MCDS: AI Augmented Workflow Scheduling in Mobile Edge Cloud Computing Systems

Shreshth Tuli, Giuliano Casale and Nicholas R. Jennings

Abstract—Workflow scheduling is a long-studied problem in parallel and distributed computing (PDC), aiming to efficiently utilizing compute resources to meet user's service requirements. Recently proposed scheduling methods leverage the low response times of edge computing platforms to optimize application Quality of Service (QoS). However, scheduling workflow applications in mobile edge-cloud systems is challenging due to computational heterogeneity, changing latencies of mobile devices and the volatile nature of workload resource requirements. To overcome these difficulties it is important, but at the same time challenging, to develop a long-sighted optimization scheme that efficiently models the QoS objectives. In this work, we propose MCDS: Monte Carlo Learning using Deep Surrogate Models to efficiently schedule workflow applications in mobile edge-cloud computing systems. MCDS is an Artificial Intelligence (AI) based scheduling approach that uses a tree-based search strategy and a deep neural network based surrogate model to estimate the long-term QoS impact of immediate actions for robust optimization of scheduling decisions. Experiments on physical and simulated edge-cloud testbeds show that MCDS can improve over the state-of-the-art methods in terms of energy consumption, response time, SLA violations and cost by at least 6.13, 4.56, 45.09 and 30.71 percent respectively.

Index Terms—AI for PDC, Edge computing, Cloud Computing, Deep Learning, Monte Carlo learning, Workflow scheduling.

1 Introduction

The explosive growth of data and the popularity of the Internet of Things (IoT) paradigm has given rise to new parallel and distributed computing (PDC) architectures like mobile edge-cloud computing [1], [2]. Edge-cloud computing, also referred to as fog computing, is an amalgamation of cloud and edge devices that aims to bring low-latency services to the users by leveraging nodes at the edge of the network and performing only the compute-intensive processing in the cloud [3]. The ability of edge computing to provide local compute, storage and network resources has increasingly been used to execute demanding and long-running workflow applications [4], [5]. Unlike standalone workloads, workflows constitute several resource intensive and inter-dependent tasks, with their precedence constraints being modeled as a directed acyclic graph (DAG) [6]. With the burgeoning shift to AI/ML/DL based workloads, modern workflow applications have become extremely resource hungry [7]. Modern computing offers a new range of computationally heterogeneous workflows involving data gathering at the edge and processing at the cloud backend. Such workflows have recently been adapted to hybrid computing platforms like edge-cloud environments for their cost saving benefits [8], [9]. The workflow scheduling problem aims to efficiently map the tasks onto the available resources to optimize the Quality of Service (QoS) metrics.

Challenges. A significant challenge faced in workflow scheduling is the requirement of handling heterogeneous and non-stationary user demands [10], [11]. As edge and cloud nodes have different computational capacities, network bandwidth and latencies, workflow schedulers need to be aware of the resource heterogeneity to make optimal decisions. Further, the Service Level Agreement (SLA) deadlines of QoS objectives may be different for distinct applications or may even vary across the input processing requests for the same application. For instance, many industries, especially healthcare, robotics and smart-cities demand ultra-low response times for SLA sensitive tasks [12]. Battery driven mobile edge computing nodes also have strict energy consumption demands [13]. Applications deployed on a pay-per-use cloud service model aim at reducing the overall execution costs. Further, the problem of workflow scheduling is exacerbated by the dynamism of the system and the inconsistent DAG dependencies across different long-running workflows, making long-term optimization hard.

The dynamism in the system arises from two main sources: task volatility and host mobility. Task volatility covers the changing resource requirements or service demands of tasks. Host mobility entails the changing geographic location of host machines that can give rise to non-stationary latencies. With the rising use of mobile computational devices, the network characteristics like latency and bandwidth have become volatile. This requires real-time monitoring and updating the scheduling decisions, also known as rescheduling in literature [14]. As scheduling decisions change for active tasks, rescheduling might entail live task-migration, also requiring schedulers to consider migration overheads. To avoid excessive overheads, the objective score estimation of such methods needs to be extremely accurate [7].

