SSRG International Journal of Electronics and Communication Engineering
ISSN: 2348-8549/ https://doi.org/10.14445/23488549/IJECE-V13I4P127

Volume 13 Issue 4, 317-347, April 2026

Original Article

EdgeSchedAI: An Energy-Aware and Latency-Aware Task Scheduling Framework for IoT Applications in Edge-Cloud Environments

Nagendar Yamsani[1], Chenna Reddy P[2]

[1]*School of Computer Science and Artificial Intelligence, SR University, Warangal, Telangana, India,*
[1]*Research Scholar, JNTUA.*
[2]*Department of Computer Science and Engineering, JNTUA College of Engineering, Ananthapuramu, Jawaharlal Nehru Technological University Anantapur, Anantapur, Andhra Pradesh, India.*

[1]*Corresponding Author : nag.res.tech@gmail.com*

Received: 23 January 2026 Revised: 23 February 2026 Accepted: 24 March 2026 Published: 30 April 2026

Abstract *- Given the explosive growth of IoT applications, there has been a growing need to schedule tasks effectively in edge–cloud computing settings, which are usually characterized by stringent latency requirements, constrained energy resources, a dynamic workload mix, and device mobility. Existing cloud-centric approaches suffer from high communication latency, while heuristic and meta-heuristic Schedulers are unable to adapt quickly to shifting system conditions. To tackle these issues, this paper offers an energy- and latency-aware DRL-based task system for scheduling IoT applications in Cloud-edge settings. Firstly, to express the task-scheduling issue as a sequential decision-making problem and then propose a DQN-based dynamic resource allocation model that adaptively schedules tasks across edge and cloud resources determined by changing system conditions throughout time, workload profiles, and SLA requirements. By balancing and adapting speed across a wide range of operational conditions, the proposed scheduler jointly optimizes energy consumption and task latency within a unified learning framework. The proposed approach outperformed two baselines—representative heuristic and learning-based—by 22–35% and 18–30%, respectively, in average task-latency and task energy consumption, with SLA satisfaction and task success rates above 95% under moderate workloads. Provided evidence of dependable learning behavior through conjunctions of ascent and/or descent and via convergence and stability analyses. Finally, scalability and stress testing observations confirm graceful performance degradation under overload conditions. The proposed framework provides a feasible and scalable solution that ensures sustainability, responsiveness, and service resilience for real-time IoT deployments. This enables deployment on any edge device, with adaptive load balancing and multi-objective optimization, allowing the application of this framework across numerous domains, such as real-time, industrial IoT, and smart cities monitoring systems, in dynamic edge–cloud environments.*

Keywords *- Deep Reinforcement Learning, Edge–Cloud Computing, Energy-Aware Scheduling, IoT Task Scheduling, Latency-Aware Optimization.*

1. Introduction

The rapid expansion of IoT applications has revolutionized modern computing infrastructure, placing ever-increasing demands on data processing speed, energy efficiency, and service reliability. Smart city, industrial automation, healthcare monitoring, and intelligent transportation systems, etc., must produce billions of time-sensitive data events per day, requiring computing paradigms that can accommodate in-situ processing (near the source of its origin) for performing relevant tasks and that avoid the routing of all the generated data to a central cloud datacenter. Automation in industry, smart cities, healthcare surveillance, intelligent transportation systems, and other apps produces large volumes of time-sensitive data that must be processed efficiently and reliably. The communication latency and bandwidth bottleneck of conventional cloud-centric models, and the insufficient energy and processing resources of purely edge-based processing, may not meet the ultimate requirement. As a result, intelligent task scheduling across cloud levels and edges has become a critical research challenge, directly influencing task completion times, energy consumption, and overall quality of service. Various scheduling frameworks have been studied in recent years, including heuristic approaches, meta-heuristic optimization, and machine learning-based techniques. To extend device lifetime and minimize operational costs, energy-aware

schedulers have been proposed [1], while latency-aware and deadline-driven methods target responsiveness for real-time IoT workloads [2]. Deep Reinforcement Learning (DRL) is attractive for adaptive scheduling policies that operate in dynamic environments [3]. Nonetheless, most prior work optimizes a single objective in isolation — either minimizing energy consumption or reducing latency — without addressing both simultaneously under SLA constraints. It rarely provides guarantees on SLA violations, scalable designs, or their stability under workload variations [4, 5]. Additionally, most such learning-based solutions are limited to simulation-level validation and neglect convergence properties or robustness under overload conditions.

The literature has three enduring and unanswered gaps. However, first, most existing schedulers are single-objective: energy-aware methods [1, 7] ignore latency. In contrast, latency-driven methods [2, 33] do not consider energy budgets, necessitating a choice by the system designer as to which goal to prioritize at the expense of the other, entscheidend. For the second, although learning-based approaches have jointly modeled energy and latency 3, 9, the scheduling policy is typically trained without enforcing a guaranteed service level - meaning the system cannot ensure that tasks complete within the time and energy contracts.

Third, although the overload scenarios and workload shifts are common in real large-scale IoT deployments, the robustness of proposed schedulers under overload conditions and their stability across workload shifts have never been evaluated [4, 5]. The present work is motivated by these three gaps: fragmented objective optimization, lack of SLA enforcement, and untested robustness.

To directly address these identified gaps, this paper proposes EdgeSchedAI, an energy-aware and latency-cognizant task-scheduling system for IoT applications in edge–cloud environments that optimally co-schedules multiple conflicting objectives under dynamic conditions. To meet this demand, the current study introduces a DR-based framework for task-scheduling learning in IoT applications in edge–cloud environments that accounts for both energy and latency. The purpose of this study is to develop an intelligent scheduler that accounts for real-time system state, workload characteristics, and SLA constraints, balancing performance criteria such as energy efficiency, latency, and service reliability by dynamically assigning duties to the edge or cloud resources.

The main novelty of this work is the joint replacement of these objectives with a single, integrated, multi-objective learning strategy using a DQN agent that addresses energy usage, task latency, and SLA compliance. This work is novel compared to existing approaches for three main reasons. To start with, energy-aware schedulers like Aslanpour et al. Optimization for consumption without latency guarantees [1] and Delgado and Famaey [8], and latency-driven methods such as Lim [33] and Bukhsh et al. While [34] minimizes delay without energy awareness, EdgeSchedAI is the first work to account for both delay and energy as coequal objectives within a single unified reward function. Secondly, as opposed to the DRL-based scheduler of Sellami et al. Workflow of Jayanetti et al. [2]. The underlying logic in existing approaches [5, 7, 8] is partially modulated SLAs, and no robustness evaluation is made; proposed framework can be robustly evaluated by itself as it explicitly models the SLA constraints into the learning objective and then verifies its scheduling stability under dynamic workload shifts and overload [9] third, unlike the majority of learning-based works-including Wang and Yang [10] and Zhu et al. While many approaches in the literature [14] report nothing more than modest performance metrics, without any material tests of convergence or ablation, the work here provides a full reliability signature: the reward convergence curves, policy stability analysis, component-wise ablation, and graceful degradation under stress, which allows all aspects of the learning behavior to be fully verified.

The primary contributions made by this work are:(i) a unified DQN-based task scheduling framework (EdgeSchedAI) that formulates joint energy–latency minimisation as a sequential decision-making problem under explicit SLA constraints — addressing the fragmented single-objective optimisation prevalent in existing work; (ii) an adaptive scheduling policy to cope with dynamic IoT workloads and heterogeneous edge–cloud resources; (iii) a comprehensive experimental evaluation including performance comparison, convergence analysis, ablation study, scalability assessment and stress testing; and (iv) a direct comparative positioning against five representative state-of-the-art methods [1, 2, 4, 5, 9] across energy awareness, latency awareness, SLA enforcement, and adaptability dimensions, demonstrating that EdgeSchedAI is the only framework achieving full coverage across all four criteria simultaneously.

This is the format for the remainder of the paper. Section 2 provides a thorough analysis of the literature. of work on edge–cloud task scheduling. The proposed system model and scheduling method are then described in Section 3. Section 4 reports the experimental findings and performance assessment. In Section 5, provide more detailed explanations of the findings and the limitations of the study. Section 6 summarises the paper and outlines pathways for future work.

2. Related Work

The following Section provides a survey of recent edge-cloud task-scheduling work, focusing on energy-latency trade-offs, SLA compliance, and learning-based task allocation for IoT workloads. Task scheduling is an extensively studied yet fast-evolving problem in edge–cloud environments owing to

the increasing diversity of IoT workloads and stringent performance demands of real-time applications. Work in this area can generally be classified into four types — (i) Energy-aware approaches that reduce power consumption at edge nodes and cloud servers, (ii) Latency-aware and deadline driven approaches that optimise task responsiveness [8], (iii) SLA-aware approaches that enforce contractual service guarantees [9], and (iv) learning-based approaches, especially DRL, that adjust the scheduling decisions dynamically [10]. In the following subsections, the survey representative for each category is presented, and the gaps that inspire the current work are highlighted.

2.1. Energy-Aware Scheduling in Edge–Cloud Systems

Aslanpouret al.[1] The introduction of an Energy-conscious scheduling for serverless edge computing improved the availability of bottleneck nodes and service-level satisfaction. Future studies could evaluate long-term performance across different renewable energy pathways and examine scalability in larger edge ecosystems. DRL-based scheduling system in fog-IoT recognizing energy-efficiency with minimum latency, Sellami et al. [2], but is limited to job complexity and needs real-time evaluation.

However, it is limited to simulations, and real-life implementation will be explored in future work. Xun et al. [3] proposed a DQ-L job scheduler for edge clouds. The Energy-Aware Real-time Workflow Scheduling Algorithms (EAWSA) proposed in Zhii et al. [4] is limited to simulation use; the approach will be validated in a future study. Beyond that, this work is limited to simulations; searching for a real-world or practical Fog-Cloud task migration will be addressed in future studies (Mahapatra et al.).[5].

2.2. Latency-Aware and Deadline-Driven Scheduling

The work of Raju and Mothku [6] applied fuzzy reinforcement learning to fog task scheduling and is intended for real-time implementation; however, their assessment is based solely on simulation. This paper proposes an energy-conscious serverless edge system scheduler, named Faas House. Scalability is a future work and an offline modeling limitation, as noted by Aslanpour et al. [7]. The energy-aware scheduler proposed by Delgado and Famaey [8] for battery-free Internet of Things devices is constrained by a short time horizon. It aims to make it more applicable to real-world settings in future work mentioned a DRL-based edge-to-cloud workflow scheduler, Jayanetti et al. [9], that has been validated only in simulation, with limited real-world testing; scalability aspects could be the next direction for the work. Wang and Yang [10] improved efficiency by applying DRL to edge-cloud scheduling. However, the method can still be further enhanced by accelerating learning speed and convergence, and by increasing robustness and fault tolerance.

Heuristics for device-edge-cloud job scheduling are provided in Li et al. [11], but the results are close to ideal, and high scalability and energy efficiency remain needed. By optimizing edge-cloud task offloading to minimize latency and ensure task execution stability, these simulations show that the hybrid approach proposed in Fan et al. [12] outperforms existing techniques. On the side, Zhang et al.[13] presented a multi-objective resource scheduling model for task time, resource cost, load balance, and user satisfaction in an edge cloud architecture.

The authors in Zhu et al.[14] proposed an intelligent workflow scheduler using GNN and PPO to improve QoSin for dynamic IoT workloads in edge-cloud networks. An autonomic cloud computing work-scheduling system that minimizes time, utilization, and security using evolutionary algorithms (Malathi et al.).[15].

2.3. SLA-Aware and QoS-Driven Scheduling

Multi-cloud task scheduling solutions have been reviewed in Zhang et al. [16], and their independent and workflow scheduling challenges, optimization approaches, and future research directions are discussed. QoS-SLA-aware Optimization for IoT-QoS-based Service Placement in Fog-cloud System to optimize service time, energy, prices, and SLA Toghyani et al. [17] A real-time cost-effective task scheduling approach for fog-cloud based IoT applications was proposed in Abohamama et al. [18], which executed tasks with lower cost, failure rate, and execution time than other methods in the same class.

Karthikeyan and Balamurugan [19] propose an effective load-balancing strategy for cloud-IoT systems based on user and application requirements that minimizes energy consumption, migrations, and SLA violations. In IoT fog multi-cloud systems, Kanbar and Faraj [20] enhanced quality of service and SLA; RADISH is a model for optimizing job scheduling, resource allocation, and load balancing.

ALTS for Cloud task scheduling was proposed by Mubeen et al. [21]; it improves the make span, reduces SLA violations, and reduces resource consumption compared with GA- and ACO-based approaches. Alsadie [22] reviewed fog task scheduling approaches, identified significant challenges, and offered future directions in machine learning, federated learning, and explainable AI.

The proposed D-NSGA-II improves fog task scheduling, reducing energy and latency, and outperforms the existing meta-heuristics in the conducted experiments Mousavi et al. [23]. The DoSP architecture Sriraghavendra et al.[24] reduces the response time of IoT services requested in fog-cloud environments by using a GA to optimize service placement.

More extensive scenario validation will be done in future development. INSCM was proposed for MEC job scheduling tasks to balance execution time and energy (Li et al.).[25].

Table 1. Summary of Representative Edge–Cloud Task Scheduling Studies and Identified Research Gaps

Ref.	Authors	Core Technique	Optimization Focus	Evaluation Setting	Key Contributions	Identified Research Gap
[1]	Aslanpour et al.	Energy-aware heuristic scheduling	Energy efficiency, quality of service	Simulation	Improves bottleneck node uptime in serverless edge systems	Lacks scalability analysis and multi-objective optimization
[2]	Sellami et al.	DRL-based scheduling	Energy, latency	Simulation	Demonstrates reduced latency using DRL in fog-IoT networks	Limited job complexity handling and real-world validation
[3]	Xun et al.	Deep Q-learning	Delay, energy	Simulation	Learns adaptive scheduling policies for edge clouds	No SLA modeling or deployment-level validation
[4]	Zhii et al.	EAWSA workflow scheduling	Energy, real-time execution	Simulation	Energy-aware workflow scheduling for device-edge-cloud systems	Restricted to workflows; lacks adaptability and scalability
[5]	Mahapatra et al.	Fuzzy-BLWJAYA optimization	Latency, energy	Simulation	Hybrid fuzzy–meta-heuristic task migration approach	No learning capability or dynamic workload adaptation
[6]	Raju and Mothku	Fuzzy reinforcement learning	Delay, energy	Simulation	Incorporates reinforcement learning in fog task scheduling	SLA compliance and stability analysis are not addressed
[9]	Jayanetti et al.	DRL workflow scheduler	Energy, execution time	Simulation	Optimizes precedence-constrained workflows in edge–cloud	Limited scalability and convergence analysis
[14]	Zhu et al.	GNN + PPO-based scheduling	Quality of service, workflow efficiency	Simulation	Intelligent workflow scheduling using graph learning	High complexity; lacks energy–latency joint optimization

2.4. Deep Reinforcement Learning-Based Scheduling

Despite this, SD-ECC with the JDATP method, as proposed by Tang et al. [26], faces challenges such as vendor heterogeneity and extensibility requirements, warranting further research [33]. To minimize IoT energy consumption, [27] proposed a job-scheduling approach based on bipartite matching for edge-cloud cooperation. Sun et al. [28] proposed iGATS, an evolutionary algorithm-based IoT-edge-cloud network scheduler that optimizes energy consumption while minimizing latency. To minimize energy consumption and deadline violations in fog job scheduling, the PSG and PSG-M algorithms are represented in Azizi et al.[29]. Wang et al. BTTO and AO techniques are proposed for optimal job offloading in edge-cloud systems to improve service volume and energy efficiency [30].

In another work, Thotakura et al. [31] proposed an evolutionary algorithm-based dynamic mechanism for scheduling work in edge-cloud systems to maximize resource utilization and meet deadlines. In mobile edge-cloud networks, Zhang et al.[32] proposed a method that combines semi-definite relaxation and DRL for Job Offloading and Resource Scheduling. Latency-aware job scheduling for efficient IoT applications in foggy environments using an Artificial Neural Network (ANN), Lim et al. [33]. Abstract—The LAAECHA algorithm improves the Internet of Things (IoT task processing with the benefits of enhanced network presence, system reliability, and reduced delay in decentralized edge computing, Bukhsh et al. [34]. Compared to other systems, JANUS is effective at handling IoT traffic by prioritizing latency-sensitive streams, thereby increasing throughput while reducing latency (et al. [35]).

Proposed a heuristic-based method focusing on latency reduction and resource utilization through MLFQ for task scheduling and classification in Mahapatra et al.[36]. Proposes an MLFQ-based heuristic scheduling and classification solution to minimize latency while optimizing resource utilization. The work by Hassan et al.[37] proposes a latency-, computational power-, and data volume-optimized approach for deploying IoT applications in a heterogeneous fog environment. ultimately, Basir et al. [38] proposed using the Outer Approximation Algorithm (OAA) to select cloudlets in fog networks to reduce latency. Deldari and Holghinezhad

[39] presented a cost-optimal scheduling method that guarantees deadline satisfaction for IoT applications running over hybrid fog-cloud scenarios. Arora and Singh [40] proposed the HSMF approach to place modules in the fog-cloud setting optimally.

