{j} fails at time t_{f}, the affected task must be reassigned to a functioning VM and recovered within the defined MTTR without violating the task's deadline.
4) **Deadline constraints:** Each task must be executed before its specified deadline.

$$\text{End}(t_i) \leq d_i \tag{3}$$

3.3. Mathematical formulation

Let x_{ij} be a binary decision variable representing the allocation of task t_i to VM vj, where

$$x_{ij} = \begin{cases} 1 & \text{if task } t_i \text{ is assigned to VM } v_j \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

The objective function seeks to optimize a weighted combination of the selected performance metrics. The goal is to maximize FTSR, MTBF, and TH while minimizing MTTR, DMR, and EC. This multi-objective optimization is represented as follows:

$$\begin{aligned} \min(&\lambda_1 \cdot DMR + \lambda_2 \cdot MTTR + \lambda_3 \cdot EC \\ &- \lambda_4 \cdot FTSR - \lambda_5 \cdot MBTF - \lambda_6 \cdot TH) \end{aligned} \tag{5}$$

Where λ_1 to λ_6 coefficients indicate how each measure is weighted over the others according to the application's needs.

1) **Resource capacity constraint:**

$$\textstyle\sum_i x_{ij} \cdot R_{ij} \leq C_j \forall j \tag{6}$$

2) **Processing capacity constraint:**

$$\textstyle\sum_i x_{ij} \cdot PT_{ij} \leq P_j \forall j \tag{7}$$

3) **Failure recovery timing constraint:**

If a failure occurs at time t_{f}, recovery and reassignment must be completed by the following:

$$t_{\text{f}} + MTTR \leq d_i \tag{8}$$

In the event of failure, the task reassignment must occur before the deadline. If failure occurs at time t_{f}, task reassignment occurs within MTTR and ends by d_i.

In the context of this problem, a Hybrid DDQL framework is employed to solve the fault-tolerant scheduling problem. The agent learns the following:

1) Dynamically allocate tasks to VMs based on real-time system feedback.
2) React to system failures by adjusting task allocations while ensuring the objectives are met.
3) Optimize system functioning regarding EC, fault tolerance, and scheduling efficiency.

For decisions grounded in values, the hybrid DDQL model uses DDQL; for decisions grounded in policies, it combines PPO with SAC. This combination ensures that the scheduler can react to dynamic cloud settings with different workloads and failure events, therefore addressing the complicated, high-dimensional character of the problem. The cloud computing solutions offered in real-time make it possible to solve the problem of scheduling by applying a systematic approach with the help of hybrid RL methods. This issue formulation is actually the heart of dynamic fault-tolerant scheduling.

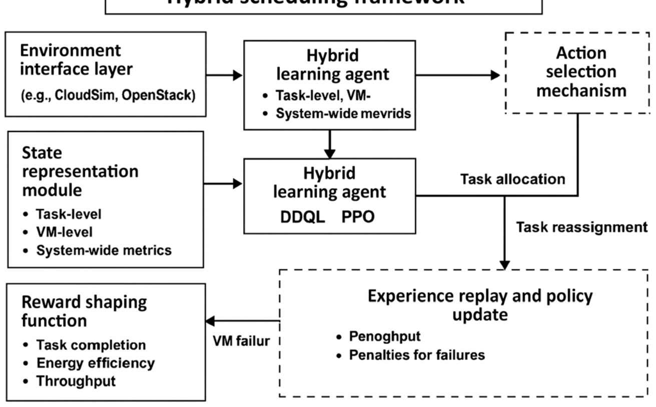
4. Proposed Methodology

This part details the suggested hybridization mechanism of scheduling that integrates value-based as well as policy-oriented RL to enhance resilience to faults, flexibility, and efficiency of dynamic cloud systems. The architecture of the framework, learning elements, and allocation strategies of resources are presented to illustrate how the model comes up with resilient and smart scheduling.

4.1. Overview of the hybrid scheduling framework

The proposed hybrid scheduling system is used to address the complex, dynamic, and fault-prone nature of the real-time cloud computing systems. Integrating value-based and policy-based learning with a hybrid reinforcement learning strategy is aimed at ensuring effective resource allocation and task scheduling. This will assist the system to maintain operational resiliency during times of failure, explore and exploit, and accommodate the changing conditions.

Figure 1
Review of the suggested model



Note: DDQL = Double Deep Q-Learning, PPO = Proximal Policy Optimization, VM = virtual machine.

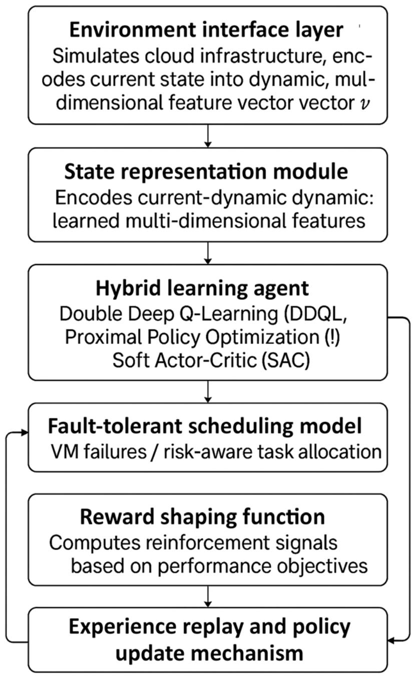
4.1.1. Architecture of the proposed system

Several necessary modules included in the proposed hybrid scheduling architecture run in concert to raise the fault tolerance and performance of real-time cloud computing systems. Figure 1 gives a general picture of the proposed model. Fundamentally, the Environment Interface Layer—which uses OpenStack or CloudSim to replicate the cloud infrastructure—is. By always giving the agent real-time data on job arrivals and completions, resource availability and consumption, system failures, recovery events, and critical performance indicators, including throughput, EC, MTTR, and MTBF, this layer offers the operational context.

Using a dynamic, multidimensional feature vector, the State Representation Module catches the cloud system's state. Task-level characteristics such as arrival time, execution length, deadline, priority, and resource needs comprise this vector. It also combines system-wide parameters, including total workload, energy use, continuous failure events, and recovery progress, with VM-level characteristics such as available processing capability and failure probability. By use of this complete state encoding, the scheduling agent may make educated judgments depending on the present situation of the system. The intelligent scheduling decisions are pushed by the hybrid learning agent in the centre of the system, which combines multiple methods of RL. It particularly applies the SAC to determine the policy robustness through entropy-regularized exploration, PPO to optimize a stable policy, but making use of clipped surrogate goals, and DDQL in determining the precise estimate of value of the task-to-VM assignments. These components work in tandem to have the agent realise the long-term value and optimum ways of job allocation in dynamic and possibly flawed environments. The Action Selection Mechanism arranges the decision-making process in each scheduling cycle. It takes policy distributions generated by PPO and SAC and evaluates Q-values generated by DDQL. The final scheduling step is determined based on a weighted ensemble decision method in which the weight of each method is dynamically varied based on environmental variables and more recent performance indicators. This allows the dynamically balanced exploration and exploitation of the framework to adapt to changes in the stability of the system and changes in the cloud workload. Based on failure probability estimation to ensure resilience, the fault-tolerant scheduling module detects failures of VMs and predicts potential hazards. It is reallocating failed or high-risk node jobs to healthy VMs within the permitted recovery time (MTTR) and the high-priority tasks in aggressive migration or replicas to maintain the target FTSR. This module is important in maintaining the operation of the system in the case of partial outages. The flowchart of the proposed model is presented in Figure 2.

A Reward Shaping Function guides the agent's learning, which defines the reinforcement signals based on multiple performance objectives. Positive rewards are assigned for achieving timely task completions, efficient energy use, and high throughput, whereas penalties are imposed for missing deadlines, excessive response times, or failed recovery attempts. This reward design enables the agent to learn strategies that balance reliability, efficiency, and performance. Lastly, the Experience Replay and Policy Update Mechanism facilitates continuous learning by storing past interactions in a replay buffer. Prioritized experience replay is employed to update the DDQL networks more effectively, whereas policy/value updates for PPO and SAC are performed through batch gradient ascent and descent. Target networks are updated periodically to stabilize training and mitigate oscillations in learning. This iterative refinement process enables the agent to evolve its scheduling strategies over time, achieving ideal performance in dynamic cloud environments. The pseudocode of the hybrid scheduling frame is given in Table 1.