Existing solutions. Over the past few years, several workflow schedulers have been proposed that aim to tackle the above mentioned challenges [5], [4], [15], [16]. Many of these methods use Artificial Intelligence (AI) based optimization schemes like evolutionary schemes or reinforcement learning [17]. Some classical methods also use heuris-

- *S. Tuli, G. Casale and N. R. Jennings are with the Department of Computing, Imperial College London, United Kingdom.*
- *N. R. Jennings is also with Loughborough University, United Kingdom. E-mails: {s.tuli20, g.casale}@imperial.ac.uk, n.r.jennings@lboro.ac.uk.*

Manuscript received —; revised —.

tics or meta-heuristics to optimize their decisions [18], [19]. However, most of these methods solve the challenges of workflow scheduling to a limited extent. Traditional reinforcement learning methods like Q-Learning are typically brittle in terms of the environment assumptions that makes them unsuitable for highly volatile scenarios [12]. Other methods like genetic algorithms have been shown to have the agility and adaptability required for dynamic workflow demands [5]. However, non-local jumps in many evolutionary algorithms lead to several task migrations, increasing network overheads and degrading their performance [7]. Other swarm optimization based methods that divide the timeline into periodic scheduling intervals use neural networks to approximate QoS scores of the next interval [4], [20]. However, such methods are myopic as they optimize short-term objective scores, leading them to often get stuck at a local optimum [21, Chapter 7]. Further, such methods often struggle to produce accurate QoS estimates in dynamic settings (with volatile tasks and mobile hosts) to execute apt decision optimization [22].

Background and new insights. To mitigate the risks of short-term goal optimization and generating a myopic QoS estimate, we develop on some recent ideas. In the past, schedulers have used deep neural networks to approximate the optimization objectives [5], [7]. Such neural networks are called deep surrogate models (DSMs) as they act as approximators for the QoS scores that are otherwise time-consuming to obtain (requiring physical execution of the scheduling decisions) [7]. Deep surrogate models provide an improved QoS due to their ability to accurately map objective scores and quickly adapt to diverse scenarios. However, due to the poor state-space exploration, such methods can perform poorly in settings unseen during model training [7]. As surrogate models only provide a single-step objective estimate, optimizing using such models can adversely effect long-term QoS scores. Thus, some reinforcement learning methods use look-ahead search schemes to get a long-term score estimate [21]. One such method is Monte Carlo learning that runs several multi-step executions to get an unbiased estimate of the long-term effect of an immediate action [21]. Running such executions in a real environment is infeasible due to the large execution times. As scheduling decisions need to be taken in near real-time, prior work does not use such look-ahead schemes and hence performs myopic optimization. However, recently proposed co-simulation methods seem like a promising solution for this problem [7]. Co-simulation allows schedulers to leverage event-driven simulators to quickly get a QoS estimate instead of executing decisions on a physical platform.

Our contributions. Combining Monte Carlo learning and deep surrogate models for QoS optimal workflow scheduling is non-trivial. This is because vanilla Monte Carlo learning requires the computationally expensive execution of several multi-step simulation runs with random scheduling actions. This may be infeasible when frequent decision making is required, especially for volatile environments [12]. Further, traditional deep surrogate models trained on historic data only provide the QoS score of the system state immediately after executing an action and not a long-term QoS. To address these issues, instead of randomly choosing the actions to simulate, we train a deep surrogate model to approximate long-term QoS scores within a Monte Carlo learning framework. To discover potentially better actions, we constantly explore the state-action search space. To start with, we use the action output of a short-term optimization scheme.

Drawing all these insights together, we present MCDS: Monte Carlo Learning using Deep Surrogate models for workflow scheduling. MCDS is an AI augmented scheduler that uses Upper-Confidence-Bound (UCB) exploration [21, Chapter 2] and short-term optimization action output as domain knowledge. Running co-simulations on several multi-step decisions allows MCDS to get a long-term QoS estimate of each immediate action choice. Once we have a trained deep surrogate model, we employ a gradient optimization using back-propagation to input (GOBI) scheme to reach optimal decisions quickly [7]. To test the efficacy of the MCDS policy, we perform experiments on a physical mobile edge-cloud setup with ten hosts and a simulated setup with fifty hosts with real-world application workloads. The set of schedulers used in our experiments includes MCDS and four other state-of-the-art baselines. Our evaluations show that MCDS performs *best* in terms of QoS parameters, giving at least 6.13%, 4.56%, 45.09% and 30.71% better energy consumption, response time, SLA violation rate and execution cost respectively. These results are obtained with up to 13.19% lower scheduling time than the baseline with the second-best QoS scores.

