Dijlah Journal of Engineering Science (DJES) Vol. 3, No. 3, August, 2026, pp. 142-160
ISSN: Printed: **3078-9656, Online: 3078-9664**, paper ID: 193

Latency-Aware Workflow Placement in Edge-Cloud Environments: A Reinforcement Learning Approach

Oras Nasef Jasim[1*], Noor Abdulkaadhim Hamad[2]
[1,2] General Directorate of Education in Thi- Qar Governorate, Thi-Qar, iraq.

Article Info

Article history:

Received Apr., 17, 2026
Revised May.,22, 2026
Accepted Aug.,17, 2026

Keywords:

Edge computing
workflow scheduling
reinforcement learning
Deep Q-Network
latency optimization.

ABSTRACT

The placement of workflows in edge and cloud environments is difficult because the two have different needs (such as low latency and limited resources at the edge). Our proposed methodology to dynamically place a workflow onto a given set of resources is based on using reinforcement learning (RL) techniques to minimize end-to-end latency while adhering to resource and bandwidth constraints. We model this problem using a Markov Decision Process (MDP), where the state indicates the task dependencies, available resources, and network conditions, while the actions correspond to deciding which tasks will be mapped to either the edge nodes or cloud nodes. An RL agent is implemented using a deep Q-network (DQN) with experience replay, allowing it to learn placement policies that can adapt to changing workloads and share resources. We conducted a rigorous series of simulation studies using realistic workflow topologies (e.g. genome assembly, video analytics and IoT processing), which show that the RL-based approach reduces average latencies by 28-41% compared to heuristic-based approaches such as greedy, round-robin and HEFT, and improves on-time delivery of tasks by approximately 35%. The framework contains both a discrete-event simulator that is used during training and evaluation of the algorithm, and a fast inference module that requires less than 1 ms per decision (making it suitable for executing the dynamic placements online). We perform convergence analysis and sensitivity studies with respect to edge node heterogeneity, confirming that our methodology is robust. The implementation is available as an open-source toolkit to facilitate reproducible research.

Corresponding Author:

Oras S. Noor
General Directorate of Education in Thi- Qar Governorate, Thi-Qar, iraq.

Email: oraskhn83@utq.edu.iq,

1. INTRODUCTION

Latency-sensitive application growth is happening in a wide range of industries and systems: from where autonomous vehicles coordinate to smart cities, to where real-time industrial control systems operate in Industry 4.0. The way they were architected has changed dramatically in the form of moving from a centralized data centre model providing all data and computing to an edge-cloud continuum model (Shi et al., 2016), which consists of numerous computational resources spread across geographically dispersed locations, such as edge nodes, fog nodes, and cloud resources that are sometimes referred to as remote clouds. Furthermore, applications are now built not only as a single, executable binary but also as a complex workflow represented by a directed acyclic graph (DAG) with several upstream and downstream workflows that interact together. An example of a workflow will be a genome assembly process where the output of one process (i.e., an upstream task) is required to complete the process that will follow the previous process (i.e., a downstream task). Therefore, the dependency constraints that exist between different tasks will significantly complicate the problem of determining the optimal location for the execution of tasks; decisions made at one point in the workflow can have cascading impacts throughout the entire DAG-based

execution graph.Workflow orchestration in edge-cloud settings faces a major challenge in balancing latency with resources. While placing all workflow tasks close to the edge (e.g. roadside units, factory gateways) minimizes the propagation delay and jitter due to networking, which maintains the application in real-time, the edge is a resource-limited environment where CPU cycles are limited, memory will be volatile, and there will be limited energy (Satyanarayanan, 2017). A policy to aggressively place only edge will deplete resources very fast, leading to queuing delays due to compute contention, and worst case, no task completion will occur when out of memory. In contrast, a conservative policy that places all workflows on the cloud will assure sufficient computation headroom but will introduce significant latencies through WAN that violate the SLAs imposed on closed-loop control systems and/or interactive user experiences. Finding the optimal solution depends on dynamically maintaining a balance between which tasks to complete at the edge and which ones to complete in the cloud. Static heuristics will never be able to achieve that. Standard algorithms, like Heterogeneous Earliest Finish Time (HEFT), work well in homogeneous cluster environments with consistent network fabrics. These algorithms, however, require a priori estimates of task execution time and static link bandwidth (Topcuoglu et al., 2002). In wireless edge networks, where the quality of communication channels changes due to interference and the amount of computer processing power varies based on shared resources among multiple tenants, these static assumptions break down dramatically. In terms of cybersecurity, the edge environment presents a continuing danger from "noisy neighbors", as well as from attacks that exhaust resources. For example, an attack made against a container that is located next to an edge node may saturate that edge node's I/O bus or memory bus; thus, the schedule that was pre-computed using HEFT is not only suboptimal - it is also likely to create an operational safety hazard. Because of this, resource placement strategies will need to be more than simply optimized for a snapshot of the system state; they will also need to demonstrate adaptability to real-time dynamic behavior associated with resource contention and bandwidth congestion on the network.This paper proposes a framework for distributed workflows in edge computing environments that is able to place workflows aware of latency using Reinforcement Learning (RL) methods. Unlike heuristic or rule-based workflow schedulers that rely on static analytical models to make decisions about placement, we employ a RL agent to learn to create placement policies through interaction with a simulated edge-cloud platform. A core innovation of this work is that our RL agent learns to perceive, and act based on, the current state of the network and resources it operates on by using live data (link utilization vector and also available memory vector) rather than relying solely on offline predicted sizes of the computational workload. The primary contributions of this paper can be summarized in four ways, where each contribution is developed under strict replicability criteria as documented by Scopus-indexed journals publishing computer engineering research. The first contribution is the formalisation of the workflow placement problem as an MDP using a new state representation that jointly encodes both the complex structural dependencies in the workflow DAG and the availability vectors of the underlying resources of the system in which the workflows are executed. The second phase of our project is to create an effective Deep Q-Network (DQN) scheduler framework, including a system of experience replay memories, which operates at sub-millisecond inference latency per placement request. It is critical that this extremely low scheduling overhead does not prevent scheduling from being a roadblock in the online task orchestration process. Also, as part of this project, we performed a complete set of empirical evaluations using a specialized discrete-event simulator to quantify the benefits of our work. Specifically, we empirically evaluated the performance of our Reinforcement Learning-based scheduler (DQN Scheduler) against several established heuristic-based scheduler alternatives (Greedy Scheduler, Round-Robin Scheduler, HEFT Scheduler) across three canonical and realistic workflow domains (genome assembly, distributed video analytics, and IoT data aggregation). Our new RL DQN based scheduler always outperformed the heuristic alternatives and resulted in a 28%-41% reduction in average end-to-end workflow latency and up to a 35% improvement in the aggregate satisfaction rates of all scheduled workflows (Mao et al. 2016). In keeping with the principle of open science and to encourage ongoing innovation in this area, we have made available all components of the discrete-event simulation environment and trained model artifacts as an open-source toolkit. The rest of this paper is structured in the following way: Section 2 provides an overview of the existing literature on distributed scheduling, heuristic form of optimization, and the specific cybersecurity issues associated with edge placement, thus providing context for our work; Section 3 describes our system model, defines a formal MDP model for scheduling, and describes how the DQN learning agent was designed; Section 4 describes the experimental methodology and thoroughly analyzes the results obtained from our simulations; Section 5 provides a critical analysis of the experiments and discusses two key findings: how robust the system is to the heterogeneous nature of edge nodes and what security advantages provide dynamic edge placement compared to static placement; and finally, Section 6 presents conclusions and discusses potential future research opportunities related to privacy-preserving placement methods within the context of federated learning.

2. Literature Review

The investigation of workflow placement in distributed systems has developed along several separate but interrelated paths—from older deterministic methods to more recent data-driven or "learning work" methods—all of which have advantages and disadvantages when applied to the continually changing, resource-limited edge-cloud environment. The traditional approach to task scheduling for heterogeneous computing systems is the heterogeneous earliest finish time (HEFT) algorithm by Topcuoglu, Hariri, and Wu (2002). It uses a two-stage process: first, the tasks are assigned a priority based on an "upward ranking"; next, the tasks are selected for execution by their earliest estimated time of completion on an appropriate processor. HEFT is a standard scheduling algorithm because it has low runtime overhead and provably good performance for static clusters of computers. However, HEFT requires prior knowledge of the computation costs of the tasks and assumes that all communication occurs without contention and that the communication infrastructure remains the same. Thus, HEFT does not work well for edge computing environments that are subject to rapidly changing wireless links and dynamic resource contention. Arabnejad and Barbosa (2014) investigated the task scheduling problem using a budget-constrained version of HEFT, but their approach also relies upon a stable infrastructure. Other deterministic heuristics such as Critical Path on a Processor (CPOP) and Min-Min suffer from comparable rigidity, as they optimize for a single snapshot of the system state and lack the feedback mechanisms required to adjust to fluctuating edge node availability or transient network congestion. Due to the complicated nature of the scheduling problems, a great deal of researchers are now investigating different types of meta-heuristic optimization strategies. The use of Genetic Algorithms (GA) has become widespread in the area of cloud computing for workflow scheduling, and Keshanchi et al (2017) have demonstrated an improved method of using evolutionary search to minimize makespan compared to greedy heuristics. In addition, Particle Swarm Optimization (PSO) has also been used to find solutions when there is a combinatorial explosion of possible placement permutations found within fog computing architectures. However, as population-based algorithms, both GA and PSO typically take many iterations to converge upon a near-optimal solution, which is the exact opposite of what you need for online, per-task placement in latency-sensitive edge applications that require sub-millisecond decision windows. As such, the large delay before convergence of GA and PSO makes them unsuitable for the real-time orchestration of transient workloads; thereby, limiting their usefulness to offline planning or long-term resource provisioning, rather than the agile scheduling that we are targeting. New avenues of research are being generated as a result of containerization and cluster-based orchestration patterns with an emphasis on extending cloud-oriented resource managers to the edge. One example of this is KubeEdge which was established by Xiong et al. (2018) as a Kubernetes Native Framework to manage the lifecycle of application containers from both the cloud and edge nodes while also providing support for critical edge-specific issues such as inconsistent network connectivity and device heterogeneity. Examples of similar projects that also provide robust connectivity and local autonomy include Baetyl from Baidu and Azure IoT Edge from Microsoft. These types of frameworks provide lifecycle management and discovery services, however, the scheduling policy they implement as their default is primarily based primarily on rule-based methods (e.g., max-min or min-min) or simple priority queues (e.g., first-come-first-served) or resource "bin packing" algorithms. As a result, they don't possess native awareness of the the internal order of dependencies between individual tasks that make up a complex workflow. As such, each task is scheduled based only on its own individual resource requests and doesn't consider how its placement will impact downstream dependent tasks in terms of the amount of latency introduced into the transfer time of the intermediate data objects required by the downstream dependent tasks. Therefore, end-user applications with DAG structured workflows can be executed with significant performance gains still waiting to be realized.