Figure 2
Diagram depicting the proposed model's flow



Environment interface layer
Simulates cloud infrastructure, encodes current state into dynamic, multi-dimensional feature vector vector v
State representation module
Encodes current-dynamic dynamic: learned multi-dimensional features
Hybrid learning agent
Double Deep Q-Learning (DDQL, Proximal Policy Optimization (!) Soft Actor-Critic (SAC)
Fault-tolerant scheduling model
VM failures / risk-aware task allocation
Reward shaping function
Computes reinforcement signals based on performance objectives
Experience replay and policy update mechanism

Table 1
Pseudocode: hybrid scheduling framework

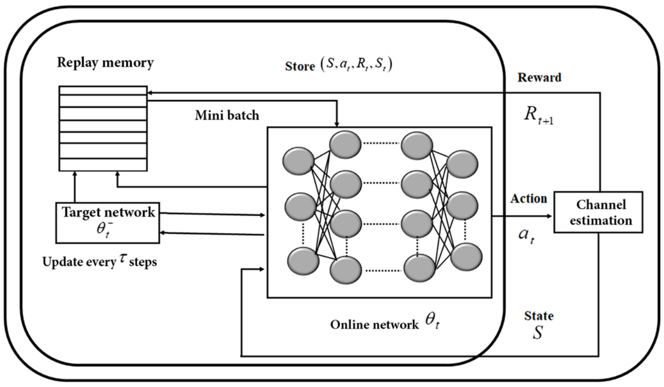
```
Initialize environment and agents (DDQL, PPO, SAC)
Initialize replay buffers
Loop while the system is running:
    Get the current system state
    // Select action using a hybrid agent
    Use DDQL to estimate Q-values
    Use PPO to get policy probabilities
    Use SAC to sample entropy-regularized action
    Combine outputs from DDQL, PPO, and SAC
    Select the best task-VM assignment
    // Check for VM failures
    If VM is at high risk of failure:
        Reassign or migrate task to healthy VM
    Execute selected action (schedule task to VM)
    Observe new state and compute reward:
        - Reward for on-time task completion, low energy, high
Throughput
        - Penalty for delays, deadline misses, failures
    Store experience in replay buffers
    // Periodically update agent models
    If it is time to update:
        Update DDQL Q-network
        Update PPO policy
        Update SAC actor-critic models
End Loop
```

Note: DDQL = Double Deep Q-Learning, PPO = Proximal Policy Optimization, SAC = Soft Actor-Critic, VM = virtual machine.

4.2. DDQL model

Figure 3 shows the results of DDQL. This strong RL method aims to fix the problems with regular DQNs, namely their tendency to overestimate Q-values. In DQNs, the Q-value update relies on selecting and evaluating the best action using the same network, which can lead to an optimistic value bias. This overestimation significantly affects performance in highly dynamic or partially observable environments, such as real-time systems with task deadlines, uncertain resource availability, and potential node failures.

Figure 3
Structure of DDQL



Replay memory
Store (S, a_t, R_t, S_t)
Reward
R_{t+1}
Mini batch
Action
Channel estimation
a_t
Target network θ_t^-
Update every τ steps
Online network θ_t
State S

To address this, DDQL decouples the action selection and action evaluation processes by utilizing the following two separate neural networks:

1) **Primary Q-network** Q_{primary}(s, a;θ): Used to select action with the highest estimated value.
2) **Target Q-network** Q_{target}(s, a;θ−): Used to compute the target Q-value for learning updates.

The key innovation of DDQL lies in the way it calculates the following target Q-value:

$$Q_{\text{target}} = r + \gamma \cdot Q_{\text{target}}\left(s', \arg\max_{a} Q_{\text{primary}}(s', a; \theta); \theta^{-}\right) \tag{9}$$

Where:

1) r: Immediate reward received after taking action 'a' in state 's'
2) 's'': Next state
3) γ: Discount factor representing the importance of future rewards
4) θ: Parameters of the primary network
5) θ^{-}: Parameters of the target network (updated periodically from θ)

Decoupling the networks in this way guarantees that the assessment of the chosen action is more stable and less prone to bias, thereby producing more accurate Q-value estimations.

DDQL acts as an intelligent, learning-based scheduler in real-time cloud settings when timely execution and fault resilience are vital. Unlike conventional rule-based methods, the DDQL agent learns appropriate scheduling rules dynamically depending on system changes via constant interaction with the surroundings. At every decision point, the agent notes the present system state, which consists of several elements, including task-specific criteria, including deadlines, projected execution durations, and priority levels. Besides the nature of the job, the agent monitors the health of the available VMs in the areas of historical reliability, CPU load, memory utilization, and availability. Such realizations will inform the decision of a VM to run specific activities at current load levels.

In addition, real-time network factors such as latency and bandwidth available are also considered, which influence the overall delays in communication and performance. Of paramount importance in causing the agent to make more regular scheduling decisions are fault-related indicators such as past failure rates and predicted fault probability. All the potential scheduling options are evaluated according to the anticipated long-term utility. Minimizing the number of deadline misses, minimizing energy use, preventing problems with task execution, and maintaining a balanced load balance across VMs are the key objectives of the system that the given reward signal is particularly intended to represent. DDQL assists in developing robust scheduling strategies that enhance the performance and resilience of the system in the long-term by incorporating the following several and sometimes conflicting goals into the process of learning.

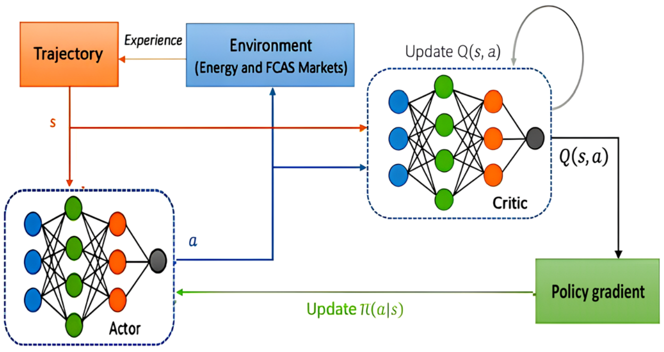
The DDQL-based scheduler undergoes the following iterative learning process:

1) **State encoding:** The current system status is converted into a high-dimensional state vector representing all relevant metrics.
2) **Action selection:** An action (task-to-VM assignment) is selected using an **ε-greedy policy**, where:
   a. With probability 'ϵ', a random action is chosen (exploration).
   b. Probability 1−ϵ1 selects the action with the highest estimated Q-value (exploitation).
3) **Environment transition:** The selected task is scheduled to the chosen VM, resulting in a new state and a reward that reflects the success or failure of that decision (e.g., timely execution, failure recovery, and resource utilization efficiency).
4) **Experience replay:** The tuple (s, a, r, s′) is stored in a replay buffer, and mini-batches are randomly sampled for training, breaking the correlation between sequential updates and improving data efficiency.
5) **Network update:** The primary network is updated to minimize the loss between predicted and target Q-values. Periodically, the target network is synchronized with the primary network to stabilize learning.
6) **Policy improvement:** Over time, the DDQL agent learns to prioritize scheduling actions that yield long-term system benefits, achieving adaptive and resilient scheduling behaviour even in task bursts, resource contention, or hardware failures.

4.3. PPO integration

Figure 4 shows the state-of-the-art policy gradient approach in RL called PPO, which strikes a compromise between stable and reliable learning requirements and the necessity for effective learning. In contrast to value-based approaches, such as Q-learning, that seek to

Figure 4
Architecture of proximal policy optimization



determine the best values for actions, PPO is a policy-based method that optimizes the policy that links states to actions. Its integration into real-time scheduling systems enhances the ability of the learning agent to make nuanced, stable decisions in highly dynamic environments.

Policy gradient approaches aim to determine the best policy $\pi\theta$(a|s), where θ is a parameter that maximizes the predicted cumulative reward. Generally, one should move the policy parameters toward the expected return's gradient while updating the policy. However, altering the policy in big chunks might result in disastrous policy deterioration if you are not careful. A clipped surrogate goal, introduced by PPO, limits the policy update size, making learning more stable and preventing this issue.

The clipped objective function in PPO is as follows:

$$L^{CLIP}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\grave{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\grave{A}_t\right)\right]. \quad (10)$$

Where:

1) $r_t(\theta) = \frac{\pi_t(a_2|s_t)}{\pi_{\theta_d}(a_t(\beta_t)}$ is the probability ratio.
2) $\grave{A}_t$ is the estimated advantage function.
3) ϵ is a hyperparameter (typically 0.1 to 0.3) that controls the extent of clipping.
4) Et denotes the expectation over timesteps (*t*) during sampled trajectories in RL.

This goal keeps the new policy from straying too far from the previous one by lowering the chance ratio. The outcome is a more cautious but steady rise in policy performance. In real-time systems, PPO is a policy optimizer that improves the scheduling agent's ability to choose actions. PPO uses stochastic policies, which let the agent respond more flexibly to changing system conditions. This is different from deterministic policies, which may not work as well in uncertain or random situations. This flexibility is helpful when the rate at which tasks come in changes, the way resources are used changes, or the system fails in an unexpected way. The agent's decision-making process is guided by a state vector that contains all the knowledge about the system. This vector usually has task-related parameters like execution time, deadline, and priority. It also includes important information from VMs, such as their availability, current workload, and past reliability. Also considered are outside factors such as network slowness and the chance of failure, which gives a complete picture of how the system is working. PPO uses this input to learn the probability distribution of possible actions, where each action represents a possible task-to-VM assignment. The agent does not choose the most likely action deterministically; instead, it samples actions from this learned distribution. This random quality makes people want to look into alternative options, which stops local optima and makes the system more resilient. PPO also ensures that policy changes are consistent and gradual by using a clipped surrogate objective. This stops scheduling behavior changes from becoming unstable.

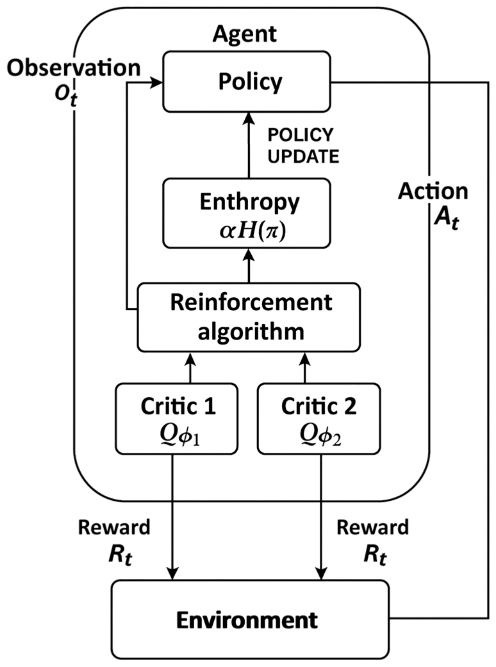
4.4. SAC module

SAC is a new RL algorithm based on the maximum entropy framework. It promotes both high reward and high entropy (exploration). This makes SAC especially useful in changing and unpredictable places, such as real-time cloud systems, where job scheduling and fault tolerance need to be flexible and strong.