In Table 1, representative edge–cloud task-scheduling approaches are summarised, along with the main research gaps that motivate the proposed system. Established approaches mainly target single-optimization of energy, latency, and workflow execution in isolation and often validate their solutions through simulation-based studies. In each of the four categories surveyed above, a clear pattern of limitations appears. Although performance is achieved through energy-aware methods [1, 7, 8], latency constraints are neglected. Latency-aware approaches [2, 33-35] decrease response time while ignoring any energy budgets On the other hand, SLA-oriented approaches [17-19] achieve service guarantees [18], however, they rely on static or heuristic decision rules and model how to allocate resources for services without any adaptive learning. The most sophisticated DRL-based schedulers [3, 9, 10, 14] are likewise assessed solely on their average performance and do not provide convergence analysis, ablation evidence, or stress-tested results under overload. No previous work has combined joint energy–latency optimisation, explicit SLA enforcement, and verified robustness within a unified adaptive learning framework; this is the primary contribution of this work.

3. Materials and Methods

This section presents the materials and methods used to design and analyze the proposed energy- and latency-aware task scheduling framework. It describes the system model, IoT application deployment, scheduling algorithm, optimization strategy, and evaluation methodology implemented within the context of edge-cloud computing.

3.1. System Overview

The system is summarised in Figure 1, an architecture for energy- and Latency-conscious task planning in IoT applications in an edge-cloud context, where scheduling is based on task characteristics such as resource usage and deadlines. This architecture comprises edge devices, cloud infrastructure, and a task scheduling framework that minimises Energy usage and latency while meeting SLA requirements.

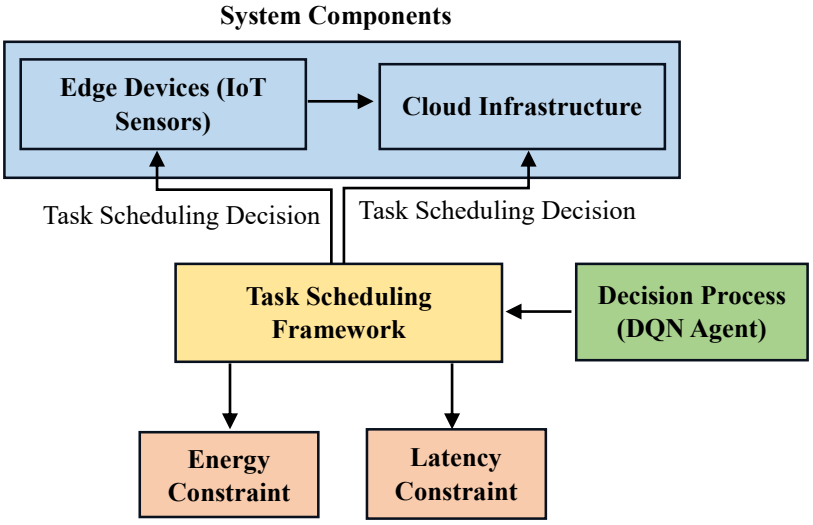

Fig. 1 System Architecture for Energy-Aware and Latency-Aware

Edge devices in this system are data sources, such as IoT sensors, cameras, and wearables. These devices create the workflows that need to be processed. Edge devices, due to their limited resources (e.g., CPU, memory, and bandwidth), can only perform simple, less computationally intensive tasks locally. However, the conditional functions that require greater processing power or require strict latency are offloaded to the Cloud for further processing.

The cloud infrastructure is a centralized resource pool. More complex tasks will require more computational resources to be solved. Cloud resources can be scaled, allowing the system to handle larger or more data-intensive projects. On the other hand, this approach only provides low edge latency; the cloud generally has higher latency due to the communication overhead of transferring tasks from edge devices to the cloud. Cloud processing requires minimizing data transfer time alongside completing the task to reduce execution time.

The scheduling framework is crucial in determining whether a task should be completed on the edge or transferred to the cloud. Depending on the nature of the task, energy demand, computational complexity, and latency requirements,

such a decision is made. In brief, the framework employs a DQN agent that learns to allocate resources over time. It continues to adjust to shifts in the amount of work and the resources available, as well as SLA constraints, ensuring that each task is executed at the minimum energy cost and delay and that each SLA is met.

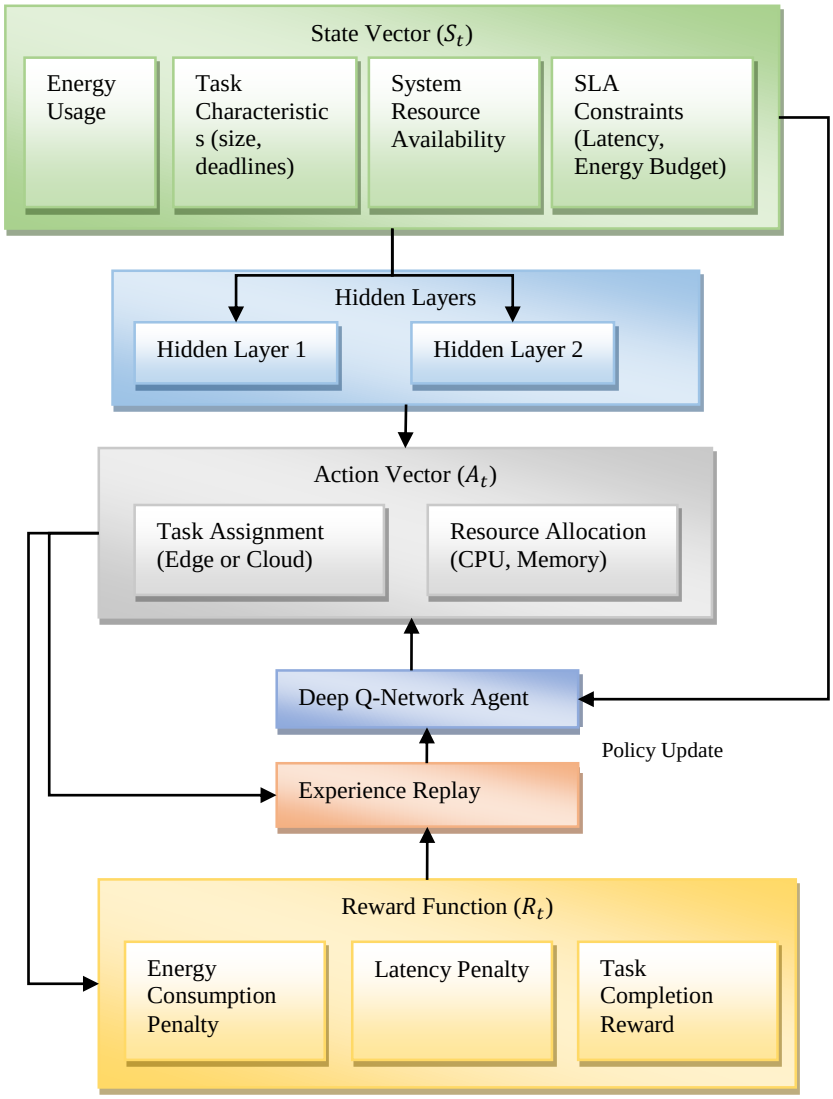

Fig. 2 DQN-based Task Scheduling Model (EdgeSchedNet)

The communication network connects edge devices and the cloud infrastructure, enabling data exchange between them. Essentially, the network is the primary means of communicating functions to the cloud from end devices with minimal latency. Because even a minor breach of a reservation can cause a significant drop in application performance, SLA-aware communication is crucial for achieving real-time SLA compliance.

A dynamic task scheduling method is proposed to adapt to real-time edge-cloud environment conditions and optimize service performance. The scheduling framework calculates the cost of processing the task on the edge device and offloading it to the cloud based on three criteria: the edge device's available resources, the task's processing urgency, and whether the process complies with SLA conditions. Subsequently, the system runs that task either at the edge or in the cloud, tracking whether it meets its energy and latency requirements.It only learns from history as it happens; it learns from earlier decisions and builds on the last decision in the context of past decisions to drive performance over time. TaInC can handle diverse workloads with variable resource availability; this flexibility makes it a practical choice for real-time IoT applications in edge-cloud environments. This provides the appropriate balance to meet the diverse and demanding needs of IoT jobs in edge-cloud systems by optimizing resources, tasks, and communication within the edge innovation platform.

At the center of this system is the real-time decision-making step, which determines where and how to process tasks in the edge-cloud-based computing environment, as illustrated in the task scheduling model in Figure 2. It effectively generates energy-efficient task assignments that meet the latency constraints of both function types, thereby ensuring SLA requirements are met. This framework utilizes a DQN-based agent that gradually improves its decision-making in task scheduling. This type of learning is essential for the system to adapt to changing times and workloads.

The importance of coordination in the system is through state representation (a textual representation of the system's state, where the data within the state captures all possible details to support related decisions), such as TP. Its associated paintings will include the size, task urgency based on their properties, the types of energy needed for the tasks or the energy usage required, the types of resources available in the edge-cloud environment, etc., and the SLA constraints characteristic, which are defined. A state that is input to the DQN agent, representing where each task is processed (at the edge, or offloaded to the cloud)

The DQN agent chooses actions from the available action space using the current state and drives the entire decision-making process. These actions include distributing tasks for edge processing or sending them to the cloud. The DQN agent will learn through reinforcement learning to evaluate each player's outcomes and select actions, ensuring it consistently allocates tasks more effectively. It then receives a reward or penalty for each decision and learns to adjust its policies, minimizing energy and latency while keeping tasks within the SLA constraints.

In principle, the incentive system is crucial for the DQN agent to learn. The framework provides positive rewards when the task meets all energy and latency constraints, and negative penalties when it fails to meet the SLA requirements. Hence, such an agent is driven to select actions that efficiently advance the task while minimizing energy consumption and delay, making it suitable for real-time applications.

Lastly, to improve learning, the system utilizes experience replay, which stores past interactions in a buffer. During training, sample these experiences randomly, breaking the correlations between consecutive experiences and helping the agent generalize better to previously unseen experiences. It allows the agent to learn more effectively by being exposed to a varied range of task types and system conditions.

After the DQN agent chooses an action, the task is processed at the edge (or possibly offloaded to the Cloud for execution). When processing on the edge, the task is completed using local resources. When offloaded to the cloud, the edge sends the task via the communication network. It forwards it to the cloud infrastructure, which then executes it and returns the result to the edge or the requesting system. The task's performance is then monitored to ensure it meets the SLA targets.

Once a task is performed, the DQN agent outputs a reward and changes the system state. The system utilizes this feedback to refine the agent's decision-making policy, thereby enhancing its ability to schedule tasks effectively over time. This agent ultimately becomes good at(making decisions on) task allocation, leading to maximum resource utilization and task completion(KSO, within its SLA constraints).

This task-scheduling framework enables the system to adapt dynamically to varying workloads and resource management issues. Using the DQN agent, the system gradually learns to make more informed scheduling decisions, enhancing overall task-processing efficiency while ensuring it can support real-time IoT applications in edge-cloud systems. This paper's notations are shown in Table 2.

Table 2. Notations Used in the Proposed System

Notation	Description
S_t	State vector at time t, representing the current system status (e.g., resources, task characteristics).
A_t	Action space at time t, representing the possible scheduling decisions (e.g., edge processing or cloud offloading).
E_{total}	Total energy consumption for all tasks, calculated as the sum of energy consumed by each task (Equation 8).
L_{total}	Total latency for a task, including edge processing time, cloud processing time, and task transmission delay (Equation 11).
R_t	Reward at time t, representing the task execution outcome (penalties for energy consumption and latency, and reward for task completion).
$Q^*(s_t,a_t)$	Optimal Q-value for the state s_t and action a_t, representing the expected future reward (Equation 4).
E_i	Energy consumed by task i during execution at the edge or in the cloud (Equation 10).
D_{trans}	Transmission delay for a task when offloaded to the cloud, calculated as task size divided by network bandwidth (Equation 3).
T_{edge}	Time taken for task execution at the edge device.

T_{cloud}	Time taken for task execution in the cloud.
L_{edge}	Processing time for a task at the edge.
L_{cloud}	Processing time for a task in the cloud.
$R_{\text{allocation}}$	Resource allocation function that ensures the efficient distribution of tasks between edge devices and cloud resources (Equation 6).
A_t^{pruned}	Pruned action space at time t, reducing unnecessary actions to improve decision-making efficiency (Equation 5).
γ	Discount factor for future rewards in the reinforcement learning model, used to balance immediate and future rewards.
n	Total number of tasks processed in the system.
f(·)	Function representing resource allocation based on available edge resources, cloud resources, and task requirements (Equation 6).
Success Rate	Percentage of tasks successfully completed within their SLA-defined constraints (Equation 12).
S_{t+1}	Updated system state after task allocation and execution, resulting from the decision-making process at time t.
L_{total}	Total latency for task completion, including processing and transmission delays (Equation 9).

3.2. IoT Application Deployment

To meet the real-time processing requirements of IoT tasks, IoT applications are deployed in a cloud-edge setting (tasks can be generated by numerous edge devices such as sensors, wearables, or cameras). These are computationally complex, time-sensitive, and deadline-oriented tasks. This deployment architecture ensures that such tasks are performed efficiently, thereby reducing energy use and fulfilling various latency constraints.

The edge devices in this configuration are important because they perform early task processing. These are devices that continuously observe the surroundings and gather data in real time. Conversely, edge technology has limited computational resources (CPU, memory, and bandwidth) and can efficiently process smaller, simpler tasks locally. However, when task complexity exceeds the edge device's processing capabilities or latency requirements are stricter, the task is offloaded to the cloud for higher-level processing.

Cloud infrastructure is the resource pool that provides scalable computing resources for more complex tasks that are not possible to perform at the edge. Compared to edge devices, the cloud provides much more computational resources, but the trade-off is higher latency due to communication overhead between the edge and the cloud. Therefore, in task scheduling, the latency incurred by transmitting tasks to the cloud must be taken into account, particularly for low-latency, real-time IoT applications.

Also integrated is a task scheduling framework that determines whether a task should execute at the edge or be processed in the cloud. The DQN agent is responsible for this decision-making; it incrementally learns the optimal task allocation policy from the actual state of the system, i.e., the accessible cloud and edge resources, as well as the energy consumption and latency constraints. There are various parameters that are part of the state vector (S_t) on the basis of which the DQN agent decides the action as expressed in Equation (1).

$$S_t = \{\textit{Available Edge Resources, Task Characteristics Cloud Resources, Energy Usage, Latency Constraints, SLA Requirements}\} \quad (1)$$

It provides enough information for the agent to compute the allocation of each task (either to the edge or to the cloud) such that the SLA for latency and energy consumption is satisfied.

The DQN agent with a task scheduling decision decides whether to process the task at the edge device side or to transfer it to the cloud. It chooses the one that maximizes energy saving and minimizes the latency for each task under its SLA. denoted the action space for this decision as A_t, where each action corresponds to a decision on how to allocate tasks. This is formally defined as an action space as in Equation (2).

$$A_t = \{\textit{Edge Processing, Cloud Offloading}\} \quad (2)$$

When the task is assigned, it is either executed, transmitted to the cloud, or processed on the edge. For edge-processed tasks, it gets executed directly on the resources available. But if the task is outsourced to the cloud, it crosses the communication network [6]. In fact, since low-latency communication between gadgets on the edge and the cloud is critical, the communication network becomes an important component to understand and improve as well. can model the network with a delay function D_{trans}: which takes into account the amount of time to transfer a task from the edge to the cloud, as in Equation (3).

$$D_{trans} = \frac{TaskSize}{NetworkBandwidth} \quad (3)$$

Such a delay needs to be minimized so that the total system can satisfy the real-time condition of the IoT application. Therefore, the total amount of time spent locally

processing at the edge.T_{edge}, the task transmission time D_{trans}, and the cloud processing time T_{cloud}gives us the total time for completion of the task, as in Equation (4).

$$T_{total} = T_{edge} + D_{trans} + T_{cloud} \quad (4)$$

To do this efficiently, the system relies on the reward function that imposes penalties on energy budget excesses and latency violations, while providing positive rewards on task completions that satisfy their SLA requirements. The reward R_tFor each scheduling decision, therefore, can be expressed as in Equation (5).

$$R_t = -(\textit{Energy Penalty}) - (\textit{Latency Penalty}) + (\textit{Task Completion Reward}) \quad (5)$$

The energy and latency penalties are measured as the energy needed to handle the task (at the edge or in the cloud), and the opt-out latency corresponding to deadline exceed, respectively. SLAs are honored by saving the task completion reward for when the task has been processed within the required time and energy budgets.

This work proposes an IoT application deployment architecture to dynamically partition processing tasks to the cloud or edge based on real-time occupation of resources and characteristics of the workload. Halo offers the flexibility of operating according to the variable nature of the system to run the tasks without violating the latency requirements and energy constraints.

It allows for system scaling in terms of the number of IoT devices and tasks and is efficient and adaptable to varying device data rates, scale, and heterogeneous environments. This deployment offers a compelling choice for IoT applications in the edge-cloud computing scope due to not only its capability to optimize resource allocation and task processing but also to offer fast operation with strict SLA requirements.

3.3. Task Scheduling Framework

The task scheduling framework is the main logic element that performs an actual, real-time intelligent decision of the location and the approach of the task processing in the edge-cloud computing system. The objective of this method is to minimize the energy consumption and latency violation constraint during task distribution, and it guarantees efficient task execution in terms of the SLA. A task scheduler framework driven by a Deep Q-Network (DQN) agent that continually self-adjusts its policies with experience feedback on previous task scheduling over time.