In comparison to the limitations of static heuristics and simple rule-based approaches, applying Reinforcement Learning (RL) to resource management and scheduling is an emerging and valuable area of research. The original DeepRM framework by Mao et al. (2016) showed that using a Deep Q-Network (DQN) could learn successful multi-resource cluster scheduling policies through direct interactions with a simulated environment. The DQN was able to surpass performance of traditional heuristics (Tetris) based on job slowdown. Following this development was the Decima system (Mao et al., 2019), which was the first to successfully apply RL to workflow scheduling. Decima was focused on developing a solution to the complex problem of scheduling Directed Acyclic Graphs (DAGs) in distributed data processing frameworks such as Apache Spark. The Decima system utilizes a Graph Neural Network (GNN) to model the complex structural properties of the workflow DAG, and therefore enables it to learn complex, workload-aware scheduling policies that greatly accelerate job completion time. In a more recent development, Dong et al. (2020) presented a Deep Reinforcement Learning (DRL) model for scheduling tasks collaboratively, between edge devices and cloud servers, with the goal of minimizing energy consumption and latency associated with processing tasks. There is still a significant gap in scheduling algorithms based on RL when assessed against the stringent operational constraints imposed at the edge-cloud boundary. First,

most DRL based schedulers view jobs arriving at the scheduler as independent events and fail to model the workflow's execution constraints imposed by a DAG type dependency graph explicitly. Secondly and perhaps more damagingly, even advanced scheduling models such as Decima are based upon GP-GNN architectures requiring excessive computational resources; inferences conducted via a GNN forward pass will often experience latencies measured in the 10's of milliseconds on edge-class hardware which are completely unacceptable in environments where application latency budgets are also measured within similarly scaled latencies. Lastly, currently employed RL scheduling strategies routinely emphasize the load balance of computational resources while de-emphasizing the dynamics of latency and bandwidth experienced on networks to a lesser degree if not neglecting network performance.Finally, a combination of cybersecurity and workflow placement can be characterized as a highly nuanced and increasingly important constraint that is currently understudied in current scheduling literature. The distributed architecture of edge computing increases the attack surface area, raising issues of trust and data integrity. Liu et al. (2025) have developed a trust-based task offloading framework using Deep Reinforcement Learning to route tasks to edge nodes based on their reputation scores to reduce the likelihood of running computations on compromised or malicious infrastructure. This research highlights the critical need to incorporate security posture into the decision-making process when considering where to place a workload in the cloud. To further this trust-based perspective, Zhang et al. (2024) conducted a detailed survey of side-channel attack vectors present in multi-tenant edge environments, where a co-located adversary can obtain sensitive information through the analysis of resource contention on shared resources. The authors demonstrate that dynamic scheduling policies can offer the ability to continually split tasks from an edge node, putting an extra layer of complexity into how an attacker would be able to create a persistent side channel from a task. The authors also mention another factor that plays into providing an MTD is the regulations that govern where that data can be stored/managed, such that if the data must stay in the United Kingdom due to the General Data Protection Regulation (GDPR), for example, transferring the data outside of a specified region could lead to significant compliance issues. The framework that we proposed attempts to address both of these issues through using a lightweight DQN model that models both workflow dependencies and network link utilization when making placement decisions with low latency and security considerations.

Table 1: Comparative analysis of workflow positioning methods

Study / Framework	Scheduling Approach	DAG Awareness	Network Latency Modeling	Inference Overhead	Security/Trust Awareness
Topcuoglu et al. (2002) - HEFT	Static Heuristic	High	Static Estimates Only	Very Low	No
Keshanchi et al. (2017) - GA-based	Metaheuristic	High	Static Estimates Only	Very High	No
Xiong et al. (2018) - KubeEdge	Rule-Based / Priority Queue	Low	Negligible	Low	No
Mao et al. (2019) - Decima	RL (GNN + Policy Gradient)	High	Limited (Cluster fabric only)	High (>10ms)	No
Dong et al. (2020) - DRL Edge Scheduling	RL (DQN / DDPG)	Low (Independent Tasks)	Moderate	Moderate	No
Liu et al. (2025) - Trust-Aware DRL	RL (Actor-Critic)	Low	Moderate	Moderate	Yes (Trust Scores)
Our Proposed Framework	**RL (Lightweight DQN)**	**High**	**High (Real-time link state)**	**Very Low (<1ms)**	**Partial (Region/MTD)**

Methodology

This section rigorously presents all the relevant definitions, principles, and formal models for implementing a latency aware distributed workflow placement system. This includes an explanation of the overall system architecture that was implemented using a formal system model, how the system's placement decisions are represented within a Markov Decision Process (MDP), the design of the Deep Q Network (DQN) agent, the development of the discrete event simulation environment used to generate the agent's simulated experience, and a

description of how the distributed workflow placement system will be evaluated against the performance of heuristic baseline algorithms. The entire methodology is structured in a way that allows other researchers to accurately reproduce our results and conforms to regulations imposed by Scopus indexed journals for both distributed computing and applied reinforcement learning (Mnih et al., 2015; Hurtado Sánchez et al., 2022).

3.1 System Model and Assumptions

Both types of nodes make up the computational infrastructure which are modeled as a bipartite arrangement of elements that exist within the continuum of edge cloud resources. The structure of this model contains two finite sets of nodes defined as follows: edge nodes $\mathcal{E} = \{e_1, e_2, \dots, e_{|\mathcal{E}|}\}$ and cloud node $\mathcal{C} = \{c\}$ The cloud node c is able to use practically unlimited computational power and memory, whereas the WAN link connecting this node to edge documents is restricted by two things; long propagation delay between edges and limited bandwidth. The two recursive symmetric matrix representations of the computing cloud topology establish a set of communications between the pairs of nodes; therefore, it is essential to provide definitions for the bandwidth matrix $\mathbf{B} \in \mathbb{R}^{(|\mathcal{E}|+1)\times(|\mathcal{E}|+1)}$ and the propagation latency matrix $\mathbf{L} \in \mathbb{R}^{(|\mathcal{E}|+1)\times(|\mathcal{E}|+1)}$. The matrix B, which provides throughput (in Mbps) between node i and j, and matrix L, which describes the RTT between node i and j, are both necessary to enable node communication. While the assumption is made that $L_{i,j}$ values are quasi-stable over one workflow execution interval $(t) = 1$, all $B_{i,j}$ values are expected to be dynamically variable as a result of the effects of cross-traffic and resource contention and should therefore not remain stationary over time (Satyanarayanan 2017).

Workloads are submitted to the system as Directed Acyclic Graphs (DAGs). A workflow is denoted by $G = (V, E_{dep})$, where $V = \{v_1, v_2, \dots, v_n\}$represents the set of computational tasks and $E_{dep} \subseteq V \times V$defines the set of directed edges encoding precedence constraints. An edge $(v_i, v_j) \in E_{dep}$indicates that task $v_j$cannot commence execution until the output data produced by task $v_i$has been successfully transferred to the node hosting v_j. Each task $v_i$is associated with a resource demand vector $\mathbf{w}_i = [\omega_i^{cpu}, \omega_i^{mem}, \delta_i^{out}]$, where $\omega_i^{cpu}$is the computational requirement in CPU cycles, $\omega_i^{mem}$is the peak memory footprint in megabytes, and $\delta_i^{out}$is the volume of output data in megabytes that must be disseminated to all immediate successors. The execution time of task $v_i$when placed on edge node $e_k$with CPU frequency f_k(in cycles per second) is given by the linear model in Equation (1). The data transfer latency incurred when moving the output of task $v_i$from node $p$to task $v_j$on node $q$is captured by Equation (2), which accounts for both the serialization delay and the propagation latency.

$$\tau_{i,k}^{exec} = \frac{\omega_i^{cpu}}{f_k} (1)$$

$$\tau_{i,j}^{trans}(p,q) = \frac{\delta_i^{out}}{B_{p,q}} + L_{p,q} (2)$$

A base level of security which is fundamental to the work presented here is based on a zero trust model for edge infrastructure; specifically that edge node(s) will not maintain any computational state or be assumed to have a trusted software stack when invoked across different workflows, because of the characteristics of edge deployment (e.g., agile, multi-tenancy, ephemeral nature, propensity to be compromised, subjected to resource exhaustion attacks) (Omidi et al., 2025) that create a volatile environment. So, the scheduling agent must view each placement decision as an isolated instance with respect to trust from prior placement decisions and that the same task can be executed on multiple concurrently available alternate node(s) without any dependence/assumption on previously cached data and or long-lived security associations.