The SAC architecture shown in Figure 5 comprises the following three neural networks:

1) **Actor-network:** Produces a stochastic policy by outputting a probability distribution over actions for a given state.
2) **Critic networks (Q1 and Q2):** Estimate Q-values for given state-action pairs. Having two critics reduces overestimation bias.

Figure 5
Soft actor-critic architecture



3) **Temperature parameter (*α*):** Governs the compromise between discovery (entropy) and profit maximization (exploitation).

The core objective of SAC is to maximize the following entropy-augmented expected return:

$$J(\pi) = \sum_t \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t,a_t) + a\,\mathcal{H}\,(\pi(\cdot|s_t))] \quad (11)$$

Where:

1) $\mathcal{H}(\pi(\cdot|s_t))$ stays the entropy of the policy at state s_t
2) α adjusts the importance of the entropy term .

SAC helps the system keep different decision paths open in real-time scheduling. This lowers the chance of deterministic policies locking into bad strategies. It is very good for situations where workload patterns and failure conditions change, because it can handle continuous action spaces and keep exploring even after convergence.

4.5. Hybrid DDQL scheduler design

The Hybrid DDQL Scheduler is the main part of the proposed smart real-time scheduling system that makes decisions. It combines three cutting-edge RL algorithms—DDQL, PPO, and SAC—into a single framework to get around the problems with each of them on their own. Every method adds its own benefits that help the system make decisions that are fault-tolerant, adaptable, and reliable in the very dynamic and unpredictable circumstances that are common in cloud computing.

The hybrid DDQL scheduler has three complementary RL modules. Each module has a different but nonetheless important role in making real-time task scheduling better. The DDQL module is in charge of value-based learning. It also uses a dual-network design to guess Q-values for state-action pairs. This means that there is an online network and a target network. This distinction leads to more consistent convergence by lowering the bias that makes people overestimate what they learn in regular Q-learning. Its power lies in optimizing cumulative incentives to learn optimal task-to-VM allocations. On the other hand, the PPO module uses a policy gradient method to make stochastic policies that work better in situations with a lot of noise or parameters that change quickly. It uses a clipped surrogate goal function to make sure that policy updates stay within a stable range, which keeps training from becoming random. This module is especially useful for scheduling problems where the system state is only partially visible or not predictable.

The SAC module, on the other hand, encourages a deeper investigation of the action space, which makes the system better at making decisions. Entropy regularization in SAC makes sure that the agent does not commit to any strategy too soon in the learning process. This makes people more flexible and adaptable, which is especially useful in places where things can go wrong or get too busy all of a sudden. SAC makes the scheduler better able to deal with uncertainty and changing system needs by letting it choose from a wide range of actions.

State acquisition: The system collects comprehensive real-time data, including task specifications (e.g., deadline, length, and criticality), VM states (e.g., CPU utilization, memory availability, and reliability score), and environmental metrics (e.g., bandwidth, latency, and failure rate).

1) **Individual policy inference:**
   a. The DDQL module evaluates the Q-values and selects the action with the highest expected return.
   b. The PPO module computes a probability distribution over the action space and samples a policy-compliant action.
   c. The SAC module produces an action by balancing the reward expectation with entropy to maintain exploratory flexibility.
2) **Weighted ensemble decision mechanism:**
   a. A fusion function integrates the three modules' output actions or action values.
   b. Weights are not static; instead, they are dynamically adjusted based on **real-time scheduling context**, such as the following:
      - Historical success rates (reward trend history)
      - System-level performance metrics (Throughput, task failure rates)
      - Task-specific urgency (approaching deadlines or high priority)
      - Detected fault likelihood (predicted or historical failures on VMs)
3) **Action execution and feedback:**
   a. The selected action (i.e., task-to-VM assignment) is executed.
   b. The environment responds with a reward and updated system state.
   c. All three modules update their internal models using the experience, enabling continued learning.

Combining these learning techniques guarantees that no one model dominates decision-making across all scenarios in the hybrid DDQL scheduler. Rather, it constantly responds to changing system circumstances and performance requirements, providing a robust, fault-tolerant, and performance-efficient scheduling mechanism for cloud-based real-time contexts.

4.5.1. Weighted ensemble decision mechanism

The outputs of DDQL, PPO, and SAC are integrated through a weighted ensemble decision mechanism. The weights are not static; instead, they are dynamically updated based on real-time scheduling performance. Let i∈{D,P,S} denote DDQL, PPO, and SAC, respectively. At each scheduling cycle *t*, each algorithm produces action-value estimates Qi(st,a) or policy distributions.

1) **Performance scoring**

For each algorithm, performance indicators such as throughput (TH), DMR, EC, and FTSR are monitored over a sliding window. A normalized performance score is computed as follows:

$$P_i(t) = \sum_{k\in\{TH,FTSR,EC,DMR\}} \alpha_k \cdot \widetilde{m}_{i,k}(t) \tag{12}$$

Where $\widetilde{m}_{i,k}(t)$ is the normalized metric value and α_k are predefined importance weights (∑αk = 1).

2) **Weight calculation**

Dynamic weights are obtained using a softmax function with sensitivity parameter β.

$$W_i(t) = \frac{\exp(\beta\cdot P_i(t))}{\sum_{j\in\{D,P,S\}} \exp(\beta\cdot P_j(t))} \tag{13}$$

3) **Ensemble decision**

The final decision is derived from the weighted Q-value aggregation:

$$Q_{\text{ens}}(s_t,a) = \sum_{i\in\{D,P,S\}} W_i(t)\cdot Q_i(s_t,a) \tag{14}$$

The action is then chosen as follows:

$$a_t = \arg\max_a Q_{\text{ens}}(s_t,a). \tag{15}$$

If at time *t*, the normalized scores yield P_D = 0.75, P_P = 0.55, P_S = 0.60, and β = 10, the resulting weights are W_D = 0.70, W_P = 0.08, and W_S = 0.22. In this case, DDQL contributes most to the final scheduling action due to its higher performance score. This mechanism ensures that the scheduler adapts weights according to the current system context, favouring the algorithm most effective under prevailing workload and fault conditions.

4.6. Planning and allocation of resources

The proposed hybrid RL system guarantees intelligent, dynamic, and fault-tolerant work management in real-time cloud settings through its scheduling and resource allocation techniques. The approach aims to achieve multiple system objectives concurrently: reducing task deadline violations, optimizing resource utilization, decreasing EC, and enhancing the overall FTSR. At the start of each scheduling cycle, the agent talks to the cloud environment to find a high-dimensional status vector that shows important system information. This includes things such as task-specific features, such as deadline, execution time, and priority level; VM-related metrics like CPU load, memory availability, and historical dependability; and environmental factors such as network delay, energy utilization, and expected failure rate. The hybrid scheduling agent looks at all the alternative task-to-VM allocations based on this composite state and chooses the best one using a weighted combination of DDQL, PPO, and SAC decision outputs. The basis of the method is to find a balance between different system goals by making trade-offs. For a high-priority task with a short deadline, a lightly loaded VM with a high-reliability score may be allocated, even if it uses more energy, because timeliness and availability are more important than energy economy. To save electricity, lower-priority tasks could be put off or planned on nodes that use less energy. This prioritization is possible because the agent's learned reward function takes into consideration numerous goals.

R = λ_1 Throughput −λ_2 DeadlineMissRate −λ_3 EnergyConsumption −λ_4·FaultPenalty may be written generally as follows: System-tuneable weights that specify the value of every objective define λ1, λ2, λ3, and λ4. The approach also includes fault condition adaptive scheduling. Based on real-time fault prediction data, the agent proactively moves or replicates important activities to healthier VMs when it sees the probability of a VM failure. To guarantee that rescheduling activities may still be finished without SLA breaches, the agent values MTTR and task deadlines. Furthermore, the scheduler adjusts its policies to improve next judgments by always learning from prior events. In this sense, resource allocation is not fixed; it changes based on feedback. The system modifies the VM state, task queue dynamics, and energy profiles as work is done. The learning model may change how it schedules things in real time based on these modifications. This cycle makes sure that the system is always aware of changes in workload and is powerful enough to handle overloads or faults. In the end, the suggested approach makes it possible to manage resources effectively and securely, which lets the cloud system meet rigorous real-time needs while also maximizing long-term operating efficiency. The hybrid learning model is a good fit for modern distributed computing environments that need high availability and smart automation because it can combine fault prediction, dynamic prioritization, and multi-objective optimization.

5. System Architecture and Workflow

This section covers the proposed hybrid DDQL-based fault-tolerant scheduler for real-time cloud computing systems, along with its methodology and architecture. By combining learning models with monitoring, scheduling, and fault recovery aspects, the architecture makes sure that the system can respond to faults and changing workloads in real time.

5.1. Cloud environment configuration

The system works in a simulated real-time cloud computing environment that is meant to seem like the way real cloud infrastructures work, with all their changes and unknowns. The key parameters and component specifications of this simulated cloud setup are summarized in Table 2.