The first step in the task scheduling process is the state representation. S_t, which contains the information about the current state of the system, such as the status of available resources, features of the task, and/or SLA violations. The state vector is defined in terms of parameters including local edge resources, task types, cloud resources, energy cost, and latency limits. The DQN agent takes this state vector and decides on the task allocation. It decides on the basis of optimizing a reward function, which balances energy efficiency with low latencies required by SLAs. A DQN agent learns continuously and updates its policy that dictates the mapping from the past experiences of scheduling tasks in the form of states and actions to the best possible mapping of tasks to decisions to make as the dynamics of the system change over time.

At each time step, the agent can take certain actions defined in the action space. A_t. The action is the task that is assigned to either the edge device or the one that is offloaded to the Cloud. Define the action space as two actions: process either offload to the cloud or at the edge (Equation 2). When a particular decision is made, invocation goes either to execution at the edge or offloaded to the cloud level for execution. DQN agent's action affects the next state the system will be in, which is captured in the forward state vector S_{t+1}, and is rewarded R_t based on the system's capability to perform the task.

At the core of the framework is a reward function that encourages good behaviors, including reduced energy consumption and compliance with task deadlines. The penalty (i.e., the penalty on both the energy consumption and the latency) and rsl (i.e., reward for the successful completion of the task with acceptable SLA) are jointly represented as a reward function. R_t (Equation 3).

The DQN agent keeps learning from the rewards obtained after every decision made in scheduling and consistently updates the policy. Learning is carried out through the traditional Q-learning algorithm, where the agent instantiates the Q-values for the state-action pairs and updates them in an effort to maximize the total reward over time. Update the Q-function using the Bellman equation (Equation 4).

Experience Replay Mechanism: To enhance the learning of the DQN agent, it uses an experience replay mechanism. This means that we save past transitions (state-action-reward-next state) in a replay buffer and sample from that buffer randomly during training to destroy correlations between consecutive points in the experience. This permits the agent to better learn and generalize in undetected states.

Another important novelty of this framework is decomposing action space pruning at the system level. Limit task scheduling to those actions that will likely occur based on resources, assignment urgency, and preferences, cutting down unnecessary actions. For instance, if the edge device is resource-constrained and the task requires high computational complexity, the task will more likely be offloaded to the cloud. This process of pruning condenses out

the decision-making complexity and guarantees the rationality and quickness of the agent's decision-making, especially for large-scale systems with many tasks and resources.

The task scheduler algorithm steps are as follows.

update the state vector S_tAccording to the system status (edge and cloud resources and characteristics of each task, as well as the SLA constraints).

It derived the state vector and fed it to the DQN agent, which predicts the best action to take from the action space. A_t (process at the edge or offload the task to the cloud).

The task is performed by the action that has been chosen, and then the environment moves to a new state. S_{t+1}.

The energy consumption, latency, and completion status decide the reward. $R_t$and this is further utilized to update the DQN agent policy.

For every task, this process is repeated, so the agent can learn to schedule more optimally with experience.

The proposed task scheduling framework schedules tasks between edge devices and cloud infrastructure while satisfying the energy and latency constraints. The DQN agent can be utilized in solving real-time IoT applications, as the usage of the DQN agent would make the system keep on learning and adapting itself to the changing system environment. It enables scalability, efficiency, and flexibility in resource-constrained environments such as IoT networks in the context of edge-cloud computing.

3.4. System Optimization

This framework also supports the optimization of the task scheduling system, which can be worked on by minimizing energy and latency to improve the scalability/efficiency of task allocation. This combines several functionalities to improve the edge-cloud performance while adhering to the SLA requirements for tasks. These optimizations are key to creating the system: (1) action space pruning, (2) balancing resource allocation, and (3) adaptive task scheduling.

Propose a method to prune the action space to simplify the decision process of the DQN agent in this paper. In a system where there are a lot of resources and tasks, the action space can be huge, which will slow down learning and decision-making. Unnecessary actions are pruned according to the available resources and the urgency of the task to solve this. Thus, for instance, if an edge device is resource-constrained, and a task is very compute-heavy, then most likely that task must be offloaded to the cloud. Such pruning limits the search space for the agent and allows for quicker and more efficient decision-making. It can formulate the action space pruning as in Equation (6).

$$A_t^{pruned} = \{\text{Reduced Set of Actions}\} \quad (6)$$

These actions depend on the current state of the system, such as the resource availability at the edge and cloud, and the deadline characterizing the task. Pruning can make the task scheduling process faster and more efficient because it reduces computation overhead.

Resource allocation balance is another key area of optimization. It balances resources across Edge devices and the cloud infrastructure, such as CPU, memory, bandwidth, and storage. In this process, task resource allocation is adjusted dynamically based on real-time task requirements. This workload distribution between edge and cloud prevents resource bottlenecks and keeps task completion times within specified limits. Combining edge and cloud resources can be modeled by Equation (7).

$$R_{allocation} = f(\text{Edge Resources}, \text{Cloud Resources}, \text{Task Requirements}) \quad (7)$$

Where f is a function that distributes resources according to the complexity and urgency of tasks, which seeks to achieve the minimum waste of resources while maintaining the maximum throughput of the system.

Adaptive task scheduling is an optimization either for reducing the overall task execution time or the waiting time of each job by dynamically choosing the task scheduling methods considering the current environment conditions. It learns the best scheduling policy online by adapting to the workloads, network, and resource availability in real time. At the heart of this process is the DQN agent that keeps improving the scheduling policies taken with reference to the history of previous task execution and system states. The system can be described as adaptive in nature, as expressed in Equation (8).

$$S_t \rightarrow S_{t+1} \quad (8)$$

Here, S_t is the system state at the time, S_{t+1} This is the new execution state once the task is scheduled and executed. It allows the system to consider real-time constraints when adapting to different scenarios, which guarantees optimal resource consumption and satisfaction of task requirements.

Also, energy-aware scheduling is an important contributor to providing an optimized system. By determining what to process at the edge or offload to the cloud, it minimizes energy consumption, allowing for more efficient decisions. Energy-intensive tasks are more likely to be offloaded to the cloud, where there are more computational resources available. On the flip side, edge processes a lower-computational-power task that helps save energy. incorporate the energy penalty into the reward function, encouraging the agent to schedule pods in an energy-efficient way. The

equation of energy optimization can be expressed as in Equation (9).

$$E_{total} = \sum_{i-1}^{n} E_i \quad (9)$$

Where E_i is the total energy consumed while executing each task at the edge or in the cloud, and n is the number of tasks. Thus, in essence, the objective now is to minimize. E_{total}at low latency while still conforming to the SLA latencies.

Latency optimization is all about being able to process every task faster. It also reduces the communication delay between the edge devices and the cloud and minimizes the execution time of the task, depending on the availability of resources. Depending on the latency requirement, tasks are either assigned to edge devices or offloaded to the cloud. When a task takes longer than the configured latency threshold, latency penalties are imposed, which encourages the system to expedite task processing and reduce backlog. Here is the total latency L_{total} for each task as in Equation (10).

$$L_{total}=L_{edge}+L_{cloud}+D_{trans} \quad (10)$$

To where $L_{edge}$is the processing time at the edge, $L_{cloud}$is the processing time at the cloud, and $D_{trans}$is the task transmission time over the network. The goal of the system is to minimize. L_{total}while satisfying a deadline constraint of each task.

All these optimizations combine to maintain energy-efficient, low-latency task scheduling while ensuring scalability and adaptiveness. Time-aware Workload Management Using Dynamic Workload Adaptive Techniques—By leveraging Pruning, resource balancing, energy-aware scheduling, and Adaptive task scheduling, the system can balance the workload dynamically while maintaining the IoT applications' real-time needs in edge-cloud scenarios. As the DQN agent is capable of continuous learning and policy updating, the system will also be able to increase its performance and efficiency as more tasks are processed.

3.5. Algorithmic Representation

This section presents the algorithmic representation of the proposed energy-aware and latency-knowledgeable task scheduling framework. It formalizes how the DQN-based scheduler makes decisions, detailing how system states, actions, rewards, and learning updates interact to enable adaptive and efficient task allocation between edge and cloud resources.

Algorithm 1. Energy-Aware and Latency-Aware Task Scheduling

Input:
Task queue Q with task characteristics (size, complexity, latency, energy profile)
System state S_t with edge and cloud resources, SLA constraints

Output:
Task allocation (edge processing or cloud offloading)
Optimized energy consumption and latency

Steps:

1: Initialize:
Set initial system state S_0 based on available resources and task queue Q.
Initialize the DQN agent with random weights and action space A_t.

2: For each task i in Q:
Update system state S_t with edge and cloud resource availability and task details.

3: Select action A_t:
The DQN agent chooses an action (edge or cloud) based on S_t.

4: Execute task:
If at the edge, compute T_{edge}; if at cloud, compute D_{trans} and T_{cloud}.

5: Calculate reward R_t:
Compute energy E_i and latency L_{total}.
Apply reward/penalty based on SLA compliance:

R_t= −Energy Penalty−Latency Penalty+ Completion Reward

6: Store experience:
Store (S_t,A_t,R_t,S_{t+1}) in the experience replay buffer.

7: Update DQN:
Sample mini-batch from replay buffer.
Update Q-values using the Bellman equation:

$Q^*(s_t,a_t)=R_t+\gamma \max_a Q(s_{t+1}, \text{a})$

8: Repeat until all tasks are scheduled.

Algorithm 1 presents an energy-aware and latency-aware task scheduling strategy for IoT applications operating in an edge–cloud computing environment. The algorithm employs a DQN agent to dynamically decide whether to carry out an impending assignment at the periphery or offload it to the cloud. Initially, the system state is defined using available edge and cloud resources, task characteristics, and SLA constraints. For each task, the current system state is updated and fed to the DQN agent, which selects an appropriate scheduling action. Task execution is then carried out either locally at the edge or remotely in the cloud, accounting for processing time and transmission delay. After execution, the algorithm evaluates energy consumption and latency to compute a reward that reflects SLA compliance. This reward guides the DQN agent's learning through experience replay and Q-value updates. By continuously learning from past scheduling decisions, the algorithm adapts to changing workloads and resource conditions, enabling efficient task allocation with minimized energy usage and reduced latency in dynamic edge–cloud IoT environments.

3.6. Deployment and Evaluation

The system deployment and validation of the task-scheduling framework in the edge-cloud IoT environment are conducted to ensure the system's real-time operational performance. This part of the paper explains the deployment of the IoT application in a simulated edge-cloud environment and its performance evaluation; key data, including energy usage, latency, and task completion rate, are measured.

The first step of the deployment procedure, namely the simulation setting, consists of setting up an IoT application and several IoT tasks with different computational requirements, deadlines, and energy usage characteristics, as the tasks are created based on the nature of real-time data, which can be derived from sensor data streams, video streams, or user input. These tasks are modeled to reflect the heterogeneous nature of IoT applications, ranging from simple (low computational complexity) to complex (high computational complexity). The simulation environment consists of resource-constrained edge devices and a cloud infrastructure capable of handling more processing-intensive tasks.

The DQN agent's decision specifies which task should be scheduled on the edge device or the cloud infrastructure. The agent learns over time by recording its experience when performing tasks and the system state, improving its policies by adapting them based on what has previously occurred, thus making even better decisions when determining which agents should perform which tasks in the future. The state vector S_t It contains numbers corresponding to the available resources, the characteristics of the tasks, and SLA constraints (Equation 1). Detailed Description: The action chosen by the DQN agent depends on which reduction of energy and latency minimizes the implemented SLA requirements.

The performance metrics are then gathered after processing the tasks and evaluated. Energy efficiency, latency, and task success rate are the main metrics for evaluating the system performance. These metrics are defined as,

Total energy consumed by edge devices and cloud infrastructure when executing the same task: Energy efficiency. It depends on the time taken, resource utilization, and task complexity. Quoting the total energy consumption E_{total} as in Equation (11).

$$E_{total} = \sum_{i-1}^{n} E_i \tag{11}$$

Where E_iExecution is the n, which is the total number of tasks completed, and each job uses energy during execution.

Latency: The time taken for edge processing and cloud processing of each task that needs to be performed. Thus, for the total latency L_{total}Each task is calculated as in Equation (12).

$$L_{total} = L_{edge} + L_{cloud} + D_{trans} \tag{12}$$

L_{edge} is processing time at the edge; $L_{cloud}$is processing time at the cloud; and $D_{trans}$is the transmission time between edge devices and the cloud.

Task Success Rate: This metric measures the number of tasks accomplished under the constraints defined by its SLA. A task is successful if it is finished before the task SLA and within the energy budget specified by the SLA. Calculate the success rate as in Equation (13).

$$Success\ Rate = \frac{Number\ of\ Successful\ Tasks}{Total\ Number\ of\ Tasks} \times 100 \tag{13}$$

Such performance metrics are constantly tracked and are used to evaluate the system's overall efficacy. Then evaluate the scheduling system on a set of metrics that compares its performance against existing baseline models in terms of energy or latency, or existing scheduling methods (note that many of those methods do not incorporate energy or latency constraints in the task allocation process).

The evaluation methodology is based on cross-validation to demonstrate the performance of a system given different task distributions and for different levels of system load. The different time series tasks are separated into training and testing sets (for CV), in which the model is trained on one subset of tasks and evaluated on the other.

So that the model generalizes unseen data and overfitting is avoided. It also contains ablations to evaluate the various

parts of the system (energy-aware scheduling, latency-aware scheduling, and experience replay).

Present the results in terms of energy efficiency, latency, and test success rate compared to other scheduling techniques. The objective is to demonstrate that the proposed system performs well in terms of energy, latency, and task success rate compared to conventional methods.

Eventually, to close the process of deployment and evaluation, a brief discussion of system scalability and adaptability is presented. Evaluate this system across a range of IoT applications and for different task sizes to verify that it scales with the availability of devices/tasks. In real-time IoT-based applications, edge-cloud system workloads vary, and the DQN agent's adaptive nature makes the system efficient and effective as workloads change.

4. Experimental Results

In this section, discuss the experimental evaluation of the proposed framework for energy- and latency-aware scheduling of IoT applications deployed in an edge–cloud environment. In this section, design experiments are conducted to validate the solution using the proposed approach. target latency, energy consumption, SLA, and scalability across various workload and system scenarios.

4.1. Experimental Setup

Evaluate the proposed method in a synthetic edge–cloud computing environment that realistically resembles practical IoT deployment scenarios. The higher-level simulation models a distributed system of edge nodes connected to a common cloud via an existing communication network. It is a discrete-time scheduler in which IoT tasks are generated, dynamically scheduled, and executed depending on the system's ongoing state. To achieve real-time scheduling under varying workload conditions, implement the proposed DQN-based job scheduling system within the model.

The edge–cloud infrastructure consists of a chain of interworking heterogeneous edge nodes with limited computational resources and a cloud layer with relatively high processing power. The edge nodes have limited budgets for CPU, memory, and energy, simulating realistic IoT gateways or micro data centers. Scalable compute resources are provided on cloud servers capable of handling computation-intensive tasks. Explicitly model network latency between the edge and the cloud using random latency values to accurately account for transmission delays during task offloading, as network conditions are always changing.

The IoT application workload is intended to mimic latency-sensitive, energy-constrained applications typically seen in bright environments, supporting real-time monitoring and data analytics. Different tasks are defined according to their computation, data arrival, execution time, and energy consumption. To accommodate both lightweight and heavyweight workloads, the tasks vary in complexity, enabling us to evaluate the scheduling framework across different implementation scenarios.

To account for the stochasticity inherent in IoT systems, task arrivals follow different distributions. Low and high arrival rates are used to subject the system to light, moderate, and heavy workloads. Every task has an SLA (SERVICE-LEVEL AGREEMENT) that defines latency requirements and tolerable energy consumption. These SLA profiles help the scheduler plan tasks based on the requirements set by an application during execution and scheduling.

Perform all experiments on a multi-core workstation with sufficient memory to run large-scale simulations. Python is used to implement the scheduling framework, while the deep learning features are written using a standard neural network framework. On a standardized OS, there is thus uniformity and reproducibility of design in simulations and learning across all operations and experiments.

4.2. IoT Application Deployment Scenario

The IoT application deployment scenario mimics a latency-sensitive and energy-constrained application prevalent in realistic edge–cloud environments, such as real-time monitoring or event-driven analytics. It produces a stream of real-time, data-saturated tasks that need to be processed immediately for the system to remain operational. Each task has strict latency requirements and energy constraints, and a good scheduling scheme is critical to ensuring application performance and reliability.

Application tasks, depending on computation time, urgency, and resource availability, are dynamically allocated to either edge nodes or cloud servers. Lightweight, Delay-sensitive assignments are executed at the edge because they require very low communication latency and minimal energy overhead. In contrast, intensive tasks are offloaded to the cloud because it provides sufficient processing resources. Such dynamic task mapping allows the optimization of distributed resources with respect to energy use and response time.

The evaluation presents different levels of workload intensity, light, moderate, and heavy workloads, to assess system behavior under various operational conditions. Light workloads are characterized by sparse task arrivals and low system utilization, while moderate workloads reflect typical operating conditions. Workload types: Heavy workloads emulate a bursty or high-demand situation, putting stress on the scheduling framework and testing its ability to withstand congestion and resource contention.