The rest of the paper is organized as follows. Section 2 overviews the related work. Section 3 outlines the system model assumptions. Section 4 details the deep surrogate model in the MCDS scheduler. Section 5 presents the Monte Carlo learning to train and generate scheduling decisions. A performance evaluation of the proposed method is shown in Section 6. Finally, Section 7 concludes.

2 Related Work

We now analyze the prior work in more detail, dividing them into two classes: heuristic and meta-heuristic approaches.

Heuristic methods. Several classical works use heuristics to optimize the workflow scheduling decisions [23], [24], [25]. Many methods in this category use heuristics motivated by the objectives they aim to optimize, such as average response time, energy consumption, execution cost, resource utilization or SLA violation rates. Some methods partition workflows into multiple sub-workflows to minimize expected communication overheads that facilitate the reduction of overall response time [24]. Other methods approximate objectives like energy consumption by easily measurable metrics like CPU utilization to output a scheduling decision [17]. Some approaches formulate the workflow scheduling task as a simplified problem like subset-sum and use mean clustering techniques [26]. However, all of these methods fail to perform well in highly volatile settings and only aim at optimizing specific objectives (via heuristics) that make them unsuitable for demanding modern-day applications [6].

Meta-Heuristic methods. This class of methods leverages high-level problem independent algorithms to find the optimal scheduling decision for the workflows. Most

state-of-the-art approaches belong to this category. Among these, many use variants of the Particle Swarm Optimization (PSO) technique [27], [4]. Such a technique starts with several random or heuristically initialized candidate solutions. Each candidate is iteratively optimized, moving it slightly in the state-space where the optimization objective tends to increase. One of the baselines in our experiments is the immune based particle swarm optimization (IMPSO) method [4]. This is because IMPSO uses candidate affinity to prevent poor candidates from being discarded in subsequent iterations, allowing it to surpass other PSO based methods in terms of execution costs and average response time. Other techniques, categorized commonly as list scheduling, use metrics like earliest finish time, critical path, and dynamic resource utilization levels [28], [6]. However, list scheduling performs poorly in settings with non-preemptable jobs and heterogeneous requirements or machines [6].

Other recent methods use genetic algorithms to optimize the scheduling decision. Most such works use a neural network to get an estimate of the QoS objectives [5], [15]. As in swarm based methods, the algorithm uses several candidate solutions. However, these algorithms can lead to non-local jumps in the search space due to operations like crossover and mutations. This can lead to better QoS estimates, but also tends to have higher task migration overheads as converged decisions can be very different from the decision of the previous scheduling interval [7]. For our experiments, we use ESVR [15] and DNSGA [5] as our baselines from this class. ESVR initializes its candidate population using the Heterogeneous Earliest Finish Time (HEFT) heuristic and optimizes using the crossover-mutation scheme. To account for volatility in the system, ESVR continuously fine-tunes the neural network surrogate using the latest workload traces and host characteristics [15]. DNSGA is a multi-objective optimization method that uses a Pareto Optimal Front (POF) aware approach that prevents the set of candidates from converging to the same optima [5]. Ismayilov et al. and Pham et al. show that these two methods out-perform previously proposed genetic algorithms based techniques and hence are used as baselines in our experiments [5], [15].

Finally, another recently proposed workflow scheduling model, namely Closure, uses an attack-defence game-theoretic formulation [16]. Unlike other schemes that assume mostly homogeneous resources, Closure has been shown to efficiently manage heterogeneous devices by calculating the Nash Equilibrium of the attack-defence game model. This is crucial in edge-cloud environments where there are contrasting resource capacities of edge and cloud nodes. This method is also one of the baselines in our experiments.