There is a pool of VMs in this simulated environment, each with its own set of capabilities and behaviours. There is also a steady stream of different jobs that need to be done quickly with limited resources. Each VM has a certain amount of processing power, measured in MIPS, which is a measure of how powerful the VM is. Also, VMs have variable amounts of resources available to them, like CPU cores, memory (RAM), and bandwidth. These resources affect how well they can host and perform different kinds of operations. Additionally, the failure probability of each VM is incorporated to replicate real-world scenarios where hardware or software malfunctions can impede job performance. This probabilistic failure model is very important for figuring out how well the system can handle faults. Tasks that come into the system are jobs that users or apps have sent in and are defined by a number of different parameters. These include the following:

1) **Arrival time (a_i):** The time a task is submitted to the cloud system.
2) **Execution time (e_i)**: The estimated time required to complete the task.
3) **Deadline (d_i):** The latest allowable time for completing the task without violating SLAs.
4) **Priority (p_i):** An importance level assigned to each task, influencing scheduling decisions—especially under overloaded or failure conditions.
5) **Resource requirements (r_i):** The CPU, memory, and bandwidth needed to execute the task effectively. Tasks may be CPU-intensive, memory-bound, or I/O-intensive, and their placement on appropriate VMs is crucial for system performance.

This rich and dynamic environment provides a challenging testbed for evaluating intelligent, learning-based scheduling and fault recovery strategies.

5.2. Real-time scheduling flow

The scheduling process is orchestrated in three key phases: An overview of these workflow stages, their corresponding components, and functional roles is summarized in Table 3.

Table 2
Cloud environment configuration

Component	Parameter	Description
Virtual machines	Processing capacity (MIPS)	Defines the computational speed of a VM (e.g., 1000 to 3000 MIPS), which affects how quickly it can process tasks.
	Available resources	Specifies the VM's available CPU cores, memory (e.g., 2GB–8GB), and bandwidth (e.g., 100–1000 Mbps).
	Failure probability	Represents the likelihood of a VM encountering a failure during execution, typically modelled between 0.01 and 0.1.
Tasks	Arrival time (a_i)	The timestamp when a task enters the cloud system for scheduling.
	Execution time (e_i)	The estimated time required to complete the task on a given VM.
	Deadline (d_i)	The task must be completed by the cutoff time to avoid SLA violations.
	Priority (p_i)	A numerical value (1 to 5) indicates the importance of the task, influencing its scheduling order.
	Resource requirements (r_i)	Specifies the task's demand for CPU cycles, RAM, and network bandwidth.

Note: MIPS = Million Instructions Per Second, SLA = service-level agreement, VM = virtual machine.

Table 3
Workflow stages and components

Stage	Component	Description
1. State observation	Monitoring module	Captures current VM/task status and failures
2. Decision making	Hybrid DDQL scheduler	Integrates DDQL + PPO + SAC for action selection
3. Execution	Task executor	Allocates tasks to selected VMs and initiates execution
4. Feedback and learning	RL agent trainer	Updates the RL model based on rewards and outcomes

Note: DDQL = Double Deep Q-Learning, RL = reinforcement learning, SAC = Soft Actor-Critic, SLA = service-level agreement.

1) **Task monitoring and state observation:**

The system monitors incoming tasks and VM states (resource utilization, queue lengths, failures, etc.). The state space is updated in real time and fed into the Hybrid DDQL agent.

2) **Action selection and resource allocation:**

Based on the observed state, the agent selects actions (e.g., VM-task mappings) using the hybrid RL strategy. The value-based DDQL helps estimate the long-term reward of actions, whereas the PPO and SAC components ensure policy robustness and stability.

3) **Execution and feedback update:**

Once a decision is made, the selected VM executes the task. The scheduler receives feedback on execution success, time, EC, and fault status. This feedback updates the learning model to improve future scheduling decisions.

5.3. Fault detection and recovery process

The system includes a proactive fault management layer. It detects hardware/software failures using log analysis, heartbeat signals, and threshold-based monitoring. The major components and their respective roles in the fault-handling process are summarized in Table 4. Upon detection, tasks assigned to the failed VM are reassigned to another available VM within the defined MTTR using the trained policy network.

6. Experimental Setup

This section outlines the simulation platform, datasets, performance metrics, and comparative models used to evaluate the effectiveness of the proposed hybrid scheduling approach.

Table 4
Fault handling components

Component	Function
Fault detector	Identifies anomalies or missed heartbeats
Fault responder	Triggers task reassignment within MTTR
Reinforcement agent	Selects the optimal VM for the reassigned task
Recovery validator	Confirms successful task migration and resumption

Note: MTTR = mean time to recovery, VM = virtual machine.

6.1. Dataset and simulation tools

The experimental setup was deployed within a controlled, simulated cloud environment to validate the proposed fault-tolerant scheduling mechanism. The evaluation leverages benchmark workloads from real-world cloud data centre traces to ensure realistic and representative task patterns. The simulation framework is designed to replicate the dynamics of actual cloud computing environments, including resource heterogeneity, task unpredictability, and fault occurrences.

The complete simulation environment configuration, including tools, platforms, and datasets used for modelling and validation, is summarized in Table 5. The simulation environment integrates multiple platforms and tools, each tailored to a specific layer of cloud infrastructure modelling and learning-based optimization: A widely used discrete event simulation toolkit that allows researchers to model cloud data centres, VM provisioning, scheduling algorithms, and resource allocation policies. CloudSim simulates various IaaS-level components, including hosts, VMs, and tasks (cloudlets). It provides detailed control over system parameters such as bandwidth, memory, processing power (MIPS), and task workloads. The configuration doesn't employ a live deployment of OpenStack, but it does use a simulated middleware layer to emulate OpenStack-like features like creating and moving VMs dynamically and injecting faults. This abstraction enables the assessment of dynamic resource adaptation in reaction to VM failures or resource saturation, together with fault recovery mechanisms. To make intelligent scheduling and fault tolerance possible, we apply deep reinforcement learning (DRL) methods as DDQL, PPO, and SAC. TensorFlow and PyTorch are two Python-based RL programs that speed up training by using GPUs and let you change hyperparameters easily. These are the tools that are used to build and train these models. In trace-driven simulations that try to show how tasks really behave, workloads from the Google Cluster and PlanetLab datasets are used. These databases have information about task arrival rates, resource use, task length, and failure records. This makes it possible to model workloads that are different, change over time, and come in bursts. The infrastructure setup shows a range of VM types, such as small, medium, and large instances. These VMs have varying CPU cores (1–8), memory sizes (512MB–8GB), and processing speeds (100–1000 MIPS). This variety helps you test load balancing, resource-aware scheduling, and DRL agent adaptation across several VM setups.

6.2. Performance metrics

When evaluating the suggested Hybrid DDQL-based fault-tolerant scheduling method in real-time cloud settings, significant performance metrics considered were system resilience, scheduling

Table 5
Simulation environment setup

Tool/platform	Purpose	Configuration details
CloudSim	Cloud infrastructure simulation	Simulates data centres, hosts, VMs, task scheduling, energy consumption, and fault modelling
OpenStack (simulated)	Middleware abstraction fault modelling	Emulates OpenStack APIs for VM creation, deletion, dynamic scaling, and controlled fault injection
RL libraries	DRL agent implementation in Python	TensorFlow 2.x and PyTorch were used to build and train DDQL, PPO, and SAC models
Dataset	Task behaviour modelling	Uses Google Cluster and PlanetLab workload traces (task duration, arrival rate, failure logs)
Environment specs	Infrastructure diversity modelling	VM types: Small (1 core, 512MB, 100 MIPS), Medium (4 cores, 4GB, 500 MIPS), Large (8 cores, 8GB, 1000 MIPS)

Note: DQL = Double Deep Q-Learning, DRL = deep reinforcement learning, MIPS = Million Instructions Per Second, PPO = Proximal Policy Optimization, SAC = Soft Actor-Critic, VM = virtual machine.

efficiency, and energy awareness. The FTSR tells you how well the system can recover from mistakes and remain working without any problems with services. To find it, divide the total number of affected tasks by the number of jobs that were successfully finished once the error was fixed. A higher FTSR means that you are better at handling problems.

MTBF quantifies the system's reliability by measuring the average duration between two consecutive failures. It is defined as follows:

$$MTBF = \frac{\text{Total operational time}}{\text{Number of failures}} \tag{16}$$

A longer MTBF implies fewer interruptions and a more stable cloud infrastructure.

Complementing MTBF, MTTR indicates how quickly the system recovers from faults. It is calculated using the following:

$$MTTR = \frac{\text{Total downtime}}{\text{Number of failures}} \tag{17}$$

Lower MTTR values reflect faster recovery, which is essential for real-time applications requiring minimal service delays.

The DMR evaluates scheduling efficiency by measuring the proportion of tasks that fail to be completed within their deadline constraints. It is given by the following:

$$DMR = \frac{\text{Number of missed deadlines}}{\text{Total number of tasks}} \times 100\% \tag{18}$$

A lower DMR signifies better real-time responsiveness and task management. To emphasize the scheduler's energy efficiency, we also looked at how much energy it used to figure out how much power all the VMs used while they were doing their tasks. To keep clouds running, they need to use as little energy as possible. Lastly, throughput shows how many jobs the system can do at once by demonstrating how many jobs were done well in a certain amount of time. Higher throughput statistics mean that tasks are being handled better and resources are being used as efficiently as feasible. These performance indicators together show that the proposed hybrid RL strategy has advantages over traditional fault-tolerant scheduling methods by showing improvements in system resilience, scheduling accuracy, and operational sustainability.