An execution pipeline follows a sequencing flow in near-real time, in which tasks generated at IoT devices are

evaluated by the scheduling framework and then executed at the chosen processing layer. The execution of tasks is constantly monitored for compliance with latency and energy constraints. In contrast, feedback from completed tasks is used to improve decision-making regarding the scheduling of future tasks—the execution of this pipeline in real end-to-end IoT applications in edge-cloud deployments.

4.3. Performance Evaluation Metrics

Use a set of well-defined metrics that account for system efficiency and potential IoT apps' service quality to evaluate the performance of the proposed task scheduling framework. Choose a subset of these metrics since the focus is on the main goals, such as minimizing latency, minimizing energy consumption, and ensuring SLO compliance when workloads are dynamic.

Average task latency is how long it takes to finish a task, from when it is created on the IoT device to when the final output is delivered. It incorporates processing time and communication delay (for on/offloading tasks to the cloud). Reduced average latency implies improved responsiveness during peak events and is essential for real-time IoT applications with rigid timing constraints.

Utilization of energy is the total energy used to perform the task across both edge nodes and cloud servers. It includes both computation energy and communication energy overhead. This is particularly critical when edge devices have limited power, as it can directly affect the sustainability of a large-scale IoT deployment.

SLA satisfaction ratio: The ratio of the tasks that are executed within their latency and energy constraints. A higher SLA satisfaction ratio indicates that the scheduling framework can prioritize tasks based on their service requirements and guarantee quality of service across different workload scenarios. A task completion rate measures how many tasks are completed correctly, without any deadline breaches or system crashes. This translates to the stability of the scheduling framework in handling different workloads and resource constraints, as reflected by this metric.

The overall efficiency of the edge–cloud infrastructure includes system throughput and resource utilization. In the metric of throughput, how many tasks can be completed within a unit time? Resource utilization indicates how efficiently the processing power at the periphery and in the cloud is utilized. Increased throughput and balanced resource utilization reflect improved scalability and better system operations.

4.4. Baseline Scheduling Algorithms

Furthermore, the proposed energy- and latency-aware task scheduling framework is evaluated and compared against various energy- and latency-based baseline task scheduling algorithms, given the level of information about user demand and application requirements in edge–cloud and IoT computing contexts. These baseline methods include both heuristic and learning-based methods that do not explicitly minimize the energy–latency trade-off but instead seek to optimize performance.

FIFO Scheduling Algorithm — It is the simplest; it implements the First-In-First-Out (FIFO) mechanism and does not consider task urgency, resource constraints, or service-level constraints. Although FIFO is simple to implement, it tends to result in higher latency and lower throughput under dynamic or high-load conditions. It is therefore not appropriate for real-time IoT applications.

In Round Robin scheduling, tasks are assigned a time slice for resource use and are cycled through, ensuring fair resource allocation. While it effectively avoids starvation and balances load, it does not account for task deadlines, energy consumption, or varying computational needs, making it suitable only for latency- and energy-agnostic settings.

EDF — This algorithm schedules tasks according to their deadlines, executing the assignment with the earliest due date first. Finally, EDF is an excellent choice for complex real-time systems with strict timing requirements. Still, it does not account for energy efficiency or perform system-level resource optimization. Consequently, EDF could satisfy the latency constraints, but it would require higher energy consumption than other proposals.

Non-learning heuristic offloading strategy that decides to offload tasks based on predefined rules: task size threshold or static resource utilization cap. Such an approach is static and cannot adapt to changing workload patterns and network conditions, leading to poor performance in dynamic edge–cloud environments.

A DRL-based scheduler without energy–latency awareness serves as a learning-based baseline. This method can adapt to changes in system dynamics and make better scheduling decisions over time, but it neither directly optimizes energy usage nor the latency constraints. This illustrates the advantage of learning energy- and latency-aware objectives.

4.5. Comparative Performance Analysis

Performance Comparison Evaluation: The evaluation compares the proposed energy-aware and latency-aware task scheduling framework against various baseline scheduling algorithms under varying workload conditions. The comparison evaluates performance in terms of delay and energy usage, SLA satisfaction, task success rate, and overall system efficiency to quantify the benefits of the proposed approach in edge–cloud IoT scenarios.

Latency metrics are evaluated for light, moderate, and heavy workloads. Under moderate and heavy workloads, where congestion dominates, the proposed framework's average latency is consistently lower. Under heavy workloads, Latency spikes sharply in traditional FIFO and Round Robin schedulers, while EDF shows lower Latency but inferior energy efficiency. show that the DRL-based scheduler without energy–latency awareness achieves lower latency than heuristics, but is still worse than the proposed method, as it lacks joint optimization.

From the perspective of total energy consumption, the proposed framework strikes an intelligent balance between edge execution and cloud offloading, resulting in significantly lower energy use. Although EDF achieves the minimum latency, it causes the maximum energy outage due to its aggressive scheduling. FIFO and Round Robin also draw more energy when the load is heavy. The DRL baseline's energy efficiency is neither good nor bad, but the proposed scheduler consistently outperforms it by explicitly incorporating energy-aware decisions into the RL model.

The proposed approach is further shown to be robust with respect to SLA satisfaction and task success rates. The designed framework meets strict latency and energy constraints by providing the highest SLA satisfaction ratio and task success rates across all workload levels. At high load, baseline heuristics experience extensive SLA violations, while the DRL baseline maintains SLA targets better but cannot catch up to the proposed method.

Proportional resource allocation and also system-wide throughput (utilization) indicate the scalability of the proposed scheduler. The proposed framework achieves high throughput and stable performance under stress by dynamically accommodating workload variations. Quantitative Comparison of Scheduling Algorithms at Moderate Workload: Table 3

Table 3. Comparative Performance Analysis of Scheduling Algorithms

Scheduling Method	Avg. Latency (ms) ↓	Energy Consumption (J) ↓	SLA Satisfaction (%) ↑	Task Success Rate (%) ↑	Throughput (tasks/s) ↑
FIFO	185.6	0.48	71.2	78.4	42
Round Robin	156.3	0.44	76.9	82.1	47
EDF	112.8	0.53	84.6	88.3	51
Heuristic Offloading	139.4	0.39	81.2	85.7	49
DRL (No Energy–Latency)	101.7	0.34	89.5	91.2	56
Proposed Method	78.9	0.27	95.8	96.4	63

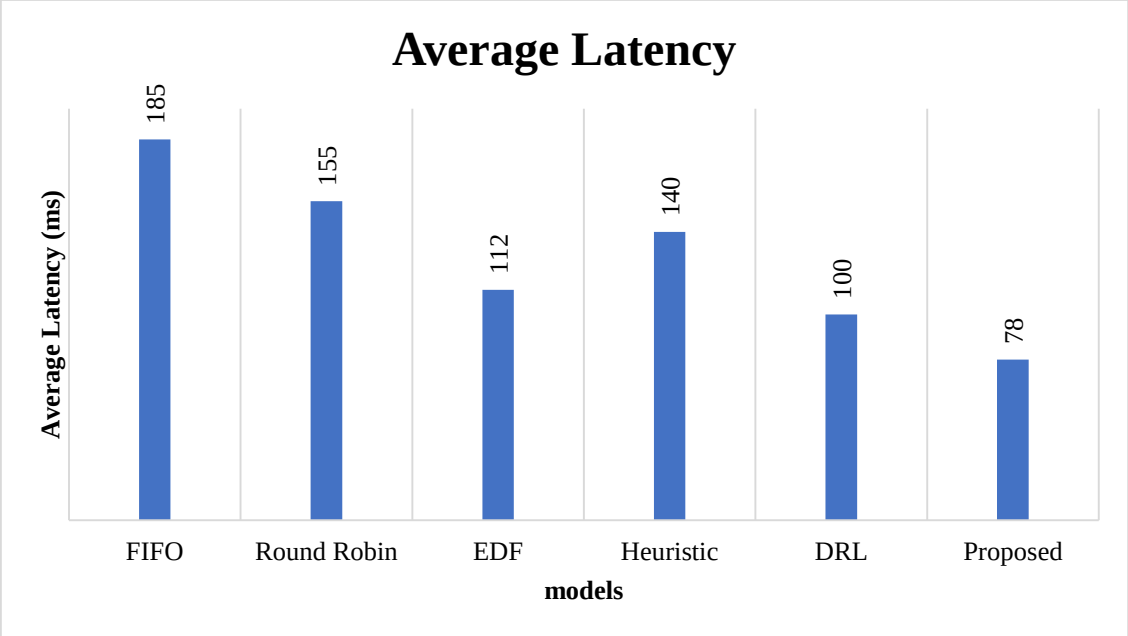

Fig. 3 Average Task Latency Comparison across Scheduling

In Figure 3, compare the average task latency across the various scheduling algorithms in the edge–cloud IoT environment. FIFO and Round Robin have the highest latency because they do not prioritize or adjust to tasks. Deadline EDF (not efficient across the system, but supports lower latency), Learning-based Scheduling Further Degrades Latency, Adapting without Energy–Latency Awareness. Evaluation results demonstrate the effectiveness of the proposed energy- and latency-aware scheduling framework in minimising latency through dynamic selection of optimal execution locations, and show that it can support real-time IoT applications under different operating conditions.

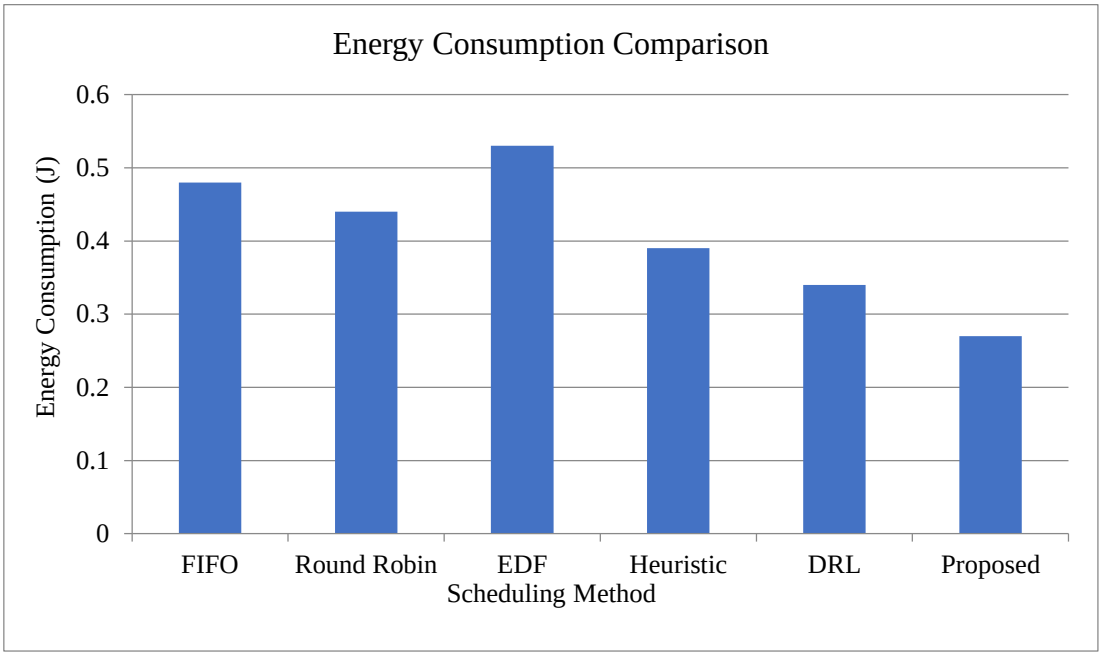

Fig. 4 Energy Consumption Comparison of Scheduling Algorithms

Figure 4 Depicts the overall energy usage for the different task scheduling approaches. Due to driving deadlines, EDF has the highest measured energy usage. In addition, FIFO and Round Robin will also lead to higher energy consumption because resources are not fully utilized. Compared with a learning-based scheduler without explicit energy–latency awareness, adaptive scheduling can achieve moderate energy savings. The results validate the success of the suggested framework in jointly optimizing execution decisions for edge and cloud resources to minimize energy consumption, indicating its suitability for energy-concerned IoT environments and sustainable edge-cloud deployments.

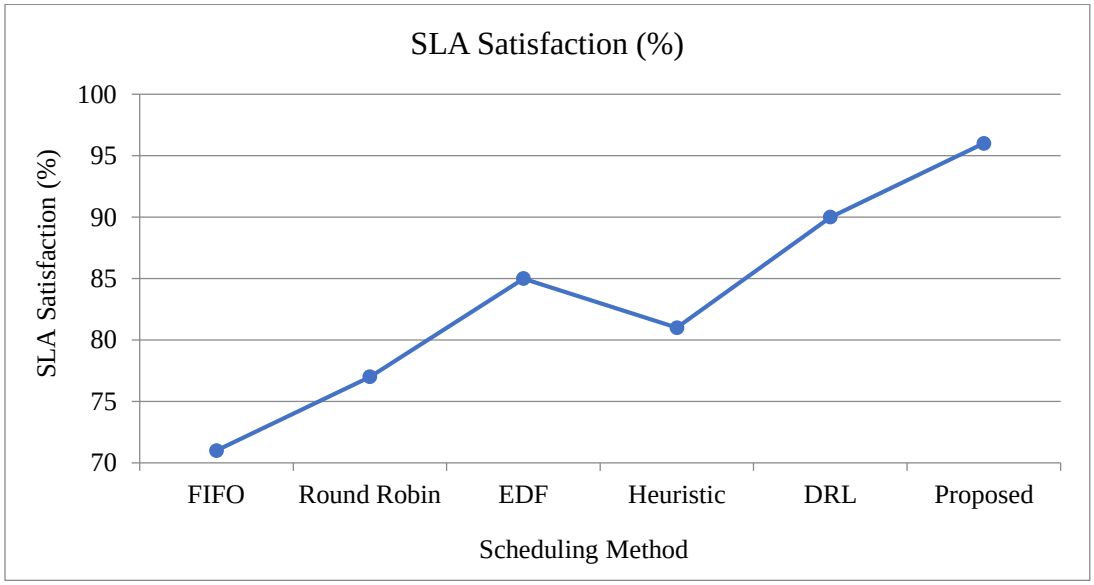

Fig. 5 SLA Satisfaction Ratio Comparisons

The SLA satisfaction ratio comparison is shown in Figure 5 for the different task scheduling approaches.

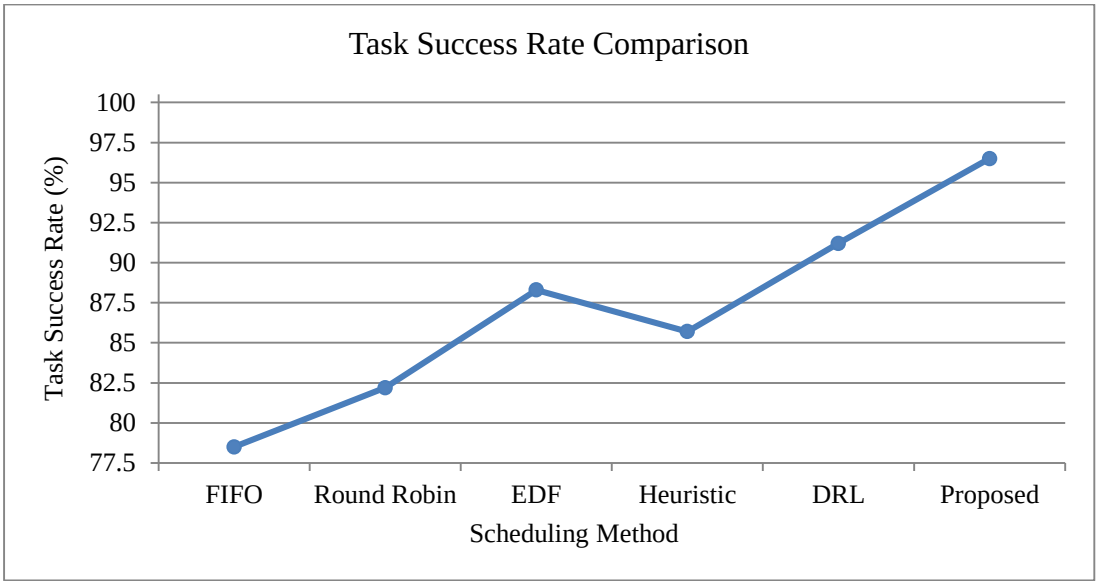

Fig. 6 Task Success Rate Comparisons

In Figure 6, a comparison of task success rates across the different scheduling algorithms is shown. Workloads become more demanding, and FIFO and Round Robin fail to meet their requests, leading to more deadline violations and resource contention. EDF makes tasks finish faster, but it does not provide any robustness during energy-limited conditions. Leverage a learning-based approach to the scheduler to increase robustness to the system's changing nature. By jointly taking into account these constraints, exposed to energy limits, latency requirements, and availability of resources, the proposed framework accomplishes the best task success rate, thus ensuring reliable task execution in time for setting up real-time IoT applications deployed over edge–cloud infrastructures.