3 System Model and Problem Formulation

3.1 Environment Model

In this work, we assume a typical mobile edge-cloud computing setup with multiple heterogeneous edge or cloud nodes in a master-slave fashion [3], [29]. We give an overview of the MCDS system model in Figure 1. The resource layer in our system includes edge nodes in the same local area network as the edge-cloud broker and cloud machines at a multi-hop distance from the broker. The edge nodes are assumed to be mobile, whereas the nodes at the cloud are assumed to be fixed in terms of their geographical location. All data inputs and outputs are asynchronously shared over a common database (DB). We consider that workflows include a batch of tasks and their corresponding precedence constraints as a DAG. In a physical deployment, these batches are passed on from sensors and actuators to the broker using gateway devices. The broker resides in the management layer and takes all the scheduling and migration decisions for each new and active task. We also assume that the resource capacities like that of the CPU, RAM, Bandwidth and Disk of each node are known in advance, and similarly we can sample the resource consumption for each task in the environment at any time. We periodically measure utilizations of CPU, RAM, Bandwidth and Disk for each task and the corresponding workflow in the system. Further, for each task, a soft service level deadline is defined at the time the workflow is sent to the edge-cloud environment. All workflows are received from an IoT layer which includes sensors and actuators to collect data from the users and send it across to the broker via the gateway devices. The broker then decides where to allocate new tasks and where to migrate active tasks in the system, if required.

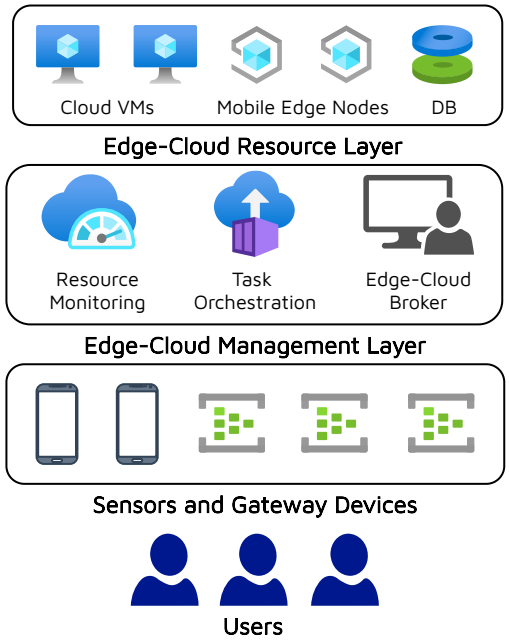


Figure 1. System Model.

3.2 Workload Model

We assume scheduling as a discrete-time control problem as common in prior work [7], [30]. We divide the timeline into equal duration intervals, with the t-th interval denoted as I_t (starting from $i = 0$). We assume that there are a fixed number of host machines, the set of which is denoted as H. Each workflow is modelled as a directed acyclic graph $G = (V, E)$, where V denotes the tasks as nodes and E denotes the execution precedence constraints as directed

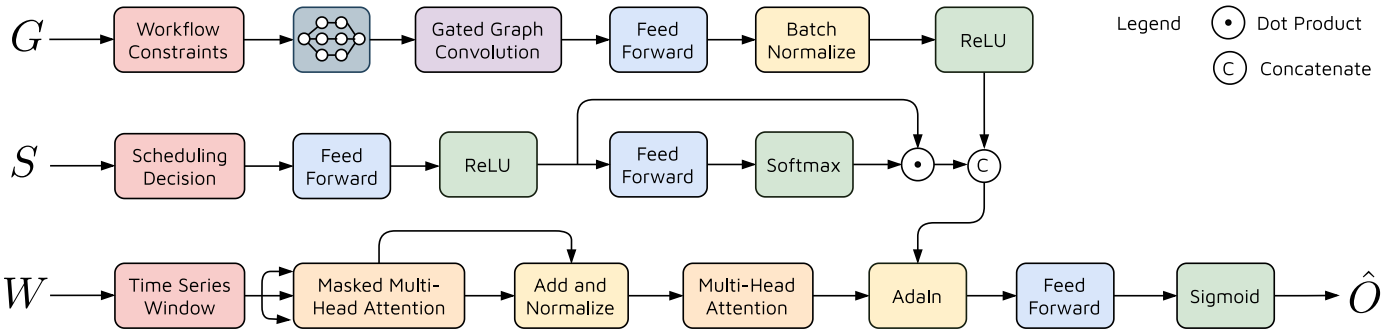


Figure 2. Deep Surrogate Model used in MCDS Scheduler. The three inputs to the model are shown in red. Feed-forward, graph convolution and attention operations are shown in blue, purple and orange. All activations are shown in green and all non-parameterized layers in yellow.

edges. Each edge (i, j) in E is an ordered pair such that task i must complete execution before task j is scheduled. We also consider that new workflows created at the interval I_t are denoted as N_t, with all active workflows being denoted as A_t. A workflow is considered to be active if at least one task of that workflow is being executed in the edge-cloud environment. If no task of a workflow $G \in N_t$ can be allocated to an edge or cloud node then it is added to a wait queue W_t. All created workflows that are not active and are not in the wait queue are considered to be completed workflows and we can calculate their QoS metrics like response time and SLA violation.