6.3. Comparative model and baseline

Using many well-recognized baseline and RL-based scheduling models in real-time cloud computing, a comparative study was done to validate the performance of the proposed hybrid value-policy-based DDQL scheduler. The Static Priority-Based Scheduling (SPS) method is the initial baseline model as it provides fixed priorities to activities and arranges them depending on stationary criteria. SPS is computationally efficient and simple, but it cannot manage errors properly and lacks the flexibility to fit changing workloads. Based on their deadlines, the second reference model—the Earliest Deadline First (EDF) scheduling method dynamically organizes jobs [46]. EDF lacks fault-tolerant techniques and fails under overload scenarios, even if it provides superior

real-time responsiveness than stationary models. Value-based RL was represented using the Deep Q-Learning (DQL) algorithm with respect to learning-based baselines [47]. DQL selects behaviors that maximize long-term rewards after estimating Q-values using a neural network. DQL suffers in high-dimensional continuous action contexts common to cloud systems, even if it is efficient in discrete environments. Included as an additional baseline was the state-of-the-art policy gradient technique PPO. PPO presents trimmed substitute goals for consistent policy updates under stability. On the other hand, employed alone, it might converge to poor rules in fast-changing or fault-prone settings. Selected for its entropy-based regularization—which balances exploration and exploitation in continuous control tasks—the SAC model. Although SAC has been demonstrated to function effectively under uncertainty, it requires careful hyperparameter tweaking and large computing resources. The proposed hybrid DDQL Scheduler provides a robust scheduling agent capable of fault recovery, deadline management, and adaptive decision-making by means of value-based DQL coupled with policy-based PPO and SAC. This hybridization helps the system to balance the stability and generality of policy optimization with quick learning from value estimation, thereby outperforming all individual baselines for response time, fault recovery, throughput, and energy economy.

7. Results and Discussion

The complete findings of the hybrid DDQL-based fault-tolerant scheduler in real-time cloud computing systems are presented in this part, together with a thorough analysis of the system's performance over several important criteria. The proposed method is compared to a number of baseline methods based on fault tolerance, recovery time, throughput, energy efficiency, and job completion efficiency.

7.1. FTSR

One of the most important ways to judge how well fault recovery methods work in cloud computing systems is by looking at the FTSR. It tells you how many tasks were finished following a fault recovery event. This shows how well the system can manage failures and get back to work.

As expected, the Hybrid DDQL-based scheduler works better than standard baseline methods such as SPS and EDF scheduling. Both SPS and EDF have set ways to manage tasks, but they do not have any built-in ways to recover from faults, which makes them especially vulnerable to problems caused by VM failures. This leads to low FTSR readings, especially when there are more faults. The Hybrid DDQL scheduler, on the other hand, can change how it works when anything goes wrong because it uses RL algorithms like DDQL, PPO, and SAC. The learning model continuously optimizes the task-to-VM mappings, reassigning tasks promptly to available VMs with the least penalty to completion time. This dynamic, adaptive behavior allows the Hybrid DDQL scheduler to achieve consistently high FTSR values, demonstrating robust fault tolerance.

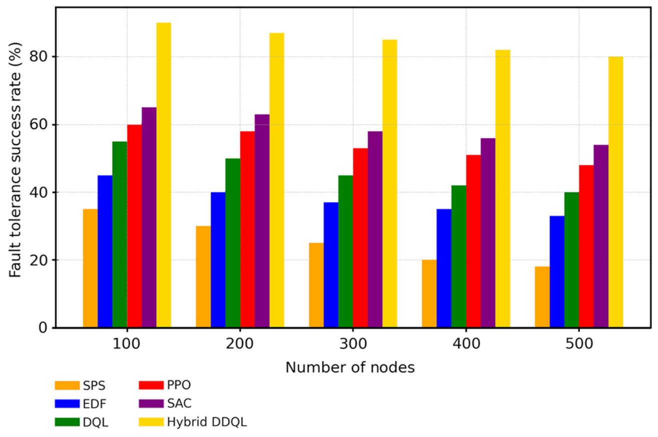
Table 6 and Figure 6 show that the Hybrid DDQL scheduler exhibits significantly higher FTSR values than all baseline methods,

Table 6
FTSR comparison

Number of Nodes	SPS (%)	EDF (%)	DQL (%)	PPO (%)	SAC (%)	Hybrid DDQL (%)
100	35	45	55	60	65	**90**
200	30	40	50	58	63	**87**
300	25	37	45	53	58	**85**
400	20	35	42	51	56	**82**
500	18	33	40	48	54	**80**

Note: DDQL = Double Deep Q-Learning, DQL = Deep Q-Learning, EDF = Earliest Deadline First, FTSR = fault tolerance success rate, PPO = Proximal Policy Optimization, SAC = Soft Actor-Critic, SPS = Static Priority-Based Scheduling.

Figure 6
FTSR comparison



Note: DDQL = Double Deep Q-Learning, DQL = Deep Q-Learning, EDF = Earliest Deadline First, FTSR = fault tolerance success rate, PPO = Proximal Policy Optimization, SAC = Soft Actor-Critic, SPS = Static Priority-Based Scheduling.

with steady performance across different node counts. The FTSR drops with increased nodes but remains consistently higher than the other models, confirming its superior fault tolerance.

7.2. Failure recovery time and MTTR

The MTTR is a key performance metric that evaluates how quickly the system can recover from a failure event and resume task execution. A lower MTTR indicates a faster recovery, which is crucial for maintaining system performance and minimizing task delays in real-time cloud computing environments. The baseline models, including SPS and EDF, suffer from slow fault recovery times due to their lack of adaptive failure handling. For instance, in both SPS and EDF, tasks are either dropped or queued for manual reassignment, resulting in prolonged recovery times. On the other hand, the Hybrid DDQL scheduler, leveraging RL, not only detects faults but also determines the most suitable VM for task reassignment, drastically reducing recovery time. The hybrid nature of the learning approach enhances the system's ability to recover quickly. DDQL provides effective value-based decision-making, while PPO and SAC contribute to faster convergence by handling the exploration-exploitation trade-off. This synergy allows the system to minimize downtime between task failure and reassignment.

As shown in Table 7 and Figure 7, the Hybrid DDQL scheduler demonstrates a remarkable reduction in MTTR compared to all baseline models. The recovery times are significantly lower, highlighting the scheduler's fault detection and recovery efficiency. As the number of nodes increases, the MTTR for Hybrid DDQL increases slightly, but it remains far superior to other methods.

7.3. MTBF

MTBF measures the average time between two consecutive failures in the system. A higher MTBF indicates a more reliable system with fewer failures.

Compared to the baseline models, the Hybrid DDQL scheduler excels in extending the MTBF. The scheduler's ability to proactively handle failures and redistribute tasks ensures that VM failures are less frequent and have minimal impact on the system's overall reliability. This is due to the dynamic task-to-VM mapping and adaptive scheduling based on real-time conditions.

In contrast, with their static scheduling methods, SPS and EDF are more prone to cascading failures as tasks are either dropped or stuck in overloaded VMs, leading to increased failure rates. The DQL, PPO, and SAC models show moderate improvements over SPS and EDF due to their ability to adapt to task load variations. However, they still fall short of ensuring fault tolerance over prolonged periods.

The Hybrid DDQL scheduler shows the highest MTBF across all node configurations, indicating the most reliable fault-tolerant system shown in Table 8 and Figure 8.

7.4. Task DMR

The DMR quantifies the number of tasks that fail to meet their deadlines relative to the total number of tasks submitted. A lower DMR signifies a better ability of the system to manage task execution within the specified time constraints. The Hybrid DDQL scheduler maintains a significantly lower DMR across all node configurations than the baseline models. This is a direct result of the scheduler's dynamic nature, which allows it to make real-time decisions based on task priority, VM availability, and failure status. The learning-based approach allows the scheduler to balance the trade-off between task completion time and system load, ensuring minimal violations of deadlines. In comparison, traditional methods like SPS and EDF struggle to handle deadline management effectively in dynamic environments. With its static nature, SPS often causes significant delays in task execution, whereas EDF, although deadline-sensitive, is not resilient to failures and overloads, which causes higher DMR.

As seen in Table 9 and Figure 9, the Hybrid DDQL scheduler demonstrates the lowest DMR and achieves superior task management under real-time conditions.

7.5. Throughput and energy efficiency

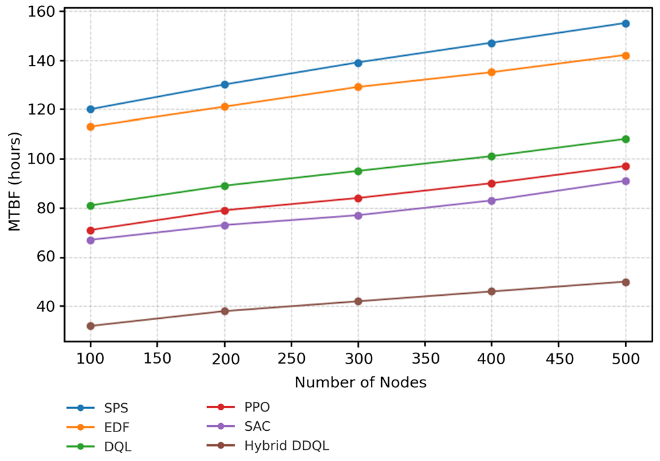
Throughput refers to the number of tasks successfully executed within a given period, whereas energy efficiency reflects the amount

Table 7
MTTR comparison

Number of nodes	SPS (seconds)	EDF (seconds)	DQL (seconds)	PPO (seconds)	SAC (seconds)	Hybrid DDQL (seconds)
100	120	110	80	70	65	30
200	130	120	85	75	70	35
300	140	130	90	80	75	40
400	150	140	100	90	85	45
500	160	150	110	100	95	50

Note: DDQL = Double Deep Q-Learning, DQL = Deep Q-Learning, EDF = Earliest Deadline First, MTTR = mean time to recovery, PPO = Proximal Policy Optimization, SAC = Soft Actor-Critic, SPS = Static Priority-Based Scheduling.