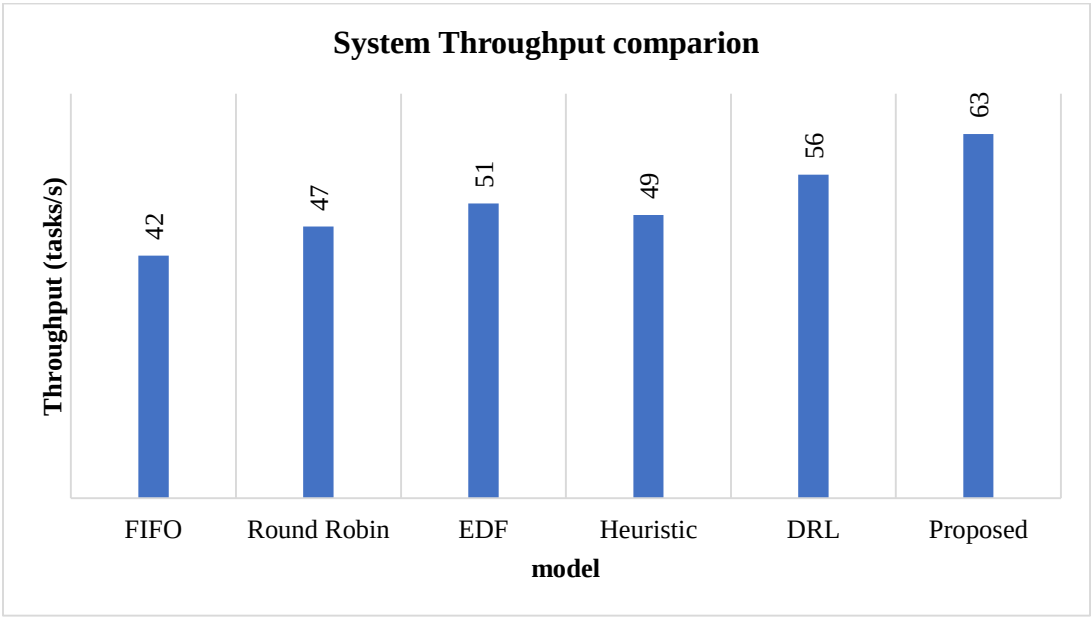

Fig. 7 System Throughput Comparisons

The throughput, expressed as the number of tasks completed per second across different scheduling methods, is illustrated in Figure 7. Traditional heuristics have limited throughput because resources are not effectively utilized. Throughput improves, but EDF and heuristic offloading fail under high workloads. The learning-based scheduler further

improves throughput through adaptive task placement. Our proposed framework achieves higher throughput through the optimal allocation of edge and cloud execution, showcasing excellent communication scalability and efficient resource utilisation in distributed computing systems forosed framework achieves a higher throughput through the optimal allocation of edge and cloud execution, showcasing excellent communication scalability and efficient resource usage of the distributed computing systems in edge-cloud IoT environments.

4.6. Impact of Energy-Aware and Latency-Aware Scheduling

In this section, first evaluate the effect of introducing energy-aware decision-making in the task scheduling framework. The impact of latency-aware prioritization, and finally, analyze the individual effects and their combination. Apart from traditional schedulers designed to optimize a single optimization goal, such as energy consumption or task latency, the proposed approach enables balanced scheduling decisions by jointly optimizing task latency and energy consumption, satisfying the diverse requirements of edge–cloud IoT, which are inherently dynamic.

Energy-aware decision-making can substantially reduce overall system-level power consumption by dynamically selecting execution locations to minimize unnecessary computation and communication overhead. Light tasks are processed at edge nodes preferentially, and the energy-intensive tasks are offloaded to the cloud when it is beneficial. It also avoids overusing the cloud and overloading edge devices, which further increases energy efficiency. Decision-making can substantially decrease overall system-level power usage by dynamically selecting execution locations that reduce unnecessary energy compared to static and SLA-only schedulers.

Latency-aware prioritization ensures that tasks with strict timing constraints are executed immediately by prioritizing low-latency execution paths. The scheduler adapts to changes in network conditions, processing delays, and task priorities to minimize average task latency and dynamically minimize deadline violations. And this works exceptionally well for mid-to-high workloads, where traditional schedulers fail to remain responsive.

In fact, the learning-based policy explicitly balances energy consumption with latency, enabling the scheduler to adapt its decisions to the actual system situation. Rather than explore excessive latency for high energy consumption, or excessive energy consumption for low latency, a minimum-cost [29] or any other heuristic framework reaches an optimum, resulting in workloads across varying model quality/runtime performance.

Table 4. Impact Analysis of Energy-Aware and Latency-Aware Scheduling

Scheduling Strategy	Avg. Latency (ms) ↓	Energy Consumption (J) ↓	SLA Satisfaction (%) ↑	Key Observation
Static Heuristic Scheduling	139.4	0.39	81.2	No adaptation to workload changes
SLA-Only DRL Scheduling	101.7	0.34	89.5	Improves latency, but energy is not optimized.
Energy-Aware Only Scheduling	118.6	0.30	87.1	Reduces energy but moderate delay increase
Latency-Aware Only Scheduling	92.4	0.41	90.3	Low latency with higher energy cost
Energy–Latency-Aware (Proposed)	78.9	0.27	95.8	Balanced and optimal performance

In Table 4, further compare energy–latency–aware approach against static heuristics and SLA–only schedulers, experimentally demonstrating its advantages in lower energy usage, lower latency, higher SLA satisfaction, and higher task success rates. It shows the necessity of jointly optimizing edge–cloud networks to deploy IoT systems in practice effectively.

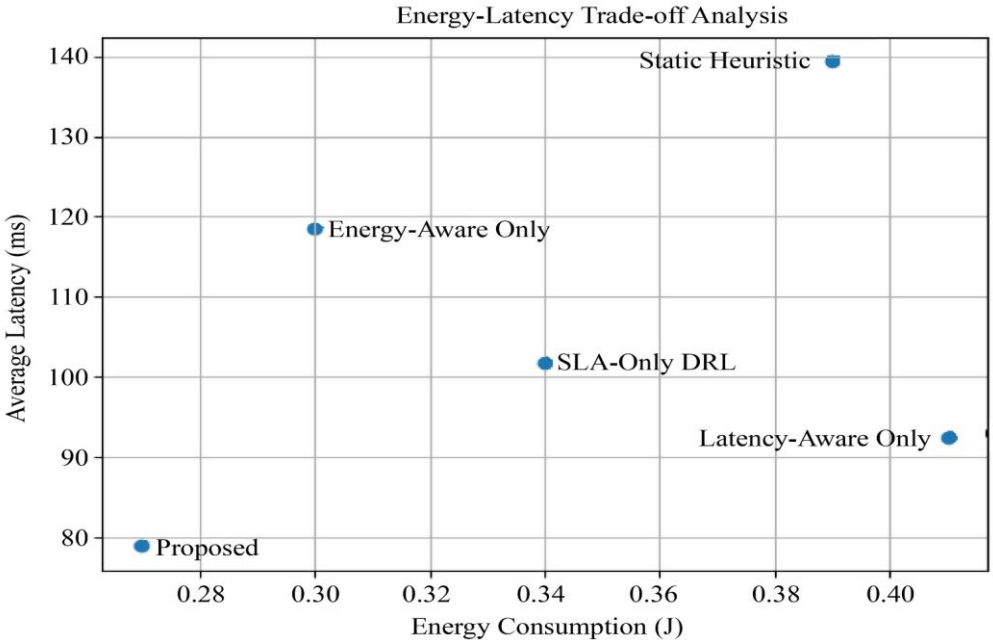

Fig. 8 Energy–Latency Trade-off Analysis of Scheduling Strategies

Scheduling strategies to minimize average task latency under different energy constraints are depicted in Figure 8. Static and SLA-only methods cannot find the right trade-off between both goals and either produce unsatisfactory latencies or consume more energy. Single-objective schedulers optimize one metric at the cost of the other.

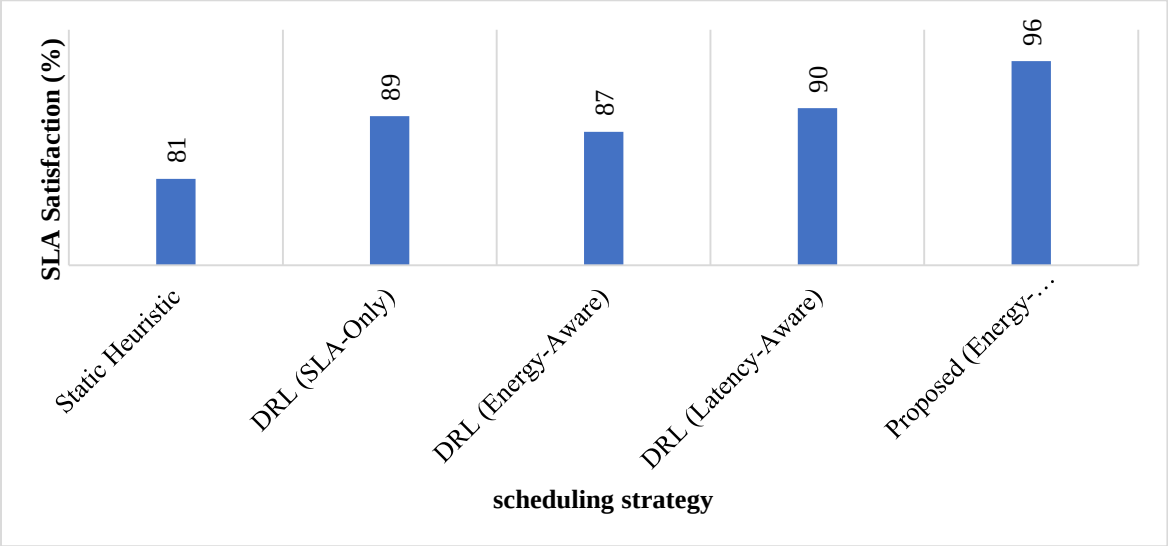

Fig. 9 Impact of Scheduling Strategies on SLA Satisfaction

To this end, propose an energy–latency-aware framework that provides the best trade-off between energy and task latency. This illustrates the benefit of jointly optimizing task scheduling to sustain. Figure 9: Comparison of SLA satisfaction ratio with different scheduling strategies. Static heuristic scheduling has the least compliance among all cases because it cannot adjust its workload distribution. SLA-only and single-objective schedulers improve satisfaction while imposing limitations due to partial optimization. The smart city infrastructure comprises various resource-critical IoT devices, each contributing to the quality of services in innovative city applications.

These results validate that the use of multi-objective learning-based scheduling tremendously improves service reliability in edge–cloud IoT systems and provides responsiveness in these environments.

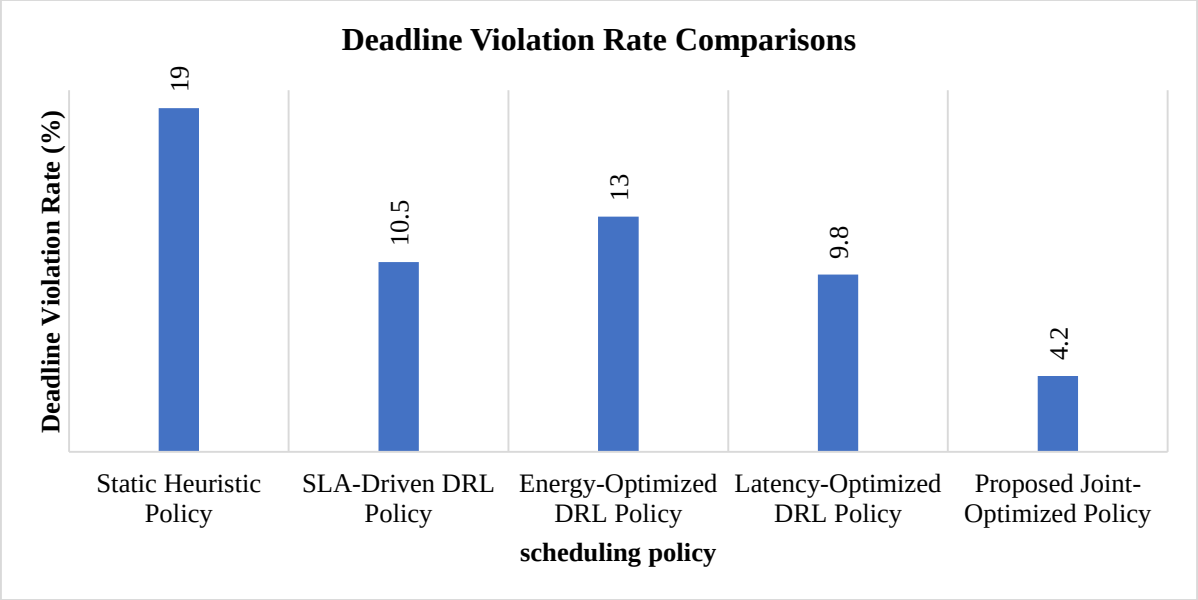

Fig. 10 Task Deadline Violation Rate Comparisons.

The Violation Rates of Task Deadlines for Different Scheduling Strategies are shown in Figure 10. Static heuristic scheduling has the highest violation rate because of its inflexible decisions. Schedulers with a single objective alleviate violations but remain susceptible to workload fluctuations. By making adaptive scheduling decisions based on system state, task urgency, and resource availability, the energy–latency-aware framework proposes to achieve the lowest deadline violation rate. It reveals that the time reflects the effectiveness of this framework, particularly in real-time IoT applications in edge–cloud environments.

4.7. Learning Convergence and Stability Analysis

In this section, analyze the cadence of learning, robustness, and adaptability of the proposed scheduling framework, which considers both energy and latency. Because the proposed scheduler is based on DRL, it is essential to demonstrate the learning's convergence and stability, as well as its dynamic adaptation to workload changes in the edge–cloud IoT environment.

By inspecting the cumulative reward evolution across training episodes, we can also analyze the behavior of reward convergence. At the start, the reward has high variance as the agent explores the action space and learns the effects of scheduling decisions. Over the course of training, the reward curve levels out and approaches a steadier state at higher values, implying that the agent has learned an effective scheduling policy that minimizes energy and latency while satisfying SLA constraints. This convergence validates the learned efficiency and robustness of the proposed framework.

Table 5. Learning Convergence and Stability Evaluation

Evaluation Aspect	Observed Behaviour	Quantitative Indicator
Reward Convergence	Stable convergence after training	Average reward stabilizes after ~600 episodes
Reward Variance	Reduced over time	Variance drops by ~65% post-convergence
Policy Stability	Consistent action selection	<5% action oscillation after convergence
Adaptation to Workload Change	Rapid re-convergence	New steady state reached within ~80 episodes
Long-Term Learning Robustness	No performance degradation	Stable rewards over extended evaluation

The temporal stability of the policy is evaluated by observing the consistency of scheduling decisions after convergence. Post-learning, the agent shows strong action-selection tendencies without oscillation, indicating that the learned policy is stable and exhibits no evidence of policy flips from one action to another (i.e., policy instability). Essentially, do not care (too) much about the scheduling behavior in proposed algorithms, because we do care

(absolutely) about stability in the real world, where unpredictable scheduling behavior will degrade system performance and reliability.

Further demonstrate the adaptability of the scheduling framework by introducing dynamic workload changes (i.e., sudden increases in task arrival rates and shifts in task characteristics). Table 5 shows how the proposed scheduler rapidly adapts its decisions to these changes, efficiently sampling different actions for a few time steps, then returning to a new optimal policy. This adaptive characteristic demonstrates the framework's capacity to maintain performance in the face of non-stationary conditions, achieving persistent performance regardless of system context.

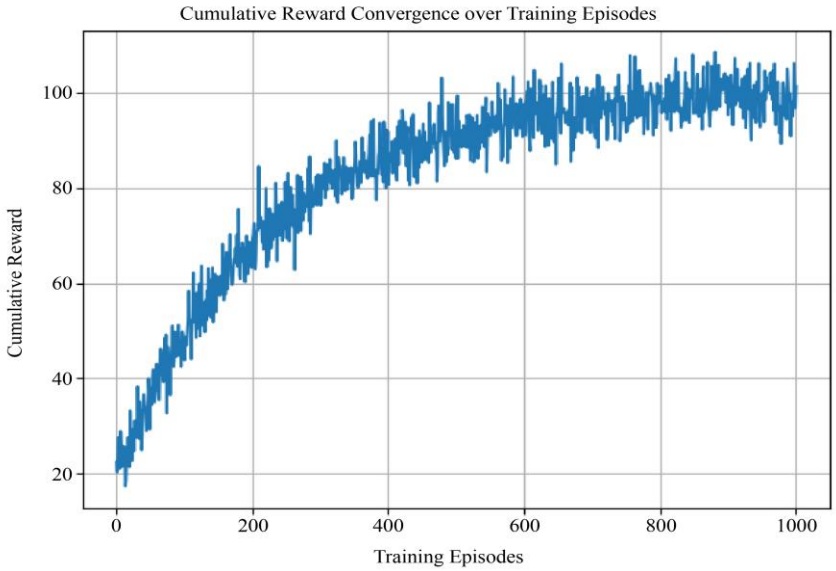

Fig. 11 Cumulative Reward Convergence over Training Episodes.

In Figure 11, the proposed energy-aware and latency-aware scheduling framework converges and learns to remain consistent with the previous learning behavior. In early training episodes, cumulative rewards vary widely due to exploration and policy adaptation. During training, the reward variance decreases, and the curve smoothens, implying convergence towards an optimal scheduling policy. From the results, we can see that the cumulative rewards increase, then plateau, confirming that the DRL agent eventually learns to balance energy consumption and latency while satisfying SLA constraints. This convergence behavior showcases the feasibility of learning that provides reliability to the learning process and has potential for deployment in dynamic edge-cloud IoT environments.