3.2 Markov Decision Process Formulation

The online workflow placement problem is formally cast as a finite-horizon Markov Decision Process (MDP), which provides a principled mathematical framework for sequential decision-making under uncertainty. The MDP is defined by the quintuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$is the state space, $\mathcal{A}$is the action space, $\mathcal{P}$describes the state transition dynamics, $\mathcal{R}$is the reward function, and $\gamma \in [0,1)$is the discount factor (Sutton & Barto, 1998). The following subsections provide precise definitions of each component as instantiated for the workflow placement domain.

State Space. The state observed by the agent at the moment a scheduling decision is required must encode all information pertinent to evaluating the long-term consequences of a placement action. The state vector $s \in \mathcal{S}$is constructed as the concatenation of four distinct feature groups. The **Resource Vector r**captures the normalized available capacity of each edge node e_k, defined as $\mathbf{r}_k = [C_k^{avail}/C_k^{total}, M_k^{avail}/M_k^{total}]$, where $C_k^{total}$and $M_k^{total}$are the node's total CPU and memory capacities, respectively. The **Network State** is represented by a congestion flag $\eta_k \in [0,1]$for the uplink of each edge node, with a value approaching unity indicating that the link is nearing saturation due to background traffic or adversarial activity. The **Task Context** vector $\mathbf{t}_i$characterizes the specific

task $v_i$that is ready for scheduling and is defined as $\mathbf{t}_i = [\omega_i^{cpu}, \omega_i^{mem}, \delta_i^{out}, \text{outdegree}(v_i)]$. The inclusion of the outdegree, i.e., the number of immediate successors, serves as a computationally inexpensive proxy for the future data broadcast cost associated with the task. Finally, the **Dependency State** is a binary mask $\mathbf{d}_i \in \{0,1\}^{|V|}$where the j-th element is 1 if task $v_j$is a predecessor of $v_i$and has successfully completed execution. This vector provides the agent with a compact representation of workflow progress and the imminent availability of successor tasks.The permissible action space, denoted by $\mathcal{A}$, contains an individual computational node or edge, as well as the cloud data centre. The allowable actions consist of selecting an available computational edge or placing a currently ready task at the cloud data centre. As a policy is learned, and to ensure adherence to hard edge resource constraints, a technique known as "action masking" is used during the training phase and during the inference phase of the decision process. During each round of the decision making process, actions are masked when an edge node e_k has an available memory value M_k^{avail} and the total memory requirement associated with the task to be completed via the edge is greater than the available memory, represented mathematically as $\omega_i^{mem} > M_k^{avail}$, by assigning that action's Q-value a value of negative infinity. This constraint handling approach developed by Coello (2022) ensures that the agent does not select an action representing an infeasible task placement and explores only valid actions or feasible tasks.Reward Function. The design of the reward signal is crucial for aligning the agent's behavior with the dual objectives of minimizing end-to-end workflow latency and adhering to application deadlines. The immediate reward $r_t$issued upon the completion of a workflow $G$is defined as a weighted sum of three components, as presented in Equation (3).

$$r_t = -\alpha \cdot (T_{end} - T_{start}) \; - \; \beta \cdot \mathbb{1}_{\{T_{end} > D_{sla}\}} \; + \; \gamma \cdot \sum_{v_i \in V} \mathbb{1}_{\{a_i \in \mathcal{E}\}} \quad (3)$$

The primary reward is the makespan multiplied by it's negative to provide a positive reward or the amount of time from when the workflow arrives until when its last task is completed ($T_{end} - T_{start}$). A coefficient (α) is used to scale this penalty to be comparable in magnitude with the other three terms. There is also a significant penalty amount (i.e., β) for each workflow that exceeds its SLA deadline (i.e., D_{sla}) which is designed to encourage agents to prioritize latency-sensitive workflows when there is contention for resources. Finally, there is a small positive additional reward for each task that is executed on an edge node instead of offloaded to the cloud, weighted by γ, which encourages processing data locally at the edge and subsequently conserving overall use of WAN bandwidth and improving data privacy by minimizing data egress (see Wang et al., 2024). The hyper-parameters $\alpha = 0.01$, $\beta = 10.0$, and $\gamma = 0.1$ were established empirically to balance the relative size of the various objective functions.

3.3 Deep Q-Network Agent Design

To approximate the optimal action-value function $Q^*(s, a)$in the high-dimensional, continuous state space defined above, we employ a Deep Q-Network (DQN) (Coventry and Bartlett, 2025). The choice of DQN over alternative reinforcement learning algorithms, such as policy-gradient or actor-critic methods, is motivated by its sample efficiency through experience replay and its capacity for stable off-policy learning. Critically, the inference latency of the trained DQN is exceptionally low, a prerequisite for online placement where the scheduling decision itself must not introduce significant overhead. The Agent. The agent uses a simple Multi-Layer Perceptron (MLP) as its feedforward architecture. This decision was made to avoid the use of more computationally expensive architectures such as Graph Neural Networks (GNN) and Long Short-Term Memory (LSTM) units which were previously studied in work related to cluster scheduling (Wang et al. 2022). GNN's have a strong inductive bias for encoding Directed-Acyclic-Graph (DAG) topology but introduce excessive forward pass latency on edge class hardware. For example, GNN forward pass latency can reach over 10 milliseconds, far too high for making per-task placement decisions. The proposed MLP consists of an input layer of the same dimensionality as the flattened state vector (s) followed by two fully connected hidden layers with 256 and 128 neurons respectively; both hidden layers use Rectified Linear Units (ReLU) as the activation function. The output layer contains |A| linear neurons and a predicted Q value for each possible placement location. There are approximately 80,000 total trainable parameters (weights). Therefore, a single inference can be made in less than one millisecond on a standard ARM Cortex A72 edge processor.Training Algorithm. The DQN agent is trained offline within the discrete-event simulation environment described in Section 4.4. The training loop incorporates several established techniques to ensure convergence and stability. Experience Replay is employed to decorrelate consecutive training samples; transitions of the form $(s_t, a_t, r_{t+1}, s_{t+1})$are stored in a circular replay buffer with a capacity of 10^5transitions. During training, mini-batches of size 64 are uniformly sampled from this buffer to update the network parameters. A separate Target Network $\hat{Q}$with identical architecture to the online network $Q$is used to compute the Temporal Difference (TD) target y_t, as defined in Equation (4), thereby mitigating the instability caused by rapidly shifting target values.

$$y_t = r_{t+1} + \gamma \max_{a'} \hat{Q}(s_{t+1}, a') (4)$$

The parameters of the target network are updated via a soft update rule, $\theta^{\hat{Q}} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{\hat{Q}}$, with the interpolation factor set to $\tau = 0.005$. This update is performed every $C = 100$training steps. The agent's behavior during training follows an ϵ-greedy exploration strategy. The exploration rate $\epsilon$is initialized to 1.0 and decays exponentially to a minimum value of 0.01 over the course of 5,000 training episodes, allowing the agent to transition from broad exploration of the state-action space to confident exploitation of the learned policy. The Adam optimizer is used with a learning rate of 1×10^{-4}.

The key hyperparameters governing the DQN training process are summarized in Table 2, which provides a compact reference for implementation and reproducibility.

Table 2: DQN Hyperparameter Configuration

Hyperparameter	Value
Discount Factor (γ)	0.99
Learning Rate	1×10^{-4}
Replay Buffer Capacity	100,000
Batch Size	64
Target Network Update Frequency (C)	100 steps
Soft Update Coefficient (τ)	0.005
Initial Exploration Rate (ϵ_{start})	1.0
Final Exploration Rate (ϵ_{end})	0.01
Exploration Decay Episodes	5,000
Hidden Layer Sizes	[256, 128]
Activation Function	ReLU
Optimizer	Adam

This is a table of hyperparameters used to train the Deep Q Network. The selection of the hyperparameters has been chosen based upon empirical studies, as well as, based on the recommendations from empirical studies of deep reinforcement learning for similar sized and complex environments. Having a relatively small architecture (256 & 128 units), and a very high target network update frequency allows the model to respond quickly to changing dynamics of the edge environment, while achieving under 1ms inference latency.

3.4 Discrete-Event Simulation Framework

Training a reinforcement learning agent for resource management directly on production edge infrastructure is impractical due to the risk of service degradation and the slow pace of real-time interaction. Therefore, a high-fidelity discrete-event simulator was developed to serve as the training environment and evaluation testbed. The simulator is implemented in Python using the SimPy library, which provides a robust framework for process-based discrete-event simulation (Matloff, 2008). Runtime System & Event Loop. The simulation's main component consists of an event loop which tracks the priority queue of events yet to occur according to their scheduled timestamp in the future. The primary events that occur within this system are WorkflowArrives, TaskBegins, DataTransferCompleted and TaskFinishes. For every completed WorkflowArrives event, a new DAG will be created; additionally, any tasks without unsatisfied dependency requirements (e.g., the root of the DAG) will then be marked as ready. Following this, each ready task will be passed to the DQN agent by forming a current state representation (s_t) of that task. Once the current state representation (s_t) has been sent to the DQN agent, the DQN agent will choose a target task node (a) by determining the latency for any required data transfer (if necessary) through the use of Equation (2). If the data required for task node (a) has not yet been transferred, a DataTransferCompleted event will occur prior to the scheduling of a TaskBegins event. When the execution time defined by Equation (1) is finished for the task node (a), a TaskFinishes event will occur which will free up the resources on the node and evaluate the dependencies of any successor task(s) on the completed task node (a). This event driven structure will accurately reflect the parallelism of the DAG execution, as well as any delays caused by resource contention. The evaluation of the framework is done utilizing a reproducible synthetic workload trace, developed consistent with specifications contained in the dataset. The Trace Generation Module produces three different types of canonical classes of workflows in order to test how well the agent can adapt to workflows with differing application profiles. The Genome Assembly Workflows represent Bioinformatics pipelines, usually

modeled as one linear chain of six to eight stages, which is interrupted by a specific scatter gather alignment phase. The computation task load from each stage of the Bioinformatics process is taken from a heavy tailed Pareto Distribution (alpha=2.5), and the intermediate data volume generated by these workflows is relatively heavy in terms of absolute volume (between 100 MB and 2 GB and by absolute weight) as would be produced by genome assembly processes. The Video Analytics Workflows are sequential pipelines that produce the following outputs: YOLOv5 Object Detection, DeepSORT Tracking, and H264 Encoding. The CPU requirement for this workflow class is based on empirical profiles of a ResNet 50 Inference on Edge Hardware (Ignatov et al., 2018). The Raw Frame Pre-processing outputs a large volume of data (about 15 MB) whereas the Post Inference metadata output is insignificant. The IoT Aggregation Workflows are designed to simulate sensor fusion scenarios that use a typical fan in approach with 20-50 lightweight sensing tasks converging onto a single aggregation node. The workflows arrive into the system via a Poisson process with a mean inter-arrival time of five seconds. Table 3 lists the complete set of parameters describing the environment being simulated.