Figure 7
MTTR comparison



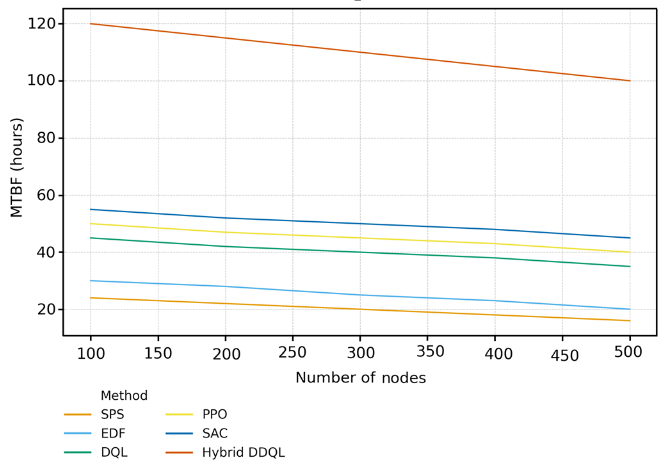
Note: DDQL = Double Deep Q-Learning, DQL = Deep Q-Learning, EDF = Earliest Deadline First, MTTR = mean time to recovery, PPO = Proximal Policy Optimization, SAC = Soft Actor-Critic, SPS = Static Priority-Based Scheduling.

Table 8
MTBF comparison

Number of nodes	SPS (hours)	EDF (hours)	DQL (hours)	PPO (hours)	SAC (hours)	Hybrid DDQL (hours)
100	24	30	45	50	55	**120**
200	22	28	42	47	52	**115**
300	20	25	40	45	50	**110**
400	18	23	38	43	48	**105**
500	16	20	35	40	45	**100**

Note: DDQL = Double Deep Q-Learning, DQL = Deep Q-Learning, EDF = Earliest Deadline First, MTBF = mean time between failures, PPO = Proximal Policy Optimization, SAC = Soft Actor-Critic, SPS = Static Priority-Based Scheduling.

Figure 8
MTBF comparison



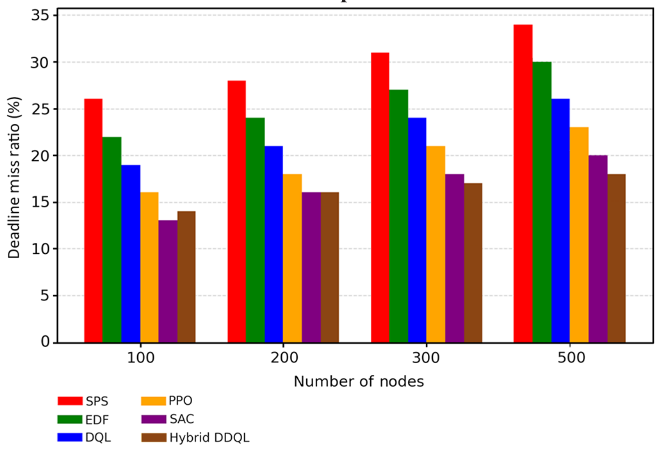
Note: DDQL = Double Deep Q-Learning, DQL = Deep Q-Learning, EDF = Earliest Deadline First, MTBF = mean time between failures, PPO = Proximal Policy Optimization, SAC = Soft Actor-Critic, SPS = Static Priority-Based Scheduling.

Table 9
DMR comparison

Number of Nodes	SPS (%)	EDF (%)	DQL (%)	PPO (%)	SAC (%)	Hybrid DDQL (%)
100	25	20	15	13	10	**5**
200	28	23	18	16	12	**7**
300	30	25	20	18	14	**10**
400	32	27	22	20	16	**12**
500	34	30	24	22	18	**15**

Note: DDQL = Double Deep Q-Learning, DMR = deadline miss ratio, DQL = Deep Q-Learning, EDF = Earliest Deadline First, PPO = Proximal Policy Optimization, SAC = Soft Actor-Critic, SPS = Static Priority-Based Scheduling.

Figure 9
DMR comparisons



Note: DDQL = Double Deep Q-Learning, DMR = deadline miss ratio, DQL = Deep Q-Learning, EDF = Earliest Deadline First, PPO = Proximal Policy Optimization, SAC = Soft Actor-Critic, SPS = Static Priority-Based Scheduling.

of power consumed during task execution. These two metrics are essential for assessing the overall operational efficiency of cloud computing systems. As shown in Table 10 and Figure 10, the proposed Hybrid DDQL scheduler consistently achieves the highest throughput across different node configurations compared to SPS, EDF, DQL, PPO, and SAC models. Although baseline approaches demonstrate gradual improvements over SPS and EDF, their performance still lags significantly behind Hybrid DDQL. The proposed model completes more tasks per unit time due to its adaptive task-to-VM allocation and efficient fault recovery, thereby enhancing system responsiveness under dynamic conditions.

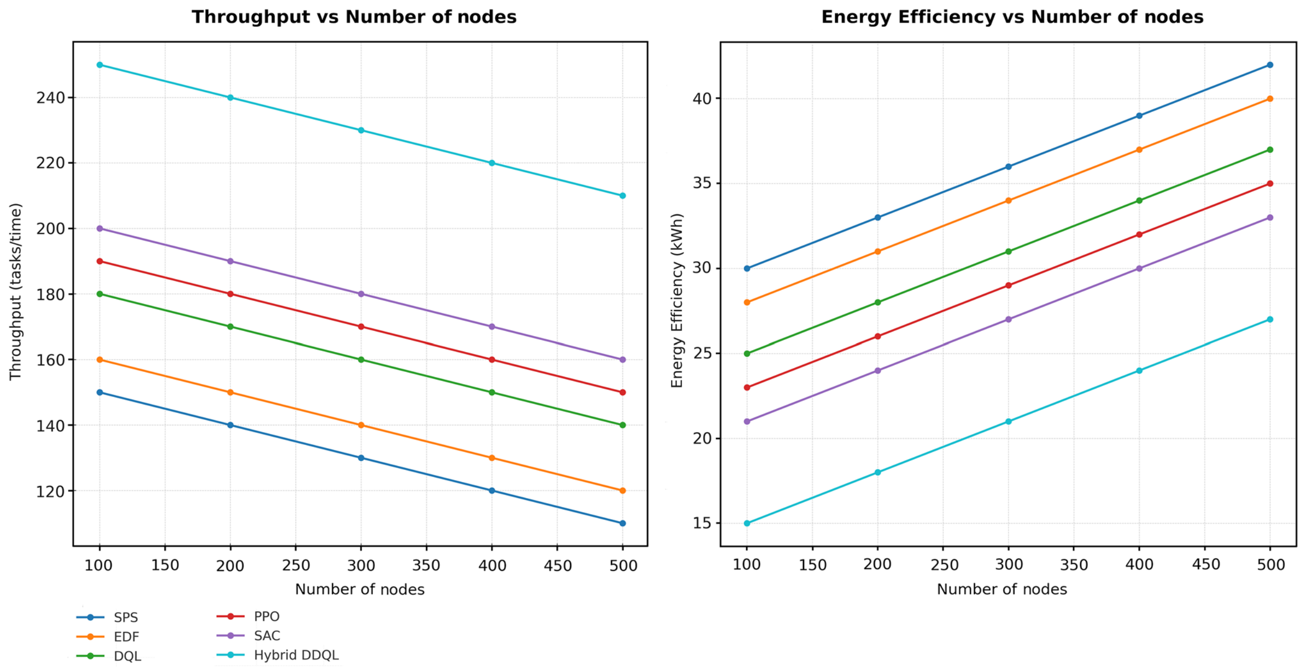
In terms of energy efficiency, Hybrid DDQL again outperforms all baselines, consuming the least energy across all node scales. Traditional models such as SPS and EDF exhibit the highest energy usage, attributed to their static nature and inability to adapt resource allocation dynamically. Learning-based models (DQL, PPO, and SAC) improve upon these baselines by reducing overhead through adaptive scheduling; however, their EC remains higher than Hybrid DDQL. By proactively balancing task deadlines, system load, and energy constraints, Hybrid DDQL minimizes wasted resources and ensures sustainable cloud operations. Overall, the comparative analysis highlights that the Hybrid DDQL scheduler not only maximizes throughput but also achieves superior energy efficiency, clearly distinguishing its performance from both traditional and single-algorithm RL approaches.

Table 10
Throughput and energy efficiency comparison

Number of nodes	SPS_ Throughput	EDF_ Throughput	DQL_ Through-put	PPO_ Through-put	SAC_ Through-put	Hybrid_ DDQL_ Through-put	SPS_ Energy	EDF_ Energy	DQL_ Energy	PPO_ Energy	SAC_ Energy	Hybrid_ DDQL_ Energy
100	150	160	180	190	200	250	30	28	25	23	21	15
200	140	150	170	180	190	240	33	31	28	26	24	18
300	130	140	160	170	180	230	36	34	31	29	27	21
400	120	130	150	160	170	220	39	37	34	32	30	24
500	110	120	140	150	160	210	42	40	37	35	33	27

Note: DDQL = Double Deep Q-Learning, DQL = Deep Q-Learning, EDF = Earliest Deadline First, PPO = Proximal Policy Optimization, SAC = Soft Actor-Critic, SPS = Static Priority-Based Scheduling.