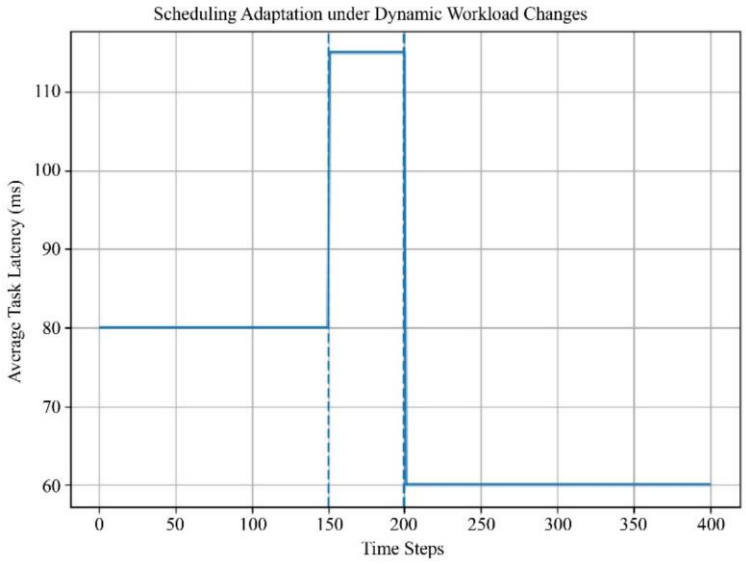

Fig. 12 Policy Stability based on Action Selection Frequency

Action selection frequencies from the learned scheduling policy across different training phases are shown in Figure 12 to assess the policy's stability. In the early stages of training, the agent will make many exploratory edge execution decisions. With the evolution of learning, action distribution has become dispersed and stabilized. In the end, the ratios of edge execution and cloud offloading stabilize, indicating a policy with minor fluctuations after convergence. It guarantees predictable scheduling behavior and stable overall system performance, which are extremely important for the applicability of reinforcement learning-based schedulers in real-world edge-cloud IoT systems.

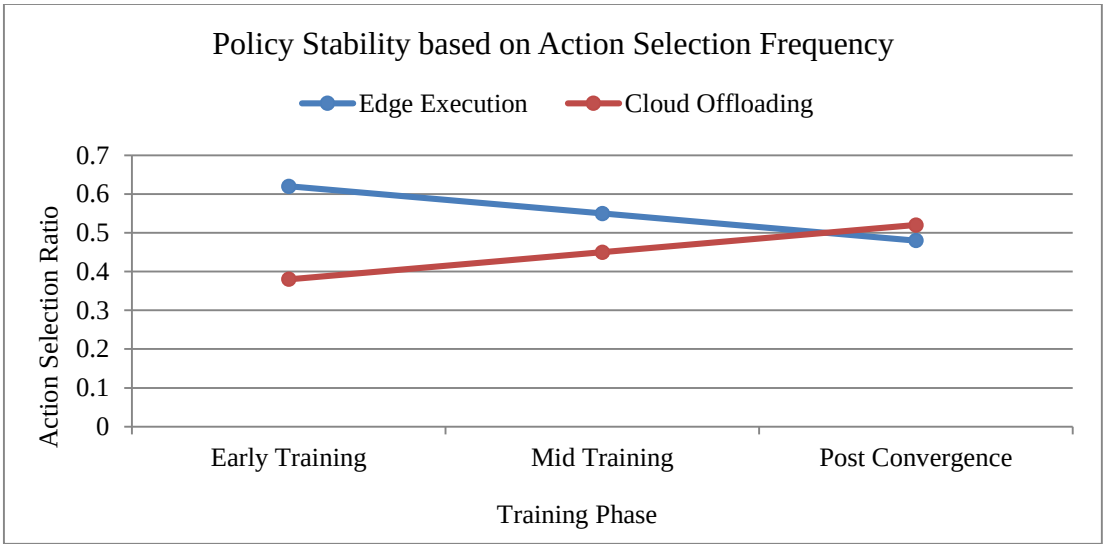

Fig. 13 Scheduling Adaptation under Dynamic Workload Changes

The adaptability of the proposed scheduling framework under dynamic workloads is illustrated in Figure 13. Abrupt increases in load intensity cause a short-term rise in average task latencies, indicating short-term behavioral responses to environmental changes. Operating on rapid timescales, the scheduler adjusts task allocation decisions to reduce latency and stabilize it again. Such behavior demonstrates the framework's capability to adapt to non-stationary workloads and maintain performance and robustness in real-time IoT applications across edge–cloud infrastructures.

4.8. Ablation Study

Here, perform an ablation study to assess the effectiveness of different components of the proposed task scheduling framework. The study assesses the effect of each element on the system's overall performance by evaluating performance while selectively turning off features—energy-aware, latency-aware, and SLA constraints. It is essential to perform such an analysis to demonstrate that the observed performance improvements are not coincidental but result from the same joint optimization approach used throughout the model.

In the absence of energy awareness, the scheduler executes the task solely based on latency and resource availability. This configuration, however, gives the lowest latency consumption under moderate and heavy loads. The lack of energy-aware decisions leads to unsafe resource use, particularly at the edge, thereby reducing the system's overall sustainability.

Without latency awareness enabled, the scheduler puts all its pressure on being energy-efficient and generally prefers cloud offload to reduce local energy consumption. This eliminates energy consumption but increases task completion time and the number of missed deadlines. This setup illustrates how central its latency-aware prioritization can be in accommodating real-time IoT applications.

Table 6. Ablation Study Results

Model Variant	Avg. Latency (ms) ↓	Energy Consumption (J) ↓	SLA Satisfaction (%) ↑	Task Success Rate (%) ↑
Without Energy-Awareness	85.6	0.41	90.2	92.1
Without Latency-Awareness	118.9	0.29	87.6	88.4
Without SLA Constraints	104.7	0.33	82.3	85.9
Full Proposed Model	78.9	0.27	95.8	96.4

Getting rid of SLA restrictions comes with a punishing downgrade in service availability. The scheduler does not enforce SLAs explicitly and cannot guarantee application deadlines and energy budgets, resulting in lower task success rates and diminished user-level quality of service. This configuration highlights the need to maintain predictable performance; hence, the learning needs to be guided by the SLA.

As shown in Table 6, the proposed complete model, incorporating energy awareness, latency awareness, and the SLA constraint, consistently outperforms all its ablated versions. It identifies the optimal trade-off among energy efficiency, latency reduction, and service reliability, and justifies the importance of these components within the overall framework.

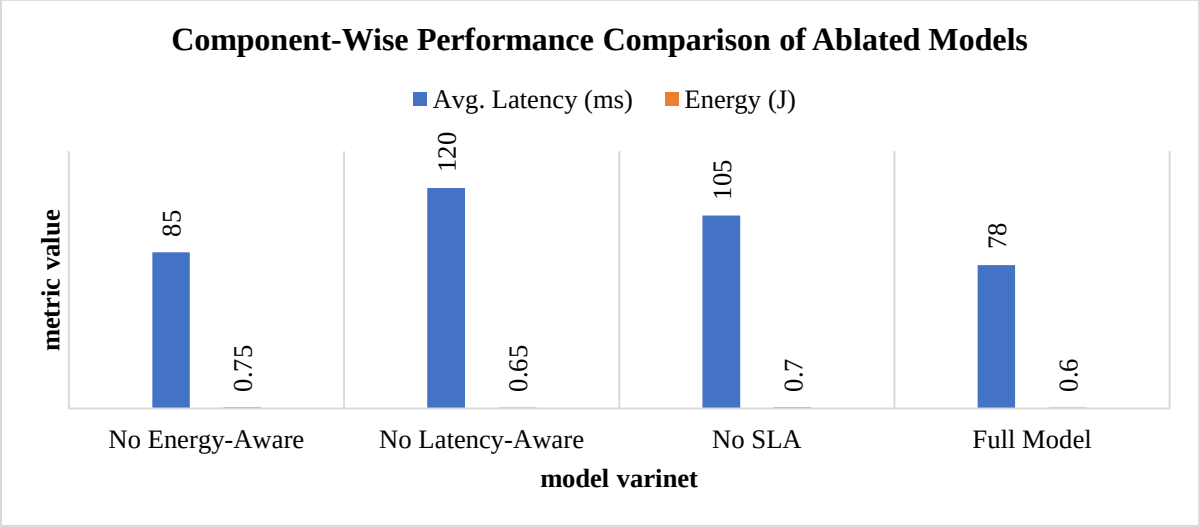

Fig. 14 Component-Wise Performance Comparison of Ablated Models

The typical latency and energy usage when each component of the proposed scheduling framework is removed individually are shown in Figure 14. If you remove energy awareness, power consumption increases; if you remove latency awareness, execution delays increase dramatically. Therefore, moderate degradation over both metrics occurs without SLA constraints. The full proposed model has been shown to have the lowest latency and energy consumption; thus, the priors on energy, latency, and SLA should be integrated. The findings confirm that optimal system performance in edge–any single element in isolation cannot achieve cloud IoT environments.

The impact of the individual constituents of the proposed framework on SLA satisfaction is shown in Figure 15.

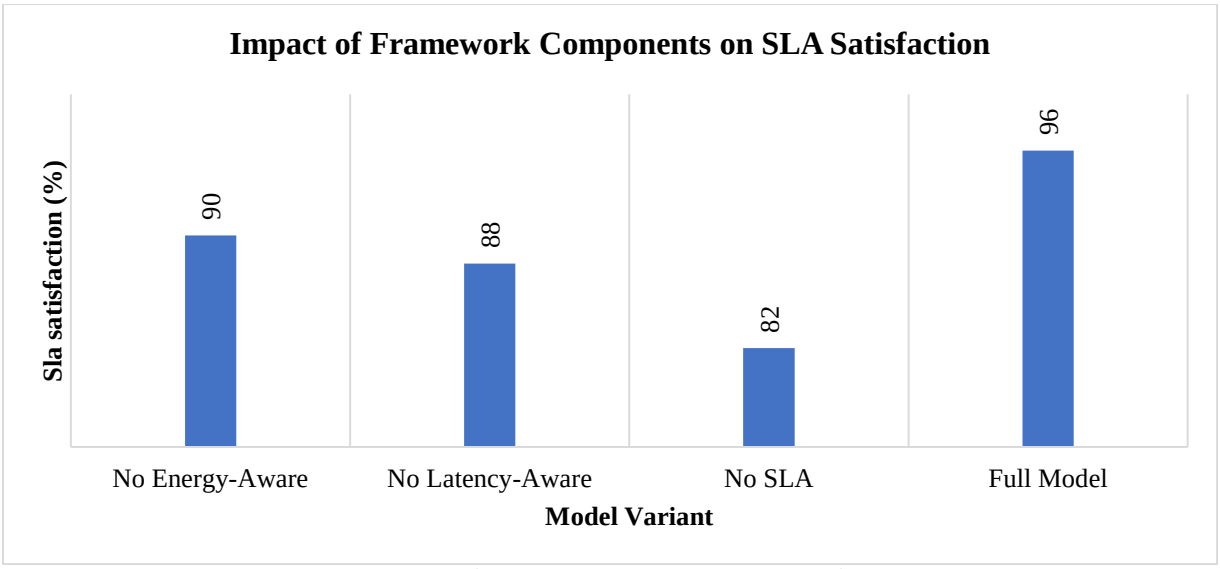

Fig. 15 Impact of Framework Components on SLA Satisfaction

SLA violations account for the largest drop in SLA satisfaction, as they stem from not enforcing SLAs in the first place, clearly demonstrating the importance of SLAs in keeping services reliable. Ablations for energy-aware and latency-aware also reduce SLA compliance, indicating that both dimensions indirectly impact SLA compliance. By requiring the least SLA to be met, the whole proposed model deftly balances explicit SLA enforcement with the need for energy and latency optimisation, thereby ensuring reliable IoT task scheduling in edge–cloud systems.

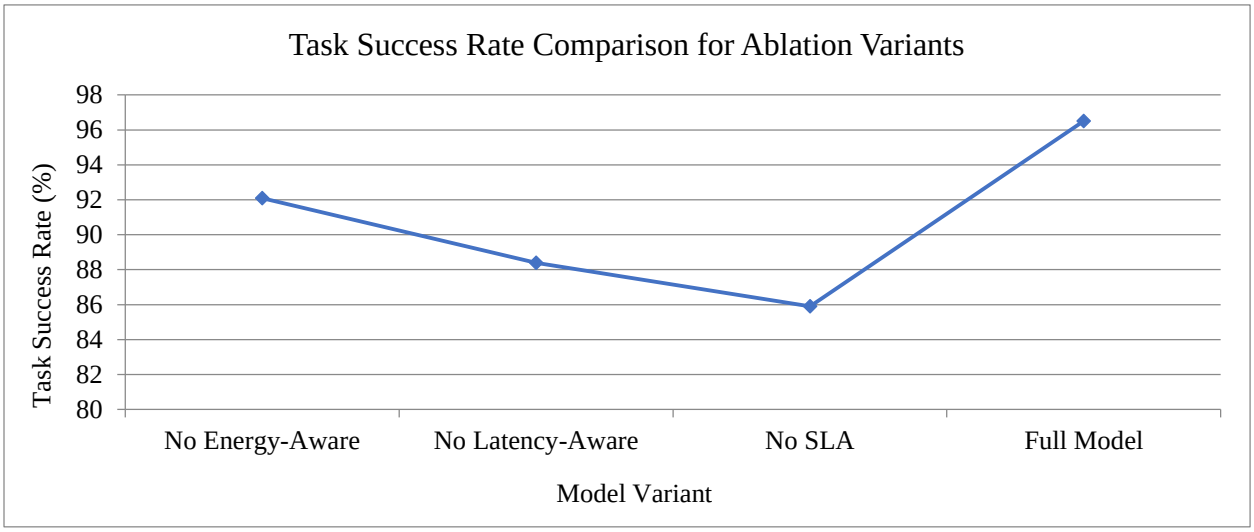

Fig. 16 Task Success Rate Comparison for Ablation Variants

Figure 16: Task success rate of various ablated versions of the scheduling framework, compared to each other. The most drastic drop in completed tasks occurs with the removal of SLA constraints, followed by the removal of latency awareness. Second, energy-aware ablation affects reliability due to increased resource contention. The full proposed model performs best in terms of task success rate, reaffirming that the integrated presence of energy awareness, latency awareness, and SLA constraints is essential for reliable, robust task execution in real-time edge–cloud IoT applications.

4.9. Scalability and Stress Testing

Assess the scalability and robustness of the identified energy- and latency-aware task scheduling framework at system scale and under workload extremes in this section. Scalability analysis: It is essential to demonstrate the effectiveness of the proposed approaches with a larger number of tasks and edge resources (scalability), while stress testing is needed to ensure that the system's behavior is correct under overload conditions, which naturally occur in real-life IoT deployments.

Increasing the task arrival rate refers to increasing the number of incoming IoT tasks per unit time (e.g., per second). Traditional heuristic schedulers tend to make static decisions, leading to rapid degradation in latency and SLA satisfaction as task arrival rates increase. In comparison, the proposed framework achieves controlled latency growth and high task success rates via dynamic adaptation of scheduling decisions. The system performs exceptionally well in meeting most SLA constraints, even though some are not fully met at very high arrival rates, resulting in slight performance degradation. Table 7 illustrates the results of scalability and stress testing.

Table 7. Scalability and Stress Testing Results

Scenario	Avg. Latency (ms) ↓	Energy Consumption (J) ↓	SLA Satisfaction (%) ↑	Task Success Rate (%) ↑
Low Task Arrival Rate	65.4	0.24	97.6	98.1
Moderate Task Arrival Rate	78.9	0.27	95.8	96.4
High Task Arrival Rate	102.6	0.32	89.7	91.3
Increased Edge Nodes (×2)	69.2	0.26	96.8	97.2
Overload Condition	128.4	0.36	82.9	85.6

The scalability with respect to the number of edge nodes is evaluated by incrementally increasing the edge infrastructure. The proposed scheduler balances task load dispatching with the increased availability of edge nodes, resulting in lower latency and better throughput.

Such a feature also indicates the potential of this framework for efficient resource utilization. In contrast, heuristic schedulers cannot afford to change their task-distribution strategies when the system topology varies.

Workloads that exceed the available resources are applied to the framework to determine the system behavior under overload conditions. Under overload, the proposed scheduler prioritizes critical, latency-sensitive tasks, ensuring they are served immediately to avoid a system crash. In contrast, less critical tasks with lower latency can be deferred or temporarily dropped, or offloaded to a backup. The proposed approach produces bandwidth scaling and correlated latency scaling, which together result in graceful performance degradation rather than service failures; see this as a very desirable property for robustness and ease of practical deployment.