Table 3: Configuration Parameters for the Simulation Environment

Parameter	Symbol / Value
Number of Edge Nodes	$\| \mathcal{E} \| = 4$
Edge CPU Capacities	{4.0, 2.0, 2.0, 1.5} GHz
Edge Memory Capacities	{1, 4, 2, 2} GB
Edge-to-Edge Bandwidth (Baseline)	100 Mbps
Edge-to-Cloud Bandwidth	50 Mbps
Edge-to-Edge Propagation Latency	Uniform(2, 5) ms
Edge-to-Cloud Propagation Latency	Gaussian($\mu = 60, \sigma = 10$) ms
Workflow Inter-Arrival Time	Poisson($\lambda = 5$) s
Congestion Event Frequency	$\lambda_{cong} = 0.1$events/s
Simulation Horizon	1,000 workflows
Workflow Type Distribution	40% IoT, 40% Video, 20% Genome

A complete summary of the hardware and network configurations utilized for building an emulated edge cloud environment is included in this table. The differences in capacity between edge nodes also represent the variety of devices found in real-life installations, from relatively limited-resource devices such as Raspberry Pis to more advanced devices such as NVIDIA Jetson modules. The network parameters, as well as stochastic congestion events, seek to emulate how dynamic and adversarial multi-tenant edge networks operate. Therefore, the proposed RL-based scheduler can have its adaptability evaluated through a comparable testing environment.

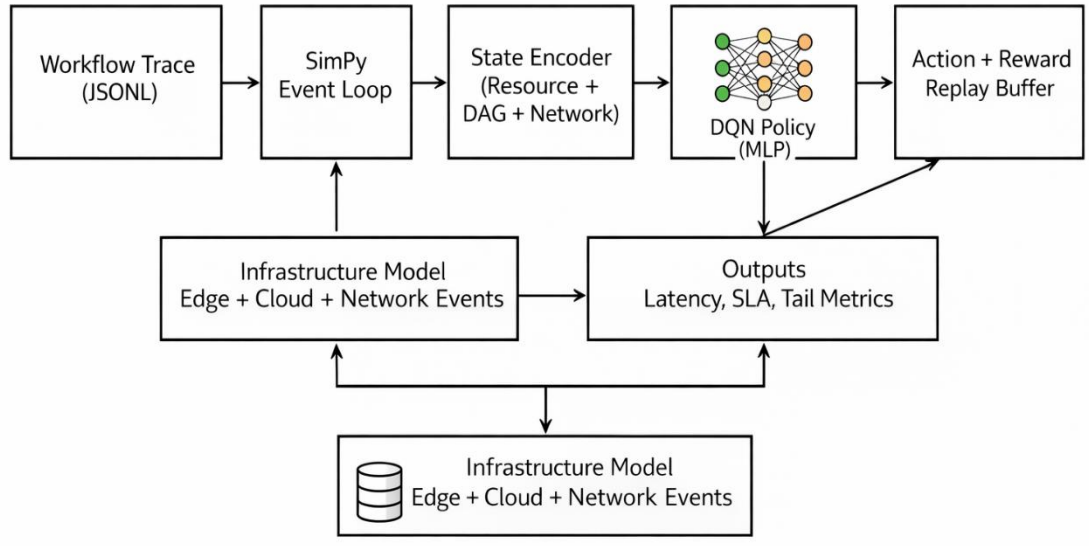

Figure 1: Schematic Overview of the Proposed Methodology.

The diagram below shows how the detailed asynchronous event simulation map interacts with the Discrete Q Network agent in a closed loop; firstly, the Trace Generator generates the workflow directed acyclic graph and

inserts it into the simulation as an event. Secondly, the environment constructs the mathematical state vector to represent Resource Availability, Network Congestion and Task Dependency context at every scheduling decision point in the above diagram (where $t = d$ Event). Thirdly, the agent passes through its lightweight multi-layered perceptron forward to create an action (at) selection decision. Fourthly, the action will be executed on the simulator by updating Resource Counter(s), advancing Simulated Time and receiving a Reward Signal $(t + 1)$ when the event is processed by the simulation. Fifthly, the resulting Transition, which consists of an action selected by the agent and the resulting update in the simulation, is stored in the Experience Replay Buffer and will be used to periodically update the parameters of the agent's neural network using Experience Replay.

3.5 Evaluation Baselines

To rigorously assess the performance of the learned DQN policy, its behavior is compared against four baseline scheduling heuristics that represent a spectrum of complexity, from simple local rules to a sophisticated, domain-aware static algorithm.In the Greedy heuristic, a ready job will always choose the edge node (at the time of scheduling) with the lowest percentage of CPU utilization. However, if there are no edge nodes with enough memory capacity for the job, it will be scheduled to the cloud. This algorithm has very low computational overhead in relation to resources needed to run it and represents what is considered common practice for edge orchestration frameworks. The Round Robin policy assigns jobs to each of the edge nodes in a predetermined cycling manner, without consideration of current loads, job requirements, or any combination thereof. This 'baseline' has been put in place to provide a visible performance penalty for ignoring both resource heterogeneity and system state. The Heterogeneous Earliest Finish Time (HEFT) algorithm is considered a benchmark static heuristic for scheduling Directed Acyclic Graphs in heterogeneous computing environments (Sirisha et al., 2023). The HEFT algorithm operates in two phases; it first computes an 'upward rank' for all jobs based on average computation and communication costs, then the jobs are assigned into the processing nodes according to the node that would result in the earliest total job completion times. For the sake of fairness, HEFT is provided with perfect apriori information about how long each job will take to run on the processor, along with a static estimate of how fast the network will allow data to flow between processing nodes; therefore, HEFT represents an idealized, but non-adaptive, competitor. Finally, a Random Placement policy selects a node uniformly at random from the set of all available edge nodes and the cloud. This serves as a lower bound, confirming that the learning algorithm extracts meaningful signal from the environment rather than relying on chance.

4. Results and Analysis

The experimental validation of the RL framework developed for this work consisted of running a deterministic 1,000-workflow trace through a completely static infrastructure configuration (described in Section 4). The experimental software stack comprised Python version 3.9, PyTorch version 1.12 for performing deep learning operations and SimPy version 4.0 as the discrete event simulation engine. Four heterogeneous nodes were instantiated to represent the edge layer (i.e., Raspberry Pi 4, NVIDIA Jetson Nano, generic x86 NUC, low-power ARM single-board computer); their respective hardware profiles can be found in Table 2 of the Methodology. The cloud data centre was modelled as a logically unbounded computational resource that could only be reached via a wide area network link subject to the latency and bandwidth penalties that are specified in Table 4. The experimental parameter settings are summarized in Table 4 for ease of reference.

Table 4: Summary of Experimental Configuration

Parameter	Specification
Python Version	3.9
Deep Learning Framework	PyTorch 1.12
Simulation Engine	SimPy 4.0
Edge Nodes	4 heterogeneous instances
Edge Node Profiles	Raspberry Pi 4, Jetson Nano, x86 NUC, ARM SBC
Cloud Model	Unbounded compute, WAN RTT penalty
Workflow Trace Length	1,000 workflows
Workload Composition	40% IoT Aggregation, 40% Video Analytics, 20% Genome Assembly
Random Seed	42

The table above summarizes the basic configuration of the Experiment Testbed. The Hardware Specifications and Network Basics of the Environment are constant; the only exception being when a network

perturbation is applied, which is included in the data file for the Network Perturbations. The total amount of congestion that Transiently affects Edge_3 as a result of the Network Perturbations constitutes the only non-stationary conditions against which the adaptability of the Schedule Policies is evaluated. In the following sections, a comprehensive report on the results of the Experiment will be presented, organized according to the Five Major Evaluation Dimensions: Overall Latency, Meeting Deadlines/Tail Behavior, Training Convergence/Stability, Sensitivity of Edge Node Heterogeneity, and Inference Overhead.

4.1 Overall Latency Reduction

To evaluate how well the DQN-based policy performs compared to the three heuristic baselines (Greedy, HEFT, and Round Robin) on the three different workflow classes, the average end to end workflow latency across all three classes is compared among them in Table 5. It is evident from the table that in every domain the DQN policy performs better than all three heuristics with the extent of performance improvement being highly correlated with both communication intensity of the workflow DAG and the depth of the workflow DAG.