3.3 Problem Formulation

We denote the QoS objective score for interval I_t as O_t that we need to maximize. We denote the utilization metrics of all hosts in interval I_{t-1} as U_t. We collect the DAGs of all active workflows A_{t-1} and denote them as G_t (to represent the workflow constraint graph). Now using U_t and G_t, we need to predict a scheduling decision S_t. All tasks with no incomplete dependent tasks in $N_t \cup W_t \cup A_t$ are called feasible tasks. G_t includes allocation decisions for feasible and migration decisions for active tasks in the system. Thus the problem can be formulated as:

$$\begin{aligned} \underset{S_t}{\text{maximize}} \quad & \sum_{t=0}^{\infty} O_t \\ \text{subject to} \quad & \forall\, t, S_t : P_t \cup Q_t \rightarrow H, \\ & \forall\, t, P_t = \text{ set of feasible tasks in } N_t \cup W_t \cup A_t, \\ & \forall\, t, Q_t = \text{ set of active tasks in the system.} \end{aligned} \tag{1}$$

4 Deep Surrogate Model

As discussed in Section 1, our MCDS scheduler uses a deep neural network as a surrogate model. The MCDS scheduler lies in the edge-cloud broker (Figure 1). We first describe the working of this model and how we use it to obtain an estimate of the objective score. To do this, we encapsulate the state of the system using the host characteristics U_t and the workload constraints G_t. To predict an estimate of the objective score, we provide the model with the system state and the scheduling decision. Instead of using U_t, we use a time-series sliding window of size k, denoted as W_t, to capture temporal trends of the host utilization characteristics. Thus, at the start of interval I_t, the input to the surrogate model is $[G_t, S_t, W_t]$. Without loss in generality, we drop the subscript t for the rest of the discussion. An overview of the deep neural network used as a surrogate model in the MCDS scheduler is shown in Figure 2. The constituent model components are detailed next.

4.1 Workflow Constraint Graph Encoding

To generate a QoS estimate, we encode the workflow constraint graph G using a gated-graph convolution network (GGCN). This is because GGCN has been shown to be effective in capturing dynamically changing graph like inputs [31]. We first form an embedding for each node in the graph; for active tasks we use the SLA deadline and the resource utilization characteristic and for inactive tasks we use the zero-vector. We denote this embedding for a task i as e_i. Following [32], the gating stage is realized as a Gated Recurrent Unit (GRU) resulting in *graph-to-graph* updates as:

$$\begin{aligned} r_i^0 &= \text{Tanh}\left(w\, e_i + b\right), \\ x_i^k &= \sum_{j \in n(i)} w^k r_j^{k-1}, \\ r_i^k &= \text{GRU}\left(r_i^{k-1}, x_i^k\right). \end{aligned} \tag{2}$$

Here, w and b are the weight matrix and bias parameter of the neural network and k is the iteration index that varies from 1 to p, each corresponding to a graph convolution operation. Further, the messages for task i are aggregated over one-step connected neighbors $n(i)$ over p convolutions, resulting in an embedding r_i^p for each task node in the graph. The stacked representation for all tasks is represented as r^p and is used in the downstream model. We generate the workflow graph encoding e^G by passing r^p through a feed-forward layer with ReLU activation as

$$e^G = \text{ReLU}(\text{BatchNorm}(\text{FeedForward}(r^p))), \tag{3}$$

where BatchNorm standardizes the output of the FeedForward layer for the input batch.

4.2 Scheduling Decision Encoding

We also encode the scheduling decision for p tasks, first as a matrix M^S and then pass it through a feed-forward layer. Then we apply *Bahdanau style* self-attention [33] to reduce

its dimensions size and generate the scheduling decision encoding e^S. Thus,

$$\begin{aligned} M &= \mathrm{ReLU}(\mathrm{FeedForward}(M^S)), \\ attn &= \mathrm{Softmax}(\mathrm{FeedForward}(M)), \\ e^S &= \sum_{i=1}^{p} attn_i \cdot M_i. \end{aligned} \tag{4}$$

Here, M_i is the one-hot vector corresponding to the i-th tasks out of p tasks in the system. This operation takes a convex combination of the multiple scheduling decisions to create a single encoding that encapsulates the inter-task dependencies. This relieves the downstream predictors from discriminating among scheduling decisions.