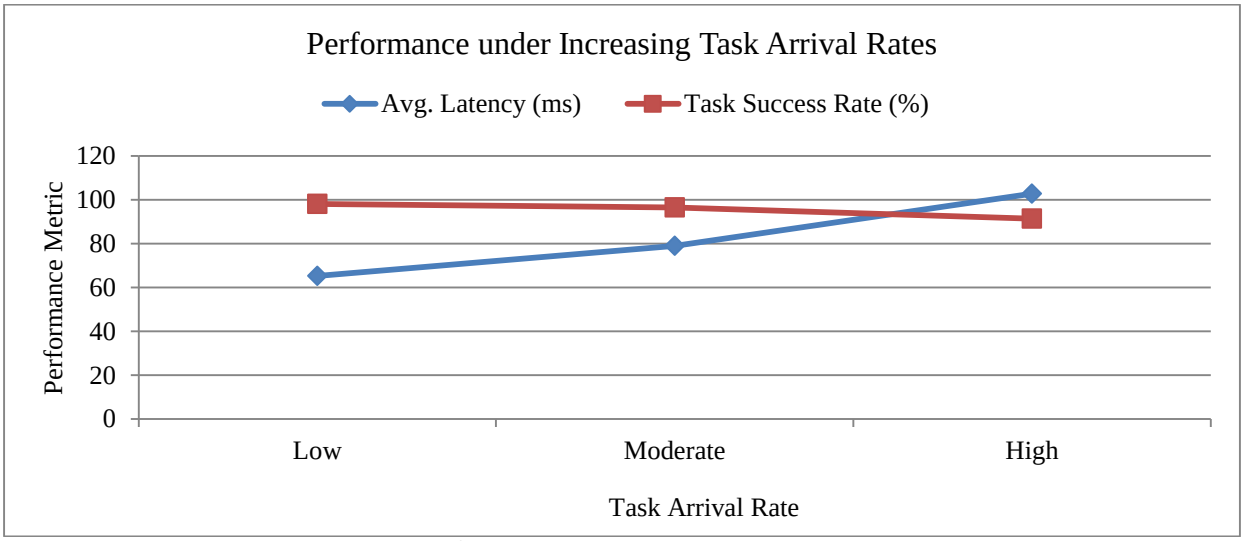

Fig. 17 Performance under Increasing Task Arrival Rates.

In Figure 17, measure system scalability by increasing task arrival rates from low to high. As workload intensity increases, average latency rises only moderately. However, the proposed scheduler achieves stability and shows only a slight performance drop. In the figure, the controlled increase in latency and the plotted data points on task success confirm that the framework can dynamically adjust scheduling decisions at higher loads. These results corroborate that the proposed approach scales well with increasing demand for IoT tasks without compromising service reliability.

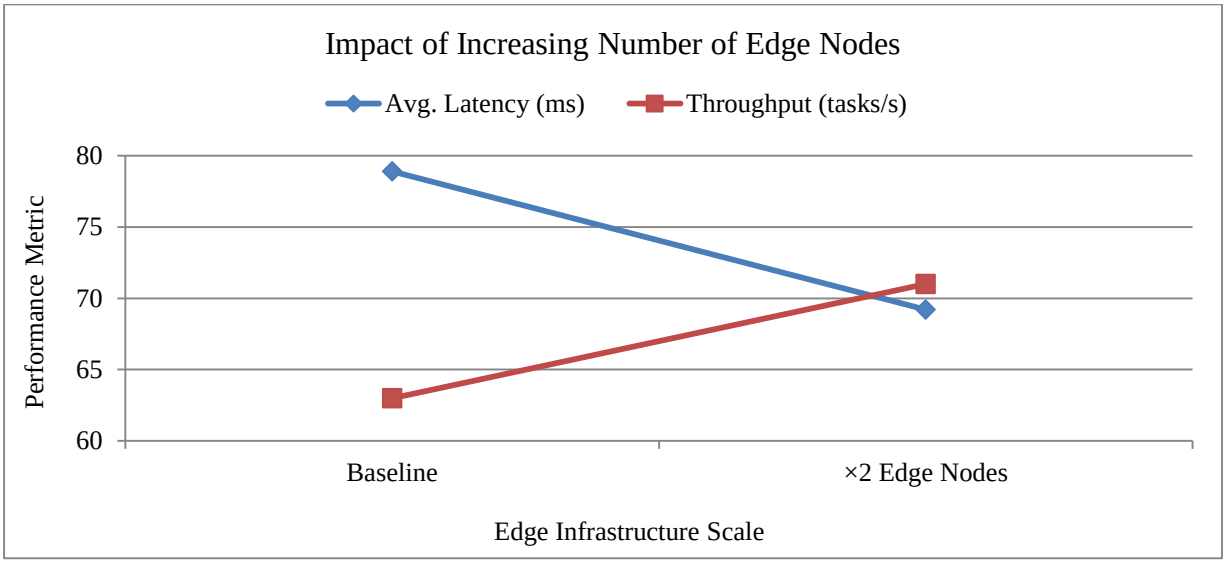

Fig. 18 Impact of Increasing Number of Edge Nodes on System Performance

System performance with a larger number of edge nodes is shown in Figure 18. Ake et al.: Stretching the edge infrastructure twice as fast will yield an 11% drop in average task latency and a 12% rise in system throughput. The increase indicates that the scheduler can better leverage more edge resources by distributing more tasks. These results confirm both the horizontal scalability of the proposed framework and its potential for large-scale IoT deployments, where edge resources are incrementally added increased on demand.

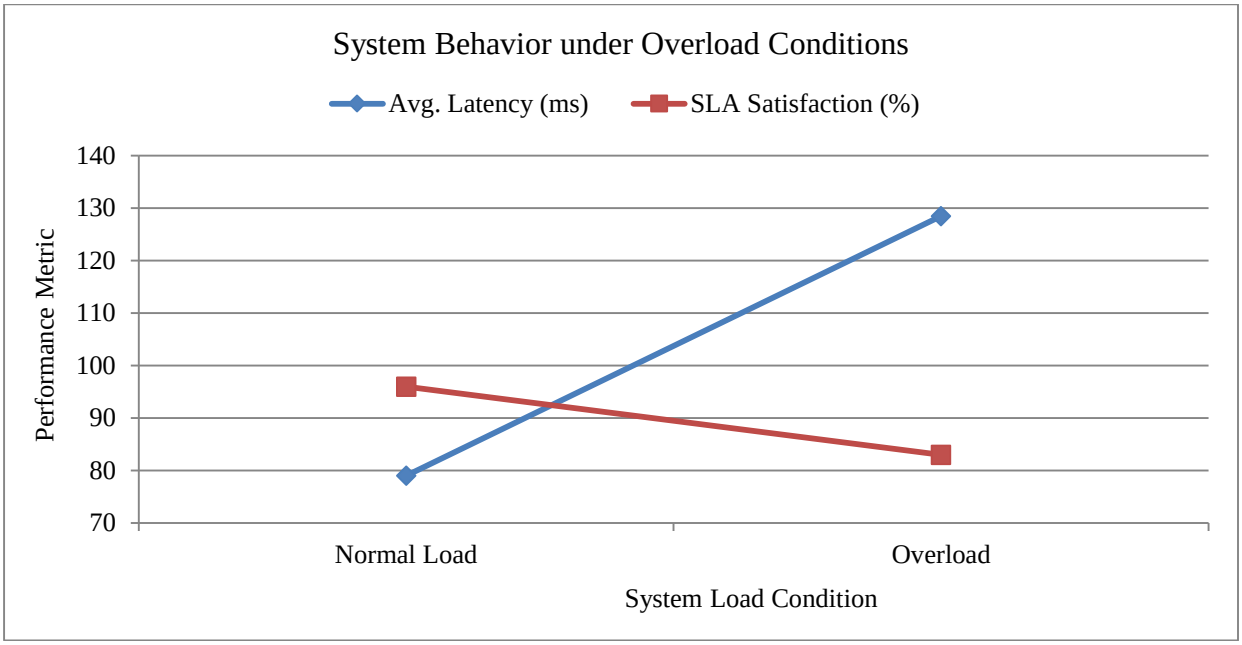

Fig. 19 System Behavior under Overload Conditions

Study the system under overload conditions, i.e., when task demand exceeds available resources, as shown in Figure 19. On overload, average latency increases, and SLA satisfaction decreases. This means that the infrastructure is experiencing stress. Nonetheless, the degradation is still graceful, not abrupt, since performance reduction is controlled. The scheduler ensures hardened tasks are executed and modifies execution strategies to retain partial SLA compliance.

Such behaviour illustrates the strength and flexibility of the proposed framework in real–life edge–cloud IoT scenarios, where workloads tend to be more bursty and unpredictable.

4.10. Discussion of Results

The proposed framework across latency, energy, SLA satisfaction, and throughput metrics consistently outperforms all baselines. This is a good topic to discuss because each improvement works in conjunction with the others. Notice that FIFO and Round Robin achieve the worst latencies (185.6 ms and 156.3 ms, respectively) because these algorithms make allocation decisions in a deadline-, resource-, and energy-neutral manner: they treat all tasks the same, regardless of urgency. By prioritizing deadlines, EDF reduces latency to 112.8 ms but employs this method aggressively, pushing energy expenditure to 0.53 J — the highest of all methods — by initiating cloud offloading immediately, regardless of whether edge computing can suffice at a lower cost. Maximum time/energy (139.4 ms, 0.39 J) is achieved by a heuristic offloading baseline that uses fixed rules based on task-size thresholds; these rules perform well under stable conditions but cannot adapt when the workload pattern changes. Notably, the DRL baseline without energy–latency awareness (101.7 ms, 0.34 J) learns from experience to improve on heuristics. Still, without a reward function that directly penalizes violations of energy and latency, it finds a reasonably fast but inefficient policy. EdgeSchedAI obtains the best result across all metrics (78.9 ms, 0.27 J, 95.8% SLA) precisely because its reward function negatively weights both energy over-consumption and latency violations while positively weighing SLA satisfaction — forcing the DQN agent to discover the single operating point that balances all three optimally instead of optimizing one to the detriment of the others.

4.11. Comparison with Existing Methods

In this section, we provide a qualitative comparison of the proposed energy-aware and latency-aware scheduling framework with state-of-the-art approaches for edge-cloud and fog-cloud task scheduling. This comparison is based on optimization objectives, learning-based performance, adaptability, and practical deployment considerations, drawing on insights from well-established methods reported in the literature.

EdgeSchedAI outperforms energy-only methods by not treating energy minimization as the sole objective. Aslanpour et al. Although both [1] and FaasHouse [7] reduce energy consumption by deferring low-priority tasks, this deferral directly increases latency—a trade-off that their frameworks accept, as they have not accounted for latency penalties in their scheduling logic. To prevent this, EdgeSchedAI encodes an additional latency penalty directly into the DQN reward function; the agent learns to immediately process urgent tasks at the edge rather than defer them, while consuming 0.27 J energy and keeping latency at 78.9 ms, a combination that neither [1] nor [7] achieves.

This is due to two one-time design choices absent in previous work, which cause EdgeSchedAI to converge faster than DRL-based methods and achieve more stable policies. Sellami et al. [2] and Jayanetti et al. [9] use DRL. Still, their reward signal does not include any explicit SLA constraints: the agent can learn policies that average out meeting both energy and latency constraints, but the Deadline of each task can be exceeded. Unlike the DRL baseline that optimizes for an average rather than a hard target by simply providing a reward proportional to the SLA satisfaction (89.5%), EdgeSchedAI's reward function has a hard constraint on the SLA; any SLA violation incurs a heavy penalty, thereby forcing the agent to treat this as a hard constraint to be satisfied (SLA satisfaction of 95.8%). Additionally, the action-space pruning mechanism of EdgeSchedAI helps reduce the decision search space by minimizing infeasible actions based on the current real-time resource status, thereby facilitating convergence to a stable policy in around 600 training episodes. Unlike meta-heuristic-based approaches such as EAWSA [4] and Fuzzy-BLWJAYA [5], EdgeSchedAI is also more adaptive because it learns online rather than offline and optimizes a predefined objective function. Meta-heuristics calculate a scheduling solution for a specific time window based on the workload profile. When the workload changes (more tasks arrive, tasks become more complex, or a node fails), the solution degrades because the algorithm cannot self-update. EdgeSchedAI's DQN agent, on the other hand, updates its Q-values throughout its scheduling decisions and, in doing so, continually re-optimizes in real time as conditions change. They can clearly observe this directly from the scalability results in Table 7: while EdgeSchedAI's SLA degrades only from 95.8% to 89.7% under a high task arrival rate, static heuristic methods suffer a much steeper degradation, as they are unable to adjust their task placement strategy accordingly.

As shown in Table 8, whereas the proposed framework combines joint energy and latency optimization with SLA-aware learning in a single decision-making process, the other methods in the table are either uni-objective or do not jointly optimize the objectives they consider. It demonstrates convergence among learning stability, workload adaptiveness, and graceful performance degradation under stress, addressing some of the most pertinent issues in state-of-the-art platforms.

Table 8. Qualitative Comparison with Existing Scheduling Methods

Method	Core Technique	Energy Awareness	Latency Awareness	SLA Awareness	Adaptability	Key Limitation
Aslanpour et al. [1]	Energy-aware heuristic	Yes	No	No	Limited	Offline modelling, simulation-only
Sellami et al. [2]	DRL-based scheduling	Yes	Yes	Partial	Moderate	Job complexity and scalability limits
Qin et al. (EAWSA) [4]	Workflow heuristic	Yes	Partial	No	Low	Simulation-only validation
Mahapatra et al. [5]	Fuzzy + meta-heuristic	Yes	Partial	Partial	Low	No online learning capability
Jayanetti et al. [9]	DRL workflow scheduler	Yes	Yes	No	Moderate	Limited real-world validation
Proposed Framework	Joint DRL optimization	Yes	Yes	Yes	High	—

Qualitative feature-level comparison between existing representations and the proposed framework (Figure 20). Prior approaches focus on either improving energy efficiency or reducing latency, but often with SLA enforcement and limited adaptability. While learning-based approaches improve adaptability, they often do not offer full multi-objective optimization capabilities. The proposed framework consistently attains high scores across all evaluated features while maintaining balance across the energy awareness, latency awareness, SLA enforcement, and adaptability metrics. The following comparison table shows how the proposed method overcomes several drawbacks of existing approaches and demonstrates its efficiency in dynamic, large-scale edge–cloud IoT environments.