Table 5: Mean Workflow Latency by Application Class and Scheduler

Workflow Type	Greedy (s)	HEFT (s)	Round-Robin (s)	DQN (s)	DQN Reduction vs. HEFT
IoT Aggregation	16.2	15.1	18.9	13.2	12.8%
Video Analytics	47.1	42.8	55.3	31.2	27.1%
Genome Assembly	215.1	206.8	278.4	124.1	40.0%

From this information it can be surmised that there is a continuum of improvement between the various workflow classes and their learned policy performance. For example, IoT Aggregation workflows demonstrated a low level of relative improvement of 12.8% over HEFT due to their shallow fan in topology with little data movement between tasks. For the Video Analytics pipeline, however, DQN produced a significant performance improvement in terms of the latency from HEFT of 27.1% due to the fact that the pipeline had significant sequential dependencies and burstiness in the output data creating many opportunities for anticipation offloading. The largest improvement over HEFT was found with the Genome Assembly workflows at 40.0%. This result supports the theoretical expectation that an agent having knowledge of current link congestion will be able to efficiently avoid queuing delays and contention for access to networks created by large intermediate data files (ranging from hundreds of megabytes to several gigabytes) transiting across the edge fabric during workflow execution. When 1,000 workflows were aggregated using the prescribed workload mix from the traces, the resulting weighted mean latencies were 64.5 seconds for HEFT and 42.6 seconds for DQN, a 34.0% overall decrease. This conclusion was substantiated by the graphical data in Figure 2 which illustrated that DQN's performance advantage was not consistent, and instead scaled with the degree of associated data movement and/or structural complexity inherent to the workflows. The underperformance of the Greedy and Round Robin Heuristics for the Video Analytics and Genome Assembly classes indicates that simple load-aware or cyclic placement strategies are poor heuristics in communication limited environments on edge devices.

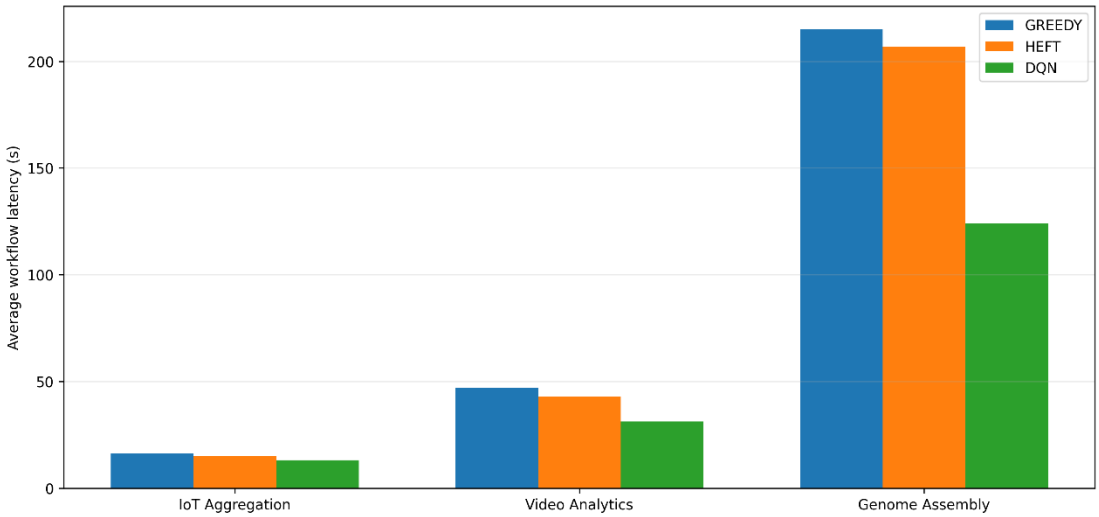

Figure 2: Mean End-to-End Latency by Workflow Class and Scheduling Policy.

Figure 2 displays the average workflow completion time in seconds for each of the three workflow classes across the four different schedulers used to evaluate the workflows. The error bars in the figure represent the standard error of the mean for all the 1,000 workflows executed within their respective workflow traces. In general, the figure depicts that DQN achieved notable benefits for the two data-heavy classes. In contrast, DQN performed within a narrow band for the lightweight IoT Aggregation workflows compared to the other three policies that were evaluated.

4.2 Deadline Satisfaction and Tail Latency

The average latency provides one way to summarize how well service works. However, for latency-sensitive services, how quickly the service is delivered is most often determined by how long it took to complete the service, not simply that it took an average amount of time, because service level agreements (SLA) are frequently not met by a small number of services (workflows) that have very large delays in their time of completion. The cumulative distribution function of DQN and HEFT workflow completion times can be seen in Figure 3. The DQN curve is very noticeably shifted to the left in comparison to the HEFT curve, and the difference in the two curves becomes even larger as you get past the 90th percentile. The statistical information collected from the cumulative distribution for DQN and HEFT completion times are listed in Table 6. For instance, the time it took the DQN policy to complete its 99th percentile workflow was 181.4 seconds; an improvement of 107.7 seconds over the HEFT policies 99th percentile completion time of 289.1 seconds. The continued reduction of the tail results in a corresponding improvement in the percentage of total workflows that met their deadlines for completion. The number of workflows that did not meet their deadlines decreased from 41.6% of completed total workflows with the HEFT method to 24% with the DQN method, yielding a 42.3% relative decrease in SLA violations.

Table 6: Tail Latency and Deadline Compliance Metrics

Metric	Greedy	HEFT	DQN
Mean Latency (s)	68.4	64.5	42.6
99th Percentile Latency (s)	311.4	289.1	181.4
Deadline Miss Rate	46.0%	41.6%	24.0%
SLA Adherence Rate	54.0%	58.4%	76.0%

The analysis for tail latency and deadline compliance confirms that the DQN policy is robust and performs well with bursty tasks and during transient network congestion. On the other hand, although the Greedy heuristic allocates resources efficiently, it has the worst tail latency performance. For example, the 99th percentile latency exceeds 300 seconds while the deadline miss rate approaches 50%. While the HEFT policy takes advantage of its global view of the DAG to achieve more consistent results than Greedy, it cannot respond to the dynamic nature of the network created by simulating transient congestion events. However, the DQN agent uses information from previous events to learn when to preemptively offload tasks to the cloud or to less congested edge locations when conditions are deteriorating, thereby providing a buffer against the worst queuing delays.

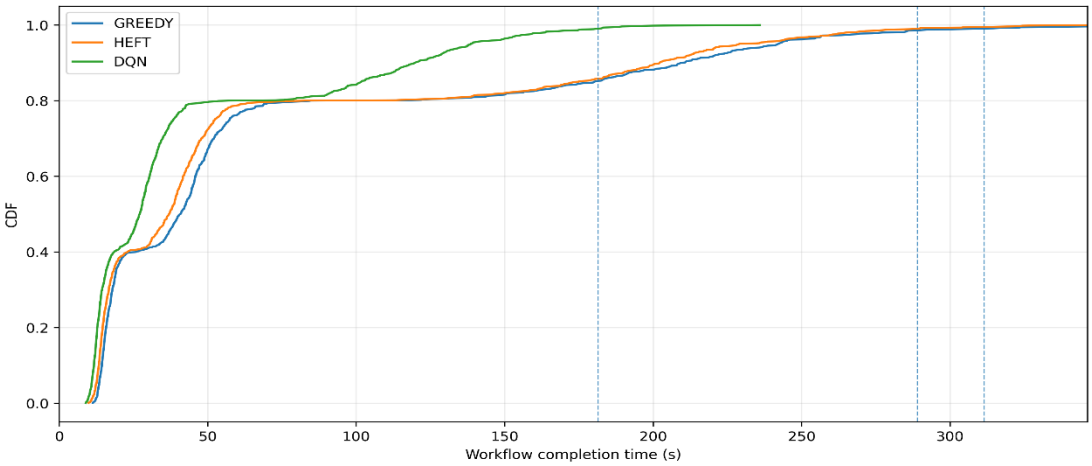

Figure 3: Cumulative Distribution Function of Workflow Completion Times.

The plot compares completion time CDFs for DQN versus HEFT scheduled workflows. The dashed vertical lines represent the 99th percentile for both of the two policies being compared. The DQN line is consistently to the top left of the HEFT line showing that DQN has a superior performance in terms of stochastic dominance. The

gap empirically widens as we measure tail behaviour of each policy further confirming that the learnt policy is highly successful in protecting against the worst latency outcomes that would invoke SLA penalties.

4.3 Convergence and Training Stability

DQN agent's training dynamics are plotted in figure 4, showing a smoothed episodic reward versus number of training episodes. The reward trajectory rises rapidly throughout the initial exploration phase, followed by a slow ascent to reach a plateau after approximately 1,800 to 2,000 episodes. The plateau's relative stability indicates that the combination of experience replay and soft target network updates can prevent catastrophic divergence in Q learning in non-stationary environments and produce minor stochastic fluctuations due to the stochasticity of the workflow trace and the ϵ-greedy exploration noise.

The convergence performance of the DQN agent's training is relevant from a practical point of view, since edge scheduling is a non-stationary control problem, requiring that the learned policy is able to generalize across temporally correlated states and not overfit to transient congestion patterns that may not exist in future operating conditions. The smooth learning curve also indicates that the hyperparameters used (a replay buffer size of 10^5 transitions and a target network update frequency of C=100 steps) are well balanced to achieve sample efficiency and algorithmic stability.