4.3 Time Series Window Encoding

For any three input matrices Q, K and V, we define Multi-Head Self Attention [34] as passing it through h (number of heads) feed-forward layers to get Q_i, K_i and V_i for $i \in \{1, \ldots, h\}$, and then applying attention as

$$\begin{aligned} \mathrm{MultiHeadAtt}(Q, K, V) &= \mathrm{Concat}(H_1, \ldots, H_h), \\ \text{where } H_i &= \mathrm{Attention}(Q_i, K_i, V_i), \end{aligned} \tag{5}$$

where $\mathrm{Attention}(Q_i, K_i, V_i)$ is the scaled-dot product attention operation [34]. For a time-series input of host resource utilization characteristics, we perform transformer like multi-head attention operations as:

$$\begin{aligned} W^1 &= \mathrm{Mask}(\mathrm{MultiHeadAtt}(W, W, W)), \\ W^2 &= \mathrm{LayerNorm}(W + W^1), \\ e^W &= W^2 + \mathrm{MultiHeadAtt}(W^2, W^2, W^2). \end{aligned} \tag{6}$$

The first attention operation is masked to prevent the encoder from looking at the datapoints for future timestamp values at the time of training as all time-series windows are given at once to allow parallel training. $\mathrm{LayerNorm}$, unlike $\mathrm{BatchNorm}$, performs normalization across the values in the input instead of the batch. Multi-head attention allows the model to make the subsequent prediction computationally simpler, enabling higher model scalability and generalizability [35]. The final window encoding obtained is e^W.

4.4 Estimating the Objective Score

Now that we have encoded representations of the host characteristics (e^W), scheduling decision (e^S) and the workflow tasks (e^G), we use Adaptive Instance Normalization (AdaIn) [36] to normalize our window encoding as:

$$\begin{aligned} e^{GS} &= [e^G; e^S], \\ e &= \mathrm{AdaIn}(e^W, e^{GS}). \end{aligned} \tag{7}$$

Here, e^{GS} is the concatenation of e^G and e^S. Motivated from style-transfer neural networks, AdaIn adaptively normalizes the time-series encoding using the workflow and scheduling decision encodings. This allows the model to maintain robustness in scenarios with highly time-varying host characteristics. The final encoding is then passed through a feed-forward layer to generate the objective score estimate

$$\hat{O} = \mathrm{Sigmoid}(\mathrm{FeedForward}(e)). \tag{8}$$

Unlike traditional deep surrogate model based schedulers, the output of our model is an estimate of the long-term objective score instead of the short-term (single scheduling interval) score. This allows the scheduler to avoid local optima when taking sequential scheduling decisions [21]. We denote this surrogate model as a function $f(G, W, S) = \hat{O}$. To train this model, we use a Monte Carlo approach as described next.

5 Monte Carlo Learning

We develop a modified version of the Monte Carlo Tree Search (MCTS) to train our deep surrogate model and also actively take scheduling decisions. MCTS is a method of building a search tree considering the current state as the root node [37]. The nodes in this tree denote states of the system and edges denote actions. Each node is assumed to be either a leaf node or have one or more child nodes corresponding to the feasible actions from this node[1]. For our Monte Carlo tree, we consider that each node is defined by the following set of variables: $[G, W, S, q, v, n]$. Here, G, W and S represent the workflow constraints, time-series window and last scheduling decision; q, v, n denote the short-term reward, *value function* and the number of visits to the node. The value function is the long-term estimate of the QoS for the node. Building this tree involves finding the feasible actions from a state, executing several simulation steps and iteratively updating the root node with the best action.

5.1 Building the Monte Carlo Tree

The process of building the Monte Carlo tree involves iteratively performing four operations: Selection, Expansion, Simulation and Backpropagation. We start with only the root node (initial state), with q, v initialized as 0 and n initialized as 1. An overview of these operations is presented in Figure 3.