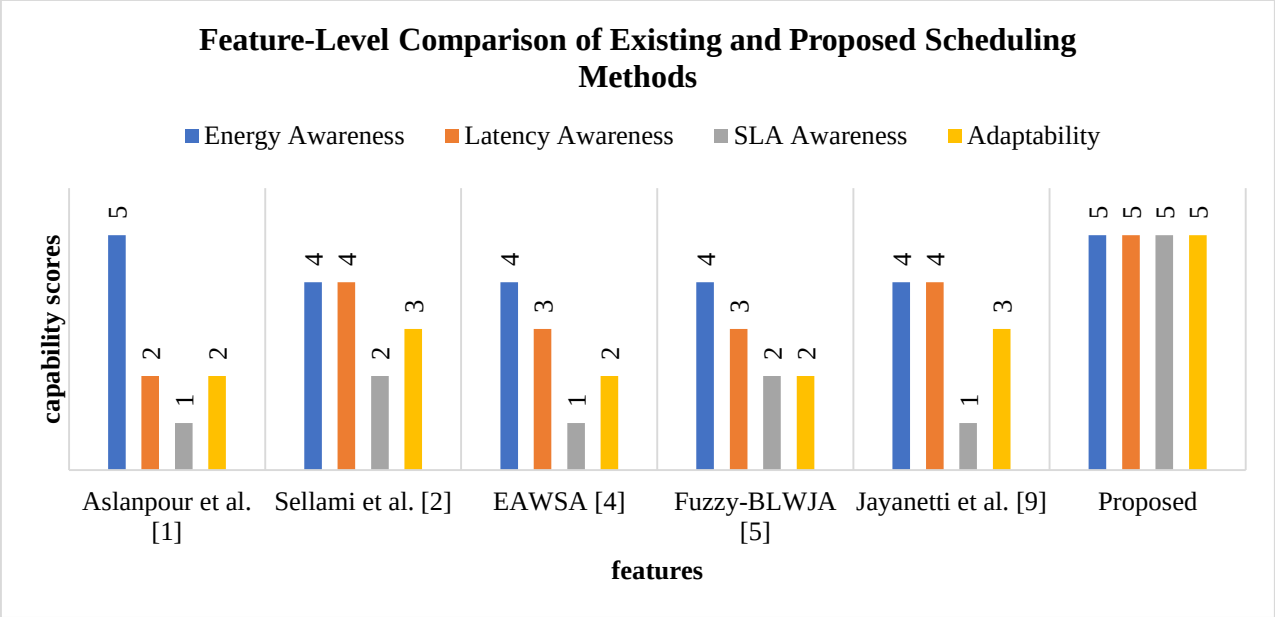

Fig. 20 Feature-Level Comparison of Existing and Proposed Scheduling Methods

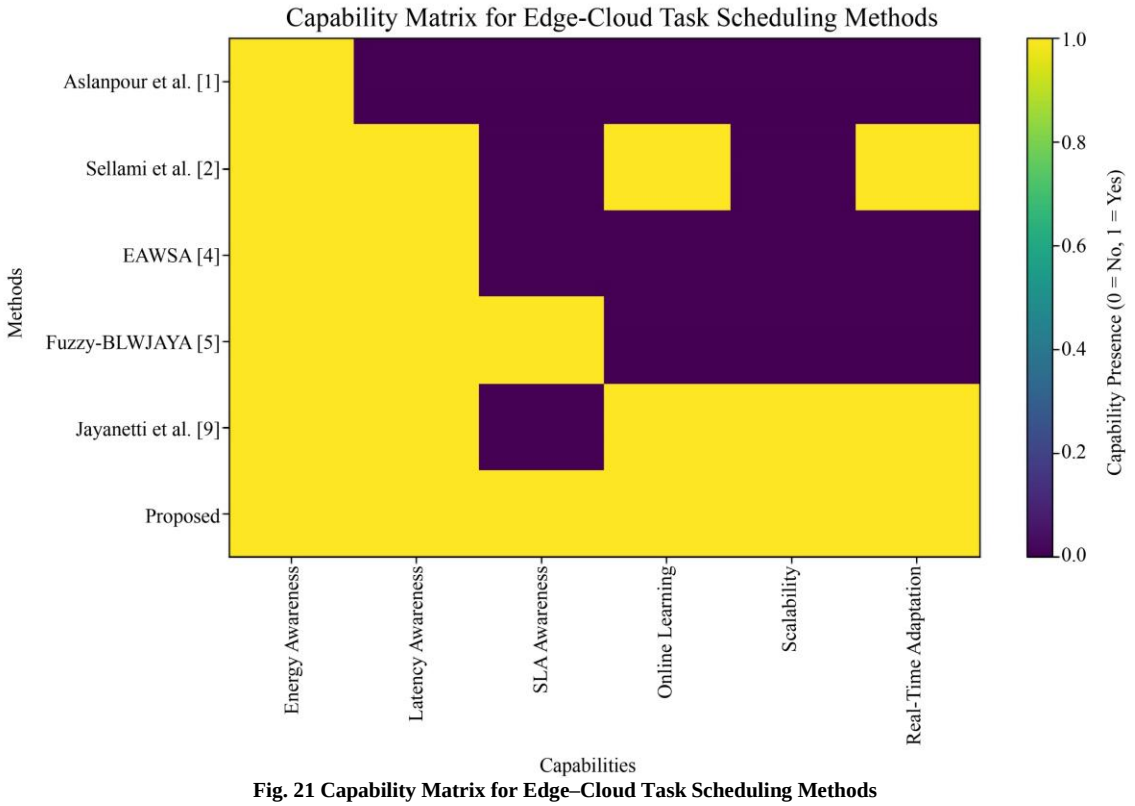

Fig. 21 Capability Matrix for Edge–Cloud Task Scheduling Methods

The overlapping area in Figure 21 illustrates the capability coverage of the existing scheduling method, in juxtaposition with work.) Existing approaches provide either energy or latency awareness but not more stringent SLA enforcement, online learning, or real-time adaptability. Many of the methods are also not scalable and are mainly evaluated through simulations. On the other hand, all dimensions exhibit full coverage of capability in the proposed framework, indicating the capability for holistic design. The following matrix illustrates how this proposed approach addresses essential gaps in current research and provides a more comprehensive solution for real-time IoT task scheduling in cloud-edge settings.

5. Discussion

With the evolution of IoT applications, which are experiencing massive growth and deployment nowadays, the quest for appropriate task scheduling mechanisms at the edge–cloud level that are highly cloud-resource-oriented has gained momentum amid latency-sensitive, energy-constrained, and dynamic workload conditions. However, natural cloud-native scheduling methods fail to meet real-time needs due to high communication overheads along the cloud-edge continuum and high centralization overheads, and pure edge-based solutions are constrained by resource limitations. As a result, current studies are limited to heuristic, metaheuristic, and learning-based approaches for improving system performance.

Among them, a significant gap found in the state of the art is the fragmented optimization of scheduling objectives. Most existing work is limited to optimizing either energy consumption, latency, or SLA compliance independently. Though these approaches yield some improvements, they cannot guarantee balanced performance at runtime for dynamic, large-scale IoT workloads. In addition, many learning-based solutions remain limited to simulation-based environments, exhibit unstable learning behavior, and have not been evaluated for robustness, scalability, or overload. This not only highlights the importance of adaptive, multi-objective scheduling frameworks but also demonstrates their reliability in realistic edge–cloud scenarios.

Considering these issues, this study suggests a first-of-its-kind scheduling framework based on deep reinforcement learning that simultaneously minimizes energy consumption and latency while explicitly satisfying SLA requirements. The main novelty lies in the joint decision system, which continuously modifies task placement decisions in real time based on system states, workload patterns, and resource states. In contrast to single-objective or static schedulers, this approach achieves trade-offs between conflicting objectives through online learning, thereby enabling sustainable and adaptive task scheduling.

Validate the proposed method by conducting a series of experiments. The other significant results are improvements in SLA satisfaction and task success rate, accompanied by substantial reductions in average task latency and energy consumption. This also provides an analysis of convergence and stability, showing that the learning process converges well to a consistent set of scheduling policies. Scalability and stress-test checks ensure the framework's performance remains stable as task arrival rates and infrastructure grow, and gracefully degrades under overload conditions.

Collectively, these results indicate that the reasons for the performance gap of EdgeSchedAI are the combined product of three complementary (and essential) design decisions: (i) a multi-objective reward function that penalizes energy overuse, latency violations and SLA violations together (compared to single-objective methods [1, 2, 4] which optimise a metric at the expense of others); (ii) online adaptive learning via experience replay (vs. static heuristics [4, 5] which cannot adapt to changes in the workload); and (iii) action space pruning that constrains scheduling decisions to only feasible options based on the current resource state (vs. unconstrained DRL baselines that waste exploration budget on infeasible actions). The ablation study in Table 6 shows that removing any one of these three elements measurably degrades performance, confirming that the gains observed are neither incidental nor disjoint. Rather, the joint design is such that the included elements are interdependent.

In essence, the framework proposed is capable of providing not just the missing characteristics of recent scheduling methods, namely adaptability, multi-objective, and robustness, in a novel single learning-based solution. These results show promise for applications in actual IoT systems, including smart cities, industrial automation, and real-time monitoring. As the present study has several limitations, these are discussed separately in section 5.1 for clarity.

5.1. Limitations of the Study

Though these results show promise, there are limitations worth acknowledging. First, because the evaluation is conducted in an emulated edge–cloud environment, uncertainties and heterogeneity endemic to genuine IoT deployments may not be fully incorporated. Then, the deep reinforcement learning model needs to be trained, which may incur initial performance overhead and computational cost before convergence, especially in highly dynamic scenarios. Third, the experiment is limited to specific workload features and network conditions, and would require further field verification to generalize the concept across various IoT applications and heterogeneous network conditions.

6. Conclusion and Future Work

In this work, we have developed a novel energy- and latency-aware deep learning-based task scheduling framework for IoT applications operating in edge–cloud computing environments. This work integrates deep reinforcement learning into the scheduling process to address key challenges posed by dynamic workloads, resource constraints, and strict service-level requirements. To enable adaptive and efficient task allocation between edge and cloud resources, the framework optimizes overall energy consumption and task latency while satisfying the SLA constraints. Extensive experimental evaluations confirm that the proposed scheduler significantly outperforms existing heuristic and learning-based approaches across multiple metrics, including response latency, energy consumption, SLA satisfaction, task success rate, and the number of tasks scaled simultaneously. The convergence and stability analysis ensure that the learning process works reliably as intended, and scalability and stress

testing validate the framework's robustness under increasing workload intensity and overload. The results of this study demonstrate the potential of multi-objective learning-driven scheduling for real-time IoT systems and validate its fitness for purpose for deployment in constrained and changing edge–cloud environments. The proposed framework addresses a critical gap by providing a well-balanced, easily adaptable solution for next-generation IoT infrastructures, tackling key limitations observed when scrutinizing existing scheduling strategies. There are multiple ways in which future work can build upon this work. Second, needed to implement and validate the framework on real-world edge–cloud test beds to understand the practical deployment challenges better. Second, use an online or federated learning mechanism to reduce training overhead and improve adaptability in a multi-site distributed environment. Additional features related to objectives, such as security, fault tolerance, and cost, will broaden the framework's applicability. Last but not least, applying this approach across a broader set of IoT applications and under heterogeneous conditions would increase its generalizability and impact in real practice.

References

[1] Mohammad Sadegh Aslanpour et al., "Energy-Aware Resource Scheduling for Serverless Edge Computing," *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, Taormina, Italy, pp. 190-199, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[2] Bassem Sellami et al., "Energy-aware Task Scheduling and Offloading using Deep Reinforcement Learning in SDN-enabled IoT Network," *Computer Networks*, vol. 210, pp. 1-20, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[3] Meng Xun et al., "Deep Reinforcement Learning for Delay and Energy-Aware Task Scheduling in Edge Clouds," *18th CCF Conference, ChineseCSCW 2023 Computer Supported Cooperative Work and Social Computing*, Harbin, China, pp. 436-450, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[4] QIN Zhiwei et al., "Energy-aware Workflow Real-time Scheduling Strategy for Device-edge-cloud Collaborative Computing," *Computer Integrated Manufacturing System*, vol. 28, no. 10, pp. 3122-3130, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[5] Abhijeet Mahapatra et al., "An Energy-Aware Task Offloading and Load Balancing for Latency-Sensitive IoT Applications in the Fog-Cloud Continuum," *IEEE Access*, vol. 12, pp. 14334-14349, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[6] Mekala Ratna Raju, and Sai Krishna Mothku, "Delay and Energy Aware Task Scheduling Mechanism for Fog-enabled IoT Applications: A Reinforcement Learning Approach," *Computer Networks*, vol. 224, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[7] Mohammad Sadegh Aslanpour et al., "Faashouse: Sustainable Serverless Edge Computing Through Energy-Aware Resource Scheduling," *IEEE Transactions on Services Computing*, vol. 17, no. 4, pp. 1533-1547, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[8] Carmen Delgado, and Jeroen Famaey, "Optimal Energy-Aware Task Scheduling for Batteryless IoT Devices," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 3, pp. 1374-1387, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[9] Amanda Jayanetti, Saman Halgamuge, and Rajkumar Buyya, "Deep Reinforcement Learning for Energy and Time Optimized Scheduling of Precedence-Constrained Tasks in Edge–Cloud Computing Environments," *Future Generation Computer Systems*, vol. 137, pp. 14-30, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[10] Yuqing Wang, and Xiao Yang, "Research on Edge Computing and Cloud Collaborative Resource Scheduling Optimization Based on Deep Reinforcement Learning," *2025 8th International Conference on Advanced Algorithms and Control Engineering (ICAACE)*, Shanghai, China, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[11] Keqin Li, "Design and Analysis of Heuristic Algorithms for Energy-Constrained Task Scheduling With Device-Edge-Cloud Fusion," *IEEE Transactions on Sustainable Computing*, vol. 8, no. 2, pp. 208-221, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[12] Wenhao Fan et al., "Collaborative Service Placement, Task Scheduling, and Resource Allocation for Task Offloading with Edge-Cloud Cooperation," *IEEE Transactions on Mobile Computing*, vol. 23, no. 1, pp. 238-256, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[13] Jiangjiang Zhang et al., "A Many-Objective Ensemble Optimization Algorithm for the Edge Cloud Resource Scheduling Problem," *IEEE Transactions on Mobile Computing*, vol. 23, no. 2, pp. 1330-1346, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[14] Kaige Zhu et al., "Learning to Optimize Workflow Scheduling for an Edge–Cloud Computing Environment," *IEEE Transactions on Cloud Computing*, vol. 12, no. 3, pp. 897-912, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[15] K. Malathi, Dr.R. Anandan, and Dr.J. Frank Vijay, "Cloud Environment Task Scheduling Optimization of Modified Genetic Algorithm," *Journal of Internet Services and Information Security*, vol. 13, no. 1, pp. 34-43, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[16] Qiqi Zhang, Shaojin Geng, and Xingjuan Cai, "Survey on Task Scheduling Optimization Strategy under Multi-Cloud Environment," *CMES - Computer Modeling in Engineering and Sciences*, vol. 135, no. 3, pp. 1863-1900, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[17] Mehrnoosh Toghyani, Reihaneh Khorsand, and Hamidreza Khaksar, "QoS-SLA-aware Optimization Framework for IoT-Service Placement in Integrated Fog-Cloud Computing," *Journal of Grid Computing*, vol. 23, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[18] A. S. Abohamama, Amir El-Ghamry, and Eslam Hamouda, "Real-Time Task Scheduling Algorithm for IoT-Based Applications in the Cloud–Fog Environment," *Journal of Network and Systems Management*, vol. 30, pp. 1-35, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[19] Ramamoorthy Karthikeyan, and Venkatachalam Balamurugan, "Energy-Aware and SLA-Guaranteed Optimal Virtual Machine Swap and Migrate System in Cloud-Internet of Things," *Concurrency and Computation: Practice and Experience*, vol. 33, no. 10, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[20] Asan Baker Kanbar, and Kamaran Faraj, "Region Aware Dynamic Task Scheduling and Resource Virtualization for Load Balancing in IoT–fog Multi-cloud Environment," *Future Generation Computer Systems*, vol. 137, pp. 70-86, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[21] Aroosa Mubeen et al., "Alts: An Adaptive Load Balanced Task Scheduling Approach for Cloud Computing," *Processes*, vol. 9, no. 9, pp. 1-15, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[22] Deafallah Alsadie, "Advancements in Heuristic Task Scheduling for IoT Applications in Fog-cloud Computing: Challenges and Prospects," *PeerJ Computer* Science, vol. 10, pp. 1-58, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[23] Soghra Mousavi et al., "Directed Search: A New Operator in NSGA-II for Task Scheduling in IoT Based on Cloud-Fog Computing," *IEEE Transactions on Cloud Computing*, vol. 11, no. 2, pp. 2144-2157, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[24] Meeniga Sriraghavendra et al., *DoSP: A Deadline-Aware Dynamic Service Placement Algorithm for Workflow-Oriented IoT Applications in Fog-Cloud Computing Environments*, Energy Conservation Solutions for Fog-Edge Computing Paradigms, Springer, pp. 21-47, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[25] Jingyao Li et al., "Low-Latency and Energy-Efficient Task Scheduling for End-Edge-Cloud Collaborative Computing," *2024 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Kuching, Malaysia, pp. 611-616, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[26] Jianhang Tang et al., "Latency-Aware Task Scheduling in Software-Defined Edge and Cloud Computing with Erasure-Coded Storage Systems," *IEEE Transactions on Cloud Computing*, vol. 11, no. 2, pp. 1575-1590, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[27] Xunzheng Zhang et al., "Energy Minimization Task Offloading Mechanism with Edge-Cloud Collaboration in IoT Networks," *2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*, Helsinki, Finland, pp. 1-7, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[28] Mengyu Sun et al., "Latency-aware Scheduling for Data-oriented Service Requests in Collaborative IoT-edge-cloud Networks," *Future Generation Computer Systems*, vol. 163, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[29] Sadoon Azizi et al., "Deadline-aware and Energy-Efficient IoT Task Scheduling in Fog Computing Systems: A Semi-greedy Approach," *Journal of Network and Computer Applications*, vol. 201, pp. 1-13, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[30] Sai Wang, Xiaoyang Li, and Yi Gong, "Energy-Efficient Task Offloading and Resource Allocation for Delay-Constrained Edge-Cloud Computing Networks," *IEEE Transactions on Green Communications and Networking*, vol. 8, no. 1, pp. 514-524, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[31] Yaswanth Chowdary Thotakura et al., "An Efficient Task Scheduling for Latency Sensitive Tasks in Edge - Cloud Computing Environment," *2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, Kamand, India, pp. 1-9, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[32] Qi Zhang et al., "Task Offloading and Resource Scheduling in Hybrid Edge-Cloud Networks," *IEEE Access*, vol. 9, pp. 85350-85366, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[33] JongBeom Lim, "Latency-Aware Task Scheduling for IoT Applications Based on Artificial Intelligence with Partitioning in Small-Scale Fog Computing Environments," *Sensors*, vol. 22, no. 19, pp. 1-12, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[34] Muhammad Bukhsh, Saima Abdullah, and Imran Sarwar Bajwa, "A Decentralized Edge Computing Latency-Aware Task Management Method with High Availability for IoT Applications," *IEEE Access*, vol. 9, pp. 138994-139008, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[35] Zhenyu Wen et al., "Janus: Latency-Aware Traffic Scheduling for IoT Data Streaming in Edge Environments," *IEEE Transactions on Services Computing*, vol. 16, no. 6, pp. 4302-4316, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[36] Abhijeet Mahapatra et al., "Latency-aware Internet of Things Scheduling in Heterogeneous Fog-Cloud Paradigm," *2022 3rd International Conference for Emerging Technology (INCET)*, Belgaum, India, pp. 1-7, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[37] Syed Rizwan Hassan et al., "Design of Latency-Aware IoT Modules in Heterogeneous Fog-Cloud Computing Networks," *Computers, Materials & Continua*, vol. 70, no. 3, pp. 6057-6072, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[38] Rabeea Basir et al., "Cloudlet Selection in Cache-Enabled Fog Networks for Latency Sensitive IoT Applications," *IEEE Access*, vol. 9, pp. 93224-93236, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[39] Arash Deldari, and Alireza Holghinezhad, "An IoT-based Bag-of-tasks Scheduling Framework for Deadline-Sensitive Applications in Fog-cloud Environment," *Computing*, vol. 107, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[40] Upma Arora, and Nipur Singh, "IoT Application Modules Placement in Heterogeneous Fog–Cloud Infrastructure," *International Journal of Information Technology*, vol. 13, pp. 1975-1982, 2021. [CrossRef] [Google Scholar] [Publisher Link]