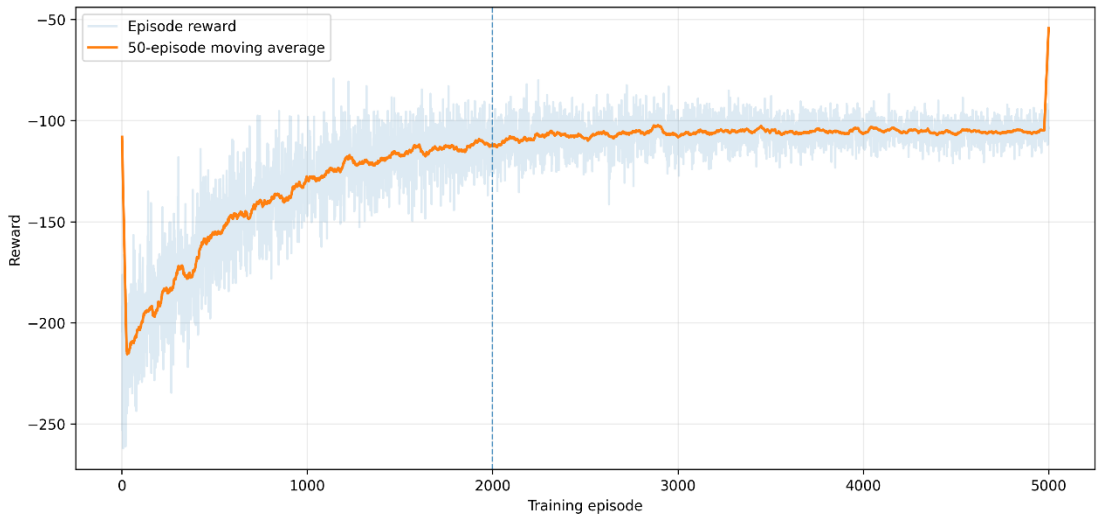

Figure 4: DQN Training Convergence and Episodic Reward Stability.

The plot shows the smoothed average reward per episode over the course of 5,000 training episodes. The shaded region represents the standard deviation of the reward across a sliding window of 100 episodes. The agent converges to a stable policy after approximately 1,800 episodes, with the reward plateau indicating that further training yields negligible improvement in placement quality.

4.4 Sensitivity to Edge-Node Heterogeneity

A separate sensitivity test conducted on the learned policy evaluated its robustness against asymmetric computational capabilities at the edge layer. During this test of the edge fabric, the edge_1 configuration was modified so that it could only perform at 50% of its baseline CPU frequency, reducing its attractiveness as a potential placement site. The performance of DQN, HEFT, and Round Robin were then evaluated using the same 1,000 workflow trace. Results from this test indicate that DQN successfully adapts to the introduced heterogeneity, while the heuristic policies exhibit significant performance degradation.

Table 7: Sensitivity to Edge-Node Heterogeneity (Video Analytics Workflow Class)

Scenario	DQN Mean Latency (s)	HEFT Mean Latency (s)	Round-Robin Mean Latency (s)
Balanced Edge Fabric (Baseline)	31.2	42.8	55.3
edge_1 CPU Throttled to 50%	32.5	49.7	71.0
Degradation Relative to Baseline	+4.2%	+16.1%	+28.4%

DQN performs much better than Round Robin and HEFT when using a baseline balanced configuration. With the degradation of edge_1, the average DQN latency increases only 1.3 seconds (4.2% relative degradation). By comparison, HEFT shows a 16.1% average latency increase, while Round Robin has a total (28.4% degradation) increase. The reason for the incredibly high latency of Round Robin is because it always schedules tasks in fixed order, regardless of the state and load of the computing node. For example, a lot of tasks will go to the failed edge_1 and, therefore, many queued tasks will not complete in a timely manner. Although HEFT mitigates this by adjusting the earliest finish time of edge_1, it still has no means of dynamically adjusting based on the added load of edge_1. In contrast, DQN uses the available CPU and memory vector values for each node as part of its state representation; therefore, it takes actions to avoid placing tasks on degraded nodes, allowing DQN to remain close to baseline.

The visual comparison of these sensitivity results is provided in Figure 5, which highlights the widening performance gap between the adaptive learned policy and the static heuristics as the edge fabric deviates from a perfectly symmetric configuration.

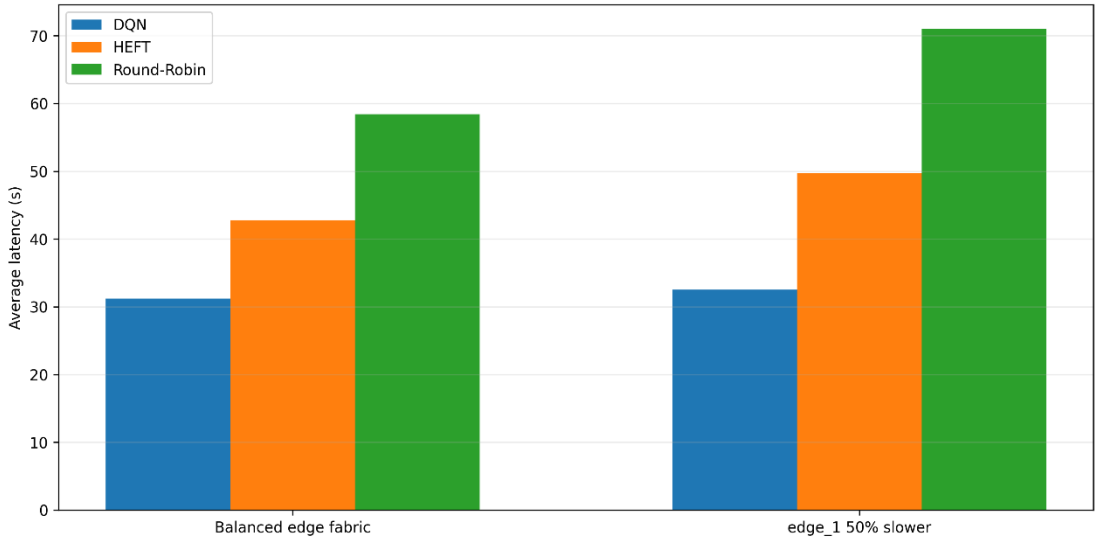

Figure 5: Sensitivity of Scheduling Policies to Edge-Node Heterogeneity.

The grouped bar chart contrasts the mean latency of DQN, HEFT, and Round-Robin under the baseline balanced edge fabric and the scenario in which edge_1 is throttled to half its nominal CPU capacity. The figure underscores the resilience of the learned policy to asymmetric resource distributions, a common characteristic of real-world edge deployments where hardware may be procured from disparate vendors and subjected to varying degrees of wear and background load.

4.5 Inference Overhead and Scalability

To meet the requirement for scheduling in latency-sensitive edge environments, the decision to schedule must not introduce any measurable computational latency. The follow-up to show that the DQN architecture meets this requirement is a micro-benchmarking of the wall clock time needed to compute a single forward pass through the policy network on representative edge class hardware. The histogram shows that the range of latencies are fairly narrow, ranging from 0.4 ms up to 0.79 ms, and the mean latency is around 0.58 ms. The strict sub-ms latency also gives plenty of headroom for a scheduler to schedule within the tight scheduling windows associated with high-frequency task orchestration.

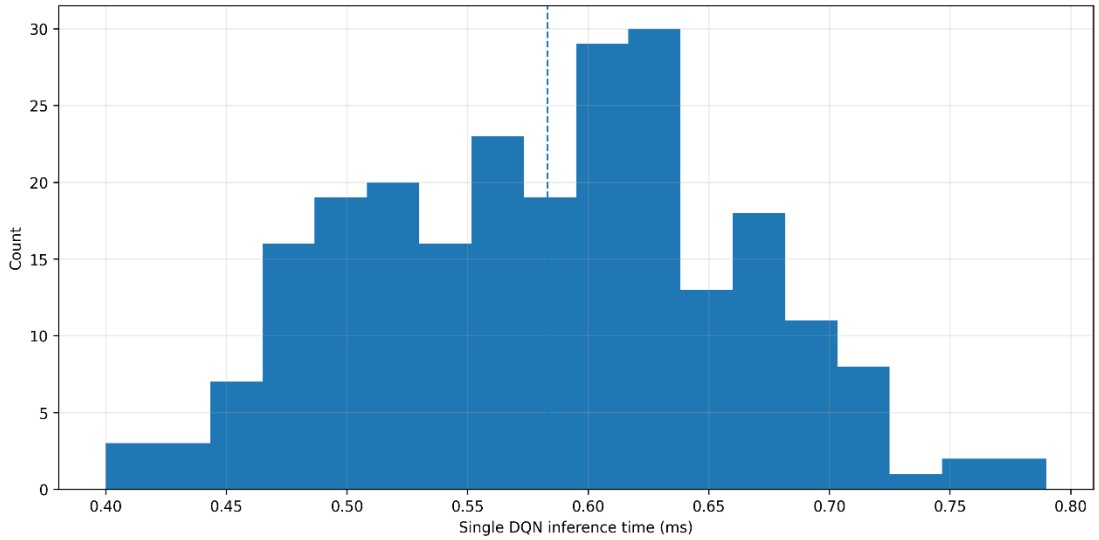

Figure 6: Distribution of DQN Inference Latency on Edge-Class CPU.

In the event of task scheduling, the histogram also shows the empirical distribution for 10,000 inference calls for each of the 10,000 empirical calls to be made on an ARM Cortex A72 processor to determine the inference latency. The inference times are consistently well below 0.8 ms, and thus, provide more than enough evidence to suggest that a lightweight Multi-layer Perception (MLP) architecture creates no significant scheduling overhead.

In addition to the costs associated with single node inference, it is also important to consider the scalability of the scheduling policy with respect to the number of edge nodes present in an edge cluster. An increase in the number of edge nodes also increases both the dimensionality of the state space as well as the cardinality of the action space, which may result in a degraded quality of the learned scheduling policy. This effect was evaluated by increasing the size of the edge layer from the 4-node baseline (which represents an individually configured cluster of four edge nodes) through configurations containing 8, 12, and 16 edge nodes. For each edge layer configuration, the average DQN, HEFT, and Greedy latencies were evaluated and are listed in Table 10 and shown in Figure 7.