Figure 10
Throughput and energy efficiency versus number of nodes



Note: DDQL = Double Deep Q-Learning, DQL = Deep Q-Learning, EDF = Earliest Deadline First, PPO = Proximal Policy Optimization, SAC = Soft Actor-Critic, SPS = Static Priority-Based Scheduling.

7.6. Comparative analysis with conventional mode

A comprehensive comparative analysis of the Hybrid DDQL scheduler versus traditional fault-tolerant scheduling models is necessary to demonstrate the supplementary benefits of employing RL methodologies in cloud computing environments. The hybrid DDQL approach combines DDQL, PPO, and SAC to give a powerful solution to the changing nature of resource management, task scheduling, and fault tolerance. The hybrid DDQL scheduler generally trumps all other models when the FTSR is 90%. This is far more than SPS and EDF, which do not work well when things go wrong, because they do not move. DQL, PPO, and SAC all show improvements, but none are as strong as the Hybrid DDQL scheduler. The hybrid DDQL scheduler has the smallest MTTR (30 seconds), which suggests that it is good at swiftly fixing mistakes and moving work around. The baseline techniques, especially SPS, take a lot longer to recover, which shows that they can not respond quickly to mistakes. You need to know MTBF in order to understand how reliable a system is. The hybrid DDQL scheduler suggests that there would be fewer failures over time, with the highest MTBF (120 hours). On the other hand, SPS and EDF fail more often, which lowers their MTBF values. The Hybrid DDQL scheduler has a low DMR (5%), which shows that it can meet deadlines. It does better than all the other models, such as PPO and SAC, which also do well but still miss more deadlines than the Hybrid DDQL strategy. The Hybrid DDQL scheduler has the highest throughput (250 tasks/time), which is the same as the RL approaches' ability to manage tasks well and recover quickly. Because they are immobile and can not handle changing workloads, traditional models such as SPS and EDF have lower throughput. Energy efficiency is a key consideration in contemporary cloud settings. The Hybrid DDQL scheduler is the most energy-efficient, using only 15 kWh. This is less than the baseline models, which use more energy since they do not handle tasks and recover from failures as well.

7.7. Practical deployment considerations

Although simulation-based evaluations demonstrate the efficiency of the proposed Hybrid DDQL scheduler, several practical deployment aspects require consideration. First, the hybrid integration of DDQL, PPO, and SAC inevitably increases computational overhead compared to single-algorithm schedulers. However, the majority of the training process is performed offline, and lightweight inference combined with model compression ensures that runtime scheduling decisions can still be made within near real-time constraints. Second, the proposed scheduler can be integrated into existing cloud management platforms such as OpenStack or Kubernetes by deploying it as an external scheduling module, leveraging existing APIs for VM placement and task assignment. This allows compatibility with widely used cloud infrastructures without significant architectural modifications. Finally, although the experiments were limited to 500 nodes, the ensemble mechanism scales effectively to larger systems since the computational complexity of weight updates and decision aggregation grows linearly with the number of actions rather than the number of nodes. Nevertheless, extending the framework to extremely large-scale infrastructures (e.g., thousands of nodes) will benefit from distributed training and hierarchical scheduling strategies, which are outlined as a future direction.

8. Conclusion

This study proposed a Hybrid DDQL scheduler for adaptive fault-tolerant scheduling in real-time cloud systems. By integrating value-based DDQL with policy-based RL methods such as SAC, the framework leverages the strengths of both paradigms to enhance adaptability, efficiency, and resilience against failures. Experimental results demonstrate that the proposed scheduler consistently outperforms conventional methods, including EDF, SPS, and single-algorithm RL models, by achieving higher FTSRs, faster recovery times, and improved performance metrics such as deadline adherence, energy efficiency, and system throughput. Practical deployment considerations were also addressed. Although hybrid integration increases training complexity, runtime inference remains lightweight through offline training and model compression. The scheduler is compatible with platforms such as OpenStack and Kubernetes, enabling integration through existing APIs. Furthermore, the ensemble mechanism scales linearly with task assignments, supporting larger infrastructures beyond the 500 nodes evaluated in this study. Future research will focus on validating the framework in real-world cloud testbeds, extending it to multi-cloud and edge environments, incorporating federated learning, and improving explain explainability of decision-making. Overall, the Hybrid DDQL scheduler represents a practical and effective solution for reliable, efficient, and adaptive cloud operations.

Acknowledgement

The authors gratefully acknowledge Sri Mata Amritanandamayi Devi (Amma), Chancellor, Amrita Vishwa Vidyapeetham, for her inspiration.

Funding Support

This work is sponsored by Sri Mata Amritanandamayi Devi (Amma), Chancellor, Amrita Vishwa Vidyapeetham.

Ethical Statement

This study does not contain any studies with human or animal subjects performed by any of the authors.

Conflicts of Interest

The authors declare that they have no conflicts of interest to this work.

Data Availability Statement

Data are available from the corresponding author upon reasonable request.

Author Contribution Statement

Chetankumar Kalaskar: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing – original draft, Writing – review & editing, Visualization, Supervision, Project administration. **Thangam Somasundaram:** Methodology, Validation, Formal analysis, Resources, Data curation, Writing – original draft, Writing – review & editing, Visualization, Supervision, Project administration, Funding acquisition.

References

[1] Botta, A., de Donato, W., Persico, V., & Pescapé, A. (2016). Integration of cloud computing and Internet of Things: A survey. *Future Generation Computer Systems*, *56*, 684–700. https://doi.org/10.1016/j.future.2015.09.021

[2] Shaukat, M., Alasmary, W., Alanazi, E., Shuja, J., Madani, S. A., & Hsu, C.-H. (2022). Balanced energy-aware and fault-tolerant data centre scheduling. *Sensors*, *22*(4), 1482. https://doi.org/10.3390/s22041482

[3] Beloglazov, A., & Buyya, R. (2012). Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers. *Concurrency and Computation: Practice and Experience*, *24*(13), 1397–1420. https://doi.org/10.1002/cpe.1867

[4] Wang, J., Li, S., Zhang, X., Wu, F., & Xie, C. (2024). Deep reinforcement learning task scheduling method based on server real-time performance. *PeerJ Computer Science*, *10*, e2120. https://doi.org/10.7717/peerj-cs.2120

[5] Shaukat, M., Alasmary, W., Alanazi, E., Shuja, J., Madani, S. A., & Hsu, C. H. (2022). Balanced energy-aware and fault-tolerant data center scheduling. *Sensors*, *22*(4), 1482. https://doi.org/10.3390/s22041482

[6] Zhou, G., Tian, W., Buyya, R., Xue, R., & Song, L. (2024). Deep reinforcement learning-based methods for resource scheduling in cloud computing: A review and future directions. *Artificial Intelligence Review*, *57*(5), 124. https://doi.org/10.1007/s10462-024-10756-9

[7] Sheng, S., Chen, P., Chen, Z., Wu, L., & Yao, Y. (2021). Deep reinforcement learning-based task scheduling in iot edge computing. *Sensors*, *21*(5), 1666. https://doi.org/10.3390/s21051666

[8] Wang, Z., Chen, S., Bai, L., Gao, J., Tao, J., Bond, R. R., & Mulvenna, M. D. (2023). Reinforcement learning based task scheduling for environmentally sustainable federated cloud computing. *Journal of Cloud Computing*, *12*(1), 174. https://doi.org/10.1186/s13677-023-00553-0

[9] Mangalampalli, S., Karri, G. R., Kumar, M., Khalaf, O. I., Romero, C. A. T., & Sahib, G. A. (2024). DRLBTSA: Deep reinforcement learning based task-scheduling algorithm in cloud computing. *Multimedia Tools and Applications, 83*(3), 8359–8387. https://doi.org/10.1007/s11042-023-16008-2

[10] Aburukba, R. (2025). Federated learning-driven IoT request scheduling for fault tolerance in cloud data centers. *Mathematics*, *13*(13), 2198. https://doi.org/10.3390/math13132198

[11] Liu, J., Wu, Z., Feng, D., Zhang, M., Wu, X., Yao, X., ... & Dou, D. (2023). Heterps: Distributed deep learning with reinforcement learning based scheduling in heterogeneous environments. *Future Generation Computer Systems*, *148*, 106–117. https://doi.org/10.1016/j.future.2023.05.032

[12] Kalaskar, C., & Somasundaram, T. (2024). Enhancing fault tolerance: Dual Q-learning with dynamic scheduling. *Indonesian Journal of Electrical Engineering and Computer Science*, *33*(2), 1150. https://doi.org/10.11591/ijeecs.v33.i2.pp1150-1168

[13] Kalaskar, C., & S., T. (2024). A graph neural network-based approach with dynamic multiqueue optimization scheduling (DMQOS) for efficient fault tolerance and load balancing in cloud computing. *International Journal of Intelligent Systems*, *2024*(1), 6378720. https://doi.org/10.1155/int/6378720

[14] Liu, X., Zha, Y., Yin, Q., Peng, Y., & Qin, L. (2015). Scheduling parallel jobs with tentative runs and consolidation in the cloud. *Journal of Systems and Software*, *104*, 141–151. https://doi.org/10.1016/j.jss.2015.03.007