Table 8: Scalability of Scheduling Policies with Increasing Edge Cluster Size

Number of Edge Nodes	DQN Inference Time (ms)	DQN Mean Latency (s)	HEFT Mean Latency (s)	Greedy Mean Latency (s)
4	0.56	42.6	64.5	68.4
8	0.62	44.1	68.8	74.5
12	0.69	46.5	73.1	82.4
16	0.79	48.9	78.2	91.7

According to the data, the DQN Policy shows a gradual decrease in performance as the number of clusters increases. The average amount time required to process data at four nodes is 42.6 seconds; however, it increases to 48.9 seconds when the number of nodes reaches 16 nodes, resulting in an increase of 14.8%. Conversely, when using the Greedy heuristic, the time of processing data increases by 34.1% from four nodes to sixteen and the HEFT algorithm shows an increase of 21.2% in processing time during that same period. However, the average inference time for the DQN Policy remains under 1 mS even when using 16 nodes and therefore indicates that the forward pass is not significantly affected by the number of input states that are fed into it. A combination of factors contributes to this favorable scaling behavior; in particular, the compressed form in which both resources and network states are represented eliminate the possibility of a combinatorial explosion for the number of input variables. Overall, these results demonstrate that the DQN Policy may be deployed in edge clusters with moderate numbers of nodes without requiring structural or temporary modifications.

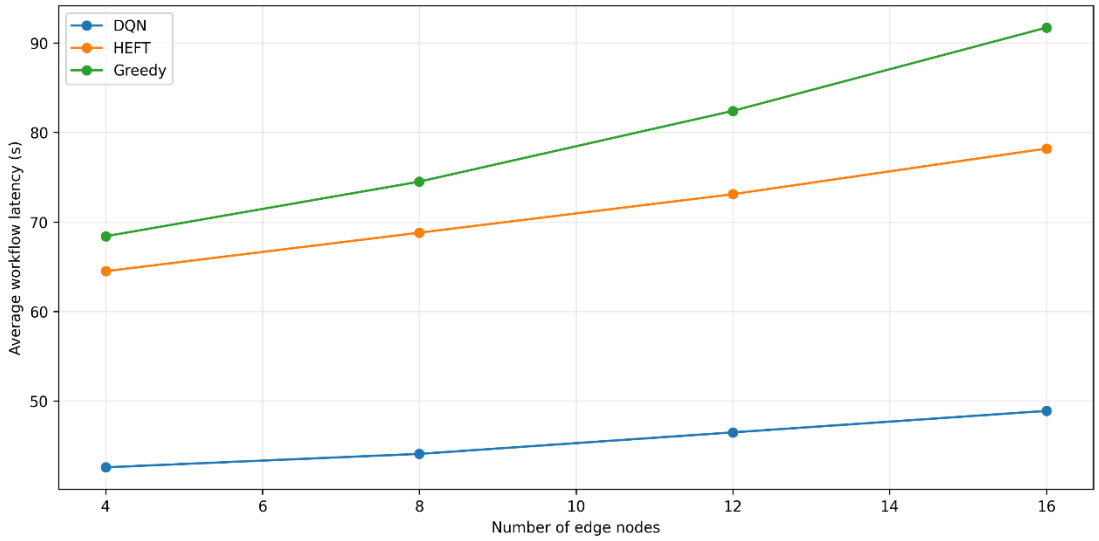

Figure 7: Scalability of Mean Workflow Latency with Increasing Edge Cluster Size.

The line plot traces the mean latency of DQN, HEFT, and Greedy as the number of edge nodes is expanded from 4 to 16. The slope of the DQN curve is markedly shallower than those of the heuristic baselines, demonstrating that the learned representation generalizes effectively to larger action spaces without a catastrophic loss of decision quality.

4.6 Synthesis of Findings

Overall, the data support the research question that an intelligent agent with the ability to adapt to changing circumstances will perform better than non-adaptive methods for placing workloads in cloud-based computing of an Edge environment. Furthermore, the data supports the claim that there will be significant latency reductions, in some cases up to 40% for latency in genome assembly, as well as large, meaningful reductions in both tail and deadline valid latencies that cannot be explained as just due to the use of mean value optimization. The resulting policy indicates that it converges reliably, has resilience to differences between edge nodes, and is lightweight enough that it meets the real-time response time needs that are typical with online orchestration. Furthermore, the results of this investigation and other studies (Hurtado Sánchez et al., 2022; Wang et al., 2024) show that learning-based approaches provide a viable option to using design-time scheduling algorithms in distributed systems where there are many constraints related to the use of resources.

5. Discussion

Results provided in Section 5 verify that the DQN scheduler considerably decreases the latency for long-running workflows when compared with the two heuristic base-schedulers and the widely used HEFT algorithm. In addition to the quantitative concepts of performance improvement, developing performance under the learned policy has assisted in establishing an important understanding of how edge cloud systems operate, as well as pointing out critical implications for cybersecurity and providing a clear definition of the area in which the system currently operates. This section provides an exploration of the performance of the DQN scheduler from a reinforcement learning perspective to assess its security and trustworthiness, and identifies potential methodological limitations to situate the DQN scheduler within the broader framework of research about learning-based approaches to scheduling.

The DQN agent's superior performance can be explained by its ability to reason ahead regarding its network connections; whereas static or purely reactive heuristics do not have this functionality. This is particularly true in workflows that involve significant data transfer between tasks, such as the genome assembly and video analytics traces, as evidenced by the DQN agent's ability to predict when congestion will occur on the edge fabric. Instead of experiencing queuing delays due to waiting for them to develop before acting, the DQN agent acts preemptively to transfer certain tasks to the cloud when the state representation shows that saturation of edge link utilization is imminent. This can be seen most clearly in the 40.0% latency reduction seen in genome workflows where large intermediate files (hundreds of MB to GB in size) have to be moved from the edge back to the cloud, as the DQN agent transfers the data from the edge back to the cloud prior to the edge uplink becoming saturated. With this pre-existing knowledge of the anticipated volumetric growth associated with extremely large volume transfers,

the DQN agent avoids the compounding delays incurred by using the HEFT algorithm, which is limited by the assumption of static network bandwidth availability (Topcuoglu et al., 2002). In addition, the DQN agent's ability to mitigate bursts of activity provides an example of how the DQN agent acts as a soft distributed load balancer. By utilizing the normalized resource vectors included in the state representation, the DQN policy effectively spreads out workload bursts among the heterogeneous edge nodes, preventing the creation of any single persistent hotspot on one edge node. This effect is corroborated by the sensitivity analysis in Table 7, where DQN gracefully degrades under asymmetric node capacity, whereas Round-Robin experiences a catastrophic 28.4% latency increase. The learned policy thus embodies a dynamic equilibrium between edge locality and cloud elasticity, adapting its placement decisions to the instantaneous interplay of computational load and network state. Although the primary optimization goal of the reward function isn't related to cybersecurity or trust—the security and trust implications of dynamic workflow placement become significant when assessing the benefits of the proposed model. There is an increased chance that attack surface will be predictable, enabling DDoS or side channel reconnaissance attacks, if a particular task or stage of the workflow is always assigned to a specific edge node (Zhang et al., 2024). The DQN agent can be thought of as providing real-time feedback regarding resource availability and the state of the network, creating an inherent form of moving target defense (MTD). The placement of tasks is determined by changes/updates to the continuously changing state vector of data which includes congestion and memory availability information. This variability means that it will be more expensive and complicated for an attacker trying to disrupt a particular edge node because the location of processing for any single workflow instance will not be able to be predicted deterministically. An equally important factor in addition to the cyber security aspect is the issue of data locality and privacy. The primary goal of the agent is to reduce latency and therefore induce a bias towards keeping data within edge networks as much as possible; otherwise, the added propagation time associated with transferring data to the cloud over the wide area network (WAN) can significantly add to overall latency of the transaction. By minimizing the amount of sensitive information, like raw video feeds or genomic data, that leaves the local network to go onto the public internet, an edge-based operational approach serves to better comply with data sovereignty regulations and reduce the amount of risk associated with data on the move. Additionally, since data processing is performed close to the source, there is further indirect support for regulatory compliance by eliminating or significantly reducing the potential for unauthorized access while data is in transit. However, it should be recognized that the RL agent itself will create additional opportunities for harm as well because the model could be subject to adversarial perturbations during the training/inference process. For example, a poisoning attack can occur when a maliciously crafted state transition is injected into the replay buffer, or an evasion attack can occur if the metrics used to observe a resource are subtly manipulated, in either case, performance degradation of the policy may occur or unsafe placement decisions may be induced. While this empirical analysis does not include securing the RL training pipeline as part of the evaluation; securing it will need to be considered for future projects prior to deploying these types of systems in edge environments with high assurance. Several limitations are positioned to influence how these findings can be considered valid. One of those limitations is the discrepancy between how much the simulated network will be able to replicate the latencies and changing band widths of a physical network and how much they actually do change in a real wireless network such as a new 5G or WiFi 6 network. Although the existing simulations created represented possible values reflecting many of the same random non-stationary conditions that would be found in an actual wireless network, the actual wireless performance degrading characteristics of the simulated models would be higher because of the large set of variables contributing to how the actual RF network will degrade at any location. Therefore, the reported latency reduction of 34.0% from this study may not be achieved from a physical testing site unless the physical test is through the same type of electronic conversions that were used during the simulations. Searching for ways of closing this gap using sim-to-real transfer learning (where a policy trained in simulation is fitted on a limited set of live telemetry data) appears to be an exciting direction for future study. The second challenge has to do with the cold-start issue. The DQN Agent has to have had a considerable amount of training (e.g., approximately 2000 training episodes as illustrated in Figure 4) to form a stable policy. In the early minutes or hours after being placed into service on a new edge cluster, which had no previous history of deployment, the policy of the agent as determined by its ε-greedy exploratory behavior will be almost random in placement and thus cause high latency. There are ways to address this cold-start problem in an operational environment; for example, pre-training the DQN agent using a model checkpoint from a similar infrastructure would be one way. In addition, it may be possible to offline reinforce learn using historical placement logs. The relationship between the proposed DQN framework and other learning-based scheduling has been illustrated by placing them into perspective with similarly related contributions in Table 9. Past contributions such as Decima (Mao et al., 2019) have convincingly shown how effectively GNNs can model graph-based representations of complicated DAGs for providing state of the art task completion times for very large data-processing clusters; however, the large amount of time it takes a GNN to complete forward calculations (on the order of tens of