[15] Gawali, M. B., & Shinde, S. K. (2018). Task scheduling and resource allocation in cloud computing using a heuristic approach. *Journal of Cloud Computing*, *7*(1), 4. https://doi.org/10.1186/s13677-018-0105-8

[16] Tong, Z., Deng, X., Chen, H., Mei, J., & Liu, H. (2020). QL-HEFT: A novel machine learning scheduling scheme base on cloud computing environment. *Neural Computing and Applications*, *32*(10), 5553–5570. https://doi.org/10.1007/s00521-019-04118-8

[17] Armstrong, R., Hensgen, D., & Kidd, T. (1998). The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions. In *Proceedings Seventh Heterogeneous Computing Workshop*, 79–87. https://doi.org/10.1109/HCW.1998.666547

[18] Hr, S., & S., T. (2021). A hybrid cloud approach for efficient data storage and security. In *2021 6th International Conference on Communication and Electronics Systems*, 1072–1076. https://doi.org/10.1109/ICCES51350.2021.9488938

[19] Banerjee, P., Roy, S., Sinha, A., Hassan, M. M., Burje, S., Agrawal, A., ..., & El-Shafai, W. (2023). MTD-DHJS: Makespan-optimized task scheduling algorithm for cloud computing with dynamic computational time prediction. *IEEE Access*, *11*, 105578–105618. https://doi.org/10.1109/ACCESS.2023.3318553

[20] Sharma, K., Hassan, M. M., Pandey, S. K., Saini, M., Doriya, R., Ojha, M. K., ..., & Soliman, N. F. (2023). Cloud based multi-robot task scheduling using PMW algorithm. *IEEE Access*, *11*, 146003–146013. https://doi.org/10.1109/ACCESS.2023.3344459

[21] Murad, S. A., Azmi, Z. R. M., Muzahid, A. J. M., Sarker, M. M. H., Miah, M. S. U., Bhuiyan, M. K. B., ..., & Bairagi, A. K. (2024). Priority based job scheduling technique that utilizes gaps to increase the efficiency of job distribution in cloud computing. *Sustainable Computing: Informatics and Systems*, *41*, 100942. https://doi.org/10.1016/j.suscom.2023.100942

[22] Murad, S. A., Azmi, Z. R. M., Muzahid, A. J. M., Bhuiyan, M. K. B., Saib, M., Rahimi, N., ..., & Bairagi, A. K. (2024). SG-PBFS: Shortest gap-priority based fair scheduling technique for job scheduling in cloud environment. *Future Generation Computer Systems*, *150*, 232–242. https://doi.org/10.1016/j.future.2023.09.005

[23] Yao, G., Ren, Q., Li, X., Zhao, S., & Ruiz, R. (2022). A hybrid fault-tolerant scheduling for deadline-constrained tasks in cloud systems. *IEEE Transactions on Services Computing*, *15*(3), 1371–1384. https://doi.org/10.1109/TSC.2020.2992928

[24] Rezaei Kalantari, K., Ebrahimnejad, A., & Motameni, H. (2020). Presenting a new fuzzy system for web service selection aimed at dynamic software rejuvenation. *Complex & Intelligent Systems*, *6*(3), 697–710. https://doi.org/10.1007/s40747-020-00168-x

[25] Shahid, M. A., Islam, N., Alam, M. M., Mazliham, M. S., & Musa, S. (2021). Towards Resilient Method: An exhaustive survey of fault tolerance methods in the cloud computing environment. *Computer Science Review*, *40*, 100398. https://doi.org/10.1016/j.cosrev.2021.100398

[26] Liu, Y., Wang, Z., & Zhou, D. (2021). Resilient actuator fault estimation for discrete-time complex networks: A distributed approach. *IEEE Transactions on Automatic Control*, *66*(9), 4214–4221. https://doi.org/10.1109/TAC.2020.3033710

[27] Liakath, J. A., Krishnadoss, P., & Natesan, G. (2023). DCCWOA: A multi-heuristic fault tolerant scheduling technique for cloud computing environment. *Peer-to-Peer Networking and Applications*, *16*(2), 785–802. https://doi.org/10.1007/s12083-022-01445-x

[28] Chen, G., Guan, N., Huang, K., & Yi, W. (2020). Fault-tolerant real-time tasks scheduling with dynamic fault handling. Journal of Systems Architecture, *102*, 101688. https://doi.org/10.1016/j.sysarc.2019.101688

[29] Sheeba, A., & Uma Maheswari, B. (2023). An efficient fault tolerance scheme based enhanced firefly optimization for virtual machine placement in cloud computing. *Concurrency and Computation: Practice and Experience*, *35*(7), e7610. https://doi.org/10.1002/cpe.7610

[30] Mushtaq, S. U., Sheikh, S., & Nain, A. (2024). The response rank based fault-tolerant task scheduling for cloud system. In *Proceedings of the 2023 1st International Conference on Advanced Informatics and Intelligent Information Systems*, 37–48. https://doi.org/10.2991/978-94-6463-366-5_5

[31] Negi, S., Rauthan, M. M. S., Vaisla, K. S., & Panwar, N. (2021). CMODLB: An efficient load balancing approach in cloud computing environment. *The Journal of Supercomputing*, *77*(8), 8787–8839. https://doi.org/10.1007/s11227-020-03601-7

[32] Mani, M., & Kaur, L. (2021). Efficient enhanced sleep awake scheduling using fuzzy logic and neural networks: A review. *International Journal of Scientific Research in Science, Engineering and Technology*, *8*(2), 18–24. https://doi.org/10.32628/IJSRSET21828

[33] Shiekh, S., Shahid, M., Sambare, M., Haidri, R. A., & Yadav, D. K. (2023). A load-balanced hybrid heuristic for allocation of batch of tasks in cloud computing environment. *International Journal of Pervasive Computing and Communications*, *19*(5), 756–781. https://doi.org/10.1108/IJPCC-06-2022-0220

[34] Singh, A. N., & Prakash, S. (2018). WAMLB: Weighted active monitoring load balancing in cloud computing. In *Big Data Analytics: Proceedings of CSI 2015*, 677–685. https://doi.org/10.1007/978-981-10-6620-7_65

[35] Radojević, B., & Žagar, M. (2011). Analysis of issues with load balancing algorithms in hosted (cloud) environments. In *2011 Proceedings of the 34th International Convention MIPRO*, 416–420.

[36] Poola, D., Salehi, M. A., Ramamohanarao, K., & Buyya, R. (2017). A taxonomy and survey of fault-tolerant workflow management systems in cloud and distributed computing environments. In I. Mistrik, R. Bahsoon, N. Ali, M. Heisel, & B. Maxim (Eds.), *Software architecture for big data and the cloud* (pp. 285–320). Elsevier. https://doi.org/10.1016/B978-0-12-805467-3.00015-6

[37] Kalaskar, C., & Thangam, S. (2023). Fault tolerance of cloud infrastructure with machine learning.

Cybernetics and Information Technologies, *23*(4), 26–50. https://doi.org/10.2478/cait-2023-0034

[38] Sheikh, S., Nagaraju, A., & Shahid, M. (2021). A fault-tolerant hybrid resource allocation model for dynamic computational grid. *Journal of Computational Science*, *48*, 101268. https://doi.org/10.1016/j.jocs.2020.101268

[39] Tamilvizhi, T., & Parvathavarthini, B. (2019). A novel method for adaptive fault tolerance during load balancing in cloud computing. *Cluster Computing*, *22*(S5), 10425–10438. https://doi.org/10.1007/s10586-017-1038-6

[40] Attallah, S. M. A., Fayek, M. B., Nassar, S. M., & Hemayed, E. E. (2021). Proactive load balancing fault tolerance algorithm in cloud computing. *Concurrency and Computation: Practice and Experience*, *33*(10), e6172. https://doi.org/10.1002/cpe.6172

[41] Mohmmed, T., & Abdalrahman, N. (2021). A load balancing with fault tolerance algorithm for cloud computing. In *2020 International Conference on Computer, Control, Electrical, and Electronics Engineering*, 1–6. https://doi.org/10.1109/ICCCEEE49695.2021.9429597

[42] Shahid, M. A., Islam, N., Alam, M. M., Su'ud, M. M., & Musa, S. (2020). A comprehensive study of load balancing approaches in the cloud computing environment and a novel fault tolerance approach. *IEEE Access*, *8*, 130500–130526. https://doi.org/10.1109/ACCESS.2020.3009184

[43] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ..., & Hassabis, D. (2013). Playing Atari with deep reinforcement learning. *arXiv Preprint: 1312.5602*. https://doi.org/10.48550/arXiv.1312.5602

[44] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv Preprint: 1707.06347*. https://arxiv.org/abs/1707.06347

[45] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., ..., & Levine, S. (2018). Soft actor-critic algorithms and applications. *arXiv Preprint: 1812.05905*. https://doi.org/10.48550/ARXIV.1812.05905

[46] Baruah, S. K., Rosier, L. E., & Howell, R. R. (1990). Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, *2*(4), 301–324. https://doi.org/10.1007/BF01995675

[47] Liu, C. L., & Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, *20*(1), 46–61. https://doi.org/10.1145/321738.321743

How to Cite: Kalaskar, C., & Somasundaram, T. (2025). Adaptive Fault-Tolerant Scheduling in Dynamic Clouds. *Journal of Computational and Cognitive Engineering*. https://doi.org/10.47852/bonviewJCCE52026266