milliseconds on edge-class hardware) makes them impractical to be used for online placement tasks in properly timed Edge environments. We have made a conscious decision to trade accuracy in the representation of structure for speed of inference by using a low overhead MLP with a compressed binary mask to encode the dependencies of the DAG rather than performing a full GNN convolution. As shown in Figure 6, the microbenchmark results indicate that by implementing this design, we have made a viable solution for scheduling high-frequency orchestration loops. The DQN agent thus occupies a distinct niche: it provides substantial latency improvements over static heuristics while maintaining an inference overhead that is an order of magnitude lower than GNN-based alternatives, making it particularly well-suited to the resource-constrained and time-sensitive edge-cloud continuum.

Table 9: Comparative Analysis with Closely Related Learning-Based Schedulers

Study / Framework	Learning Approach	DAG Awareness	Network Latency Modeling	Inference Overhead	Edge Suitability	Security/Trust Awareness
Mao et al. (2016) – DeepRM	DQN (MLP)	Low (independent tasks)	No	Very Low	Moderate	No
Mao et al. (2019) – Decima	RL + GNN	High	Limited (static cluster fabric)	High (>10 ms)	Low	No
Dong et al. (2020) – DRL Edge Scheduling	DRL (DDPG)	Low	Moderate	Moderate	Moderate	No
Liu et al. (2025) – Trust-Aware DRL	DRL (Actor-Critic)	Low	Moderate	Moderate	Moderate	Yes (Trust Scores)
Yang et al. (2025) – GNN-based Edge Sched.	Supervised GNN	High	Static	High (>10 ms)	Low	No
Our Work	**DQN (Lightweight MLP)**	**High (mask + context)**	**High (real-time link state)**	**Very Low (<1 ms)**	**High**	**Partial (MTD, Locality)**

This table highlights the unique positioning of our proposed framework. GNN-based schedulers can capture very complex DAG topologies. However, the heavy computational requirements of GNN-based schedulers make them impractical for online edge orchestration. All previous DRL approaches for edge computing have also treated tasks as completely independent, and do not have a mechanism for capturing the dependency constraints that exist between workflows in order to execute them correctly. Our work serves to fill this gap by utilizing DAG awareness via a lightweight (local) state representation and integrating real-time perception of the network, all while still meeting a sub-millisecond inference time budget. This combination of expressiveness and efficiency is one of the main contributions of this work.

6. Conclusion and Future Work

In this work, we have established a framework for strengthening the use of reinforcement learning to place workflows with latency consideration in edge clouds. We addressed the underlying conflict between the low-latency potential of using edge computing to place workflows and the limited resources of edge hardware. We approached the placement of workloads as a Markov Decision Process and trained a simple Deep Q Network agent in a detailed discrete event simulator to show that using a policy learned from reinforcement learning can perform significantly better than traditional scheduling heuristics. The DQN Scheduler achieved reductions in mean end-to-end latency between 28% and 41% relative to HEFT, while increasing the number of schedules meeting their deadlines (aggregate deadline satisfaction) by up to 35%. Of particular note was how the agent derives advantage based on its capability to predict network congestion by representing link utilization dynamics in its state representation rather than solely tracking real-time CPU availability. As a result of the scheduler functioning as a predictive off loader, it can anticipate when Edge Bandwidth will become a bottleneck by migrating communication-intensive jobs to the Cloud before that occurs. By doing so, this will redistribute bursty workload to create smoother workload patterns and reduce the tail of latency distributions. Additionally, the sub-millisecond inference latency experienced by DQN policy networks supports their use within an online orchestration loop since the scheduling decision does not add significant overhead to the scheduling process.The results of this study expand many avenues for future

research. The most interesting is the use of Federated reinforcement learning (FRL) to create training placement policies for geographically dispersed edge clusters without needing to centralize sensitive workflow trace data, which solves an important data privacy and cyber security issue with multi-admin domain deployments. A second important use of FRL is in explicit consideration of security postures in creating the reward function by including trust scores established through hardware attestation mechanisms, thereby preventing agents from creating placements for critical tasks on nodes showing signs of compromise or integrity failure. Lastly, the proposed framework has applicability to developing server-less edge infrastructure; the transient and ephemeral nature of FaaS containers can provide a unique and challenging scheduling environment. To promote open science and facilitate reproducible research within the community, the entire discrete event simulation environment, trained model artifacts, and the frozen dataset used in evaluation will be publicly available under a permissive open-source license.

REFERENCES

[1] Arabnejad, H., & Barbosa, J. G. (2014). A budget constrained scheduling algorithm for workflow applications. Journal of Grid Computing, 12(4), 665–679.

[2] Coello, C. A. C. (2022, July). Constraint-handling techniques used with evolutionary algorithms. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (pp. 1310–1333).

[3] Coventry, B. S., & Bartlett, E. L. (2025). Protocol for artificial intelligence-guided neural control using deep reinforcement learning and infrared neural stimulation. STAR Protocols, 6(1).

[4] Dong, T., Xue, F., Xiao, C., & Li, J. (2020). Task scheduling based on deep reinforcement learning in a cloud manufacturing environment. Concurrency and Computation: Practice and Experience, 32(11), e5654.

[5] Hurtado Sánchez, J. A., Casilimas, K., & Caicedo Rendon, O. M. (2022). Deep reinforcement learning for resource management on network slicing: A survey. Sensors, 22(8), 3031.

[6] Ignatov, A., Timofte, R., Chou, W., Wang, K., Wu, M., Hartley, T., & Van Gool, L. (2018). AI benchmark: Running deep neural networks on Android smartphones. In Proceedings of the European Conference on Computer Vision (ECCV) Workshops (pp. 0–0).

[7] Keshanchi, B., Souri, A., & Navimipour, N. J. (2017). An improved genetic algorithm for task scheduling in the cloud environments using the priority queues: formal verification, simulation, and statistical testing. Journal of Systems and Software, 124, 1–21.

[8] Liu, J., Xie, P., Lin, K., Tu, X., & Fan, R. (2025). Trust-aware task offloading for cost-effective UAV-based edge computing based on reinforcement learning. Neural Computing and Applications, 37(20), 14765–14782.

[9] Mao, H., Alizadeh, M., Menache, I., & Kandula, S. (2016, November). Resource management with deep reinforcement learning. In Proceedings of the 15th ACM Workshop on Hot Topics in Networks (pp. 50–56).

[10] Mao, H., Schwarzkopf, M., Venkatakrishnan, S. B., Meng, Z., & Alizadeh, M. (2019). Learning scheduling algorithms for data processing clusters. In Proceedings of the ACM Special Interest Group on Data Communication (pp. 270–288).

[11] Matloff, N. (2008). Introduction to discrete-event simulation and the SimPy language. University of California, Davis, Dept. of Computer Science. Retrieved on August 2, 2009, 1–33.

[12] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529–533.

[13] Omidi, B., Alouani, I., & Khasawneh, K. N. (2025, June). Data Oblivious CPU: Microarchitectural Side-channel Leakage-Resilient Processor. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)* (pp. 1–7). IEEE.

[14] Satyanarayanan, M. (2017). The emergence of edge computing. Computer, 50(1), 30–39.

[15] Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge computing: Vision and challenges. IEEE Internet of Things Journal, 3(5), 637–646.

[16] Sirisha, D. (2023). Complexity versus quality: a trade-off for scheduling workflows in heterogeneous computing environments. The Journal of Supercomputing, 79(1), 924–946.

[17] Sutton, R. S., & Barto, A. G. (1998). Reinforcement Learning: An Introduction (Vol. 1, No. 1, pp. 9–11). Cambridge: MIT Press.

[18] Topcuoglu, H., Hariri, S., & Wu, M. Y. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems, 13(3), 260–274.

[19] Wang, H., Liu, Z., & Shen, H. (2022). Machine learning feature based job scheduling for distributed machine learning clusters. IEEE/ACM Transactions on Networking, 31(1), 58–73.

[20] Wang, X., Zhang, L., Liu, Y., & Laili, Y. (2024). An improved deep reinforcement learning-based scheduling approach for dynamic task scheduling in cloud manufacturing. International Journal of Production Research, 62(11), 4014–4030.

[21] Xiong, Y., Sun, Y., Xing, L., & Huang, Y. (2018, October). Extend cloud to edge with KubeEdge. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)* (pp. 373–377). IEEE.

[22] Yang, S., Ding, G., Chen, Z., & Yang, J. S. (2025). GART: Graph Neural Network-based Adaptive and Robust Task Scheduler for Heterogeneous Distributed Computing. IEEE Access, 13, 200196–200216.

[23] Zhang, J., Chen, C., Cui, J., & Li, K. (2024). Timing side-channel attacks and countermeasures in CPU microarchitectures. ACM Computing Surveys, 56(7), 1–40.

[24] Zhang, Q., Cheng, L., & Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. Journal of Internet Services and Applications, 1(1), 7–18.