Selection. This operation selects a leaf-node of the tree. Starting at the root node, we recursively select the child node that has the highest Upper-Confidence-Bound (UCB). For the current node $[G, W, S, q, v, n]$, we select one of the child nodes $\{[G_i, W_i, S_i, q_i, v_i, n_i]\}_i$ using the rule [38]

$$\text{selected node} = \arg\max_i v_i + \sqrt{\frac{c_1 \times \ln n}{n_i}}. \tag{9}$$

Here the first term corresponds to the exploitation part of the search algorithm and the second term corresponds to exploration. $c_1 \in [0, 1]$ is the exploration parameter. The n value of each node that is selected in this recursive process is incremented by 1. When only the root node is present in the tree, we return that as the selected node.

Expansion. After the selection process, we reach a leaf node of the tree. We use a co-simulator to find out all possible actions $\{S_i\}_i$ from this node and expand it to generate a new node for each such decision. The co-simulator also provides us the G and W for the new nodes. For each new node in the tree, v, n are initialized as 0, 1 respectively. The

1. In our discussion, we use action and scheduling decision interchangeably.

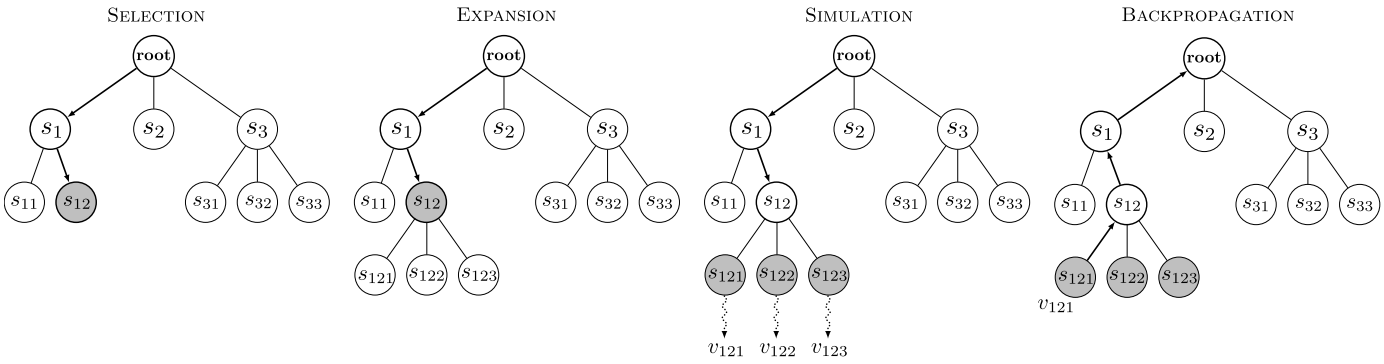


Figure 3. Building the Monte Carlo Tree. The selection operation increments the n values of the nodes. The expansion operation sets the q values using the immediate QoS score for each action. The backpropagation operation updates the v values of the nodes up to the root.

q value is set as the QoS objective score returned by the co-simulator for this decision in one scheduling interval.

Simulation. For the new nodes in our tree, we use the co-simulator to execute a sequence of ϕ actions to get a long-term QoS estimate of this node, where ϕ is called the roll-out parameter. This QoS value is set as the v value of this node. While training, the actions used in these simulations are based on a random scheduler. At test time, we use the deep surrogate model based scheduler. More details on this in Sections 5.2 and 5.3.

Backpropagation. Now that we have the updated v values of the new leaf nodes, we backpropagate these up to the root for updating the value functions. For every node $([G, W, S, q, v, n])$ other than the leaf-nodes in the path to the root node with child nodes $\{[G_i, W_i, S_i, q_i, v_i, n_i]\}_i$, we propose the weighted update rule

$$v = q + \frac{\sum_i n_i \cdot v_i}{\sum_i n_i}, \tag{10}$$

i.e., the sum of q and the weighted average of the v values of child nodes. The q value here gives us the immediate reward and weighted-average gives us the estimate of the long term reward once the action corresponding to this node is taken [38].

We perform the four operations: selection, expansion, simulation and backpropagation a fixed number of times (ψ) to generate the Monte Carlo Tree. The expansion operation adds possibly one or more nodes to the tree in each iteration. The UCB metric in the selection phase trades off between exploration of new action choices and exploitation of value function estimates in each iteration. As ψ increases, the value function estimates become closer to the expected empirical objective scores.

5.2 Model Training

To train the deep surrogate model, at every interval I_t we build the tree as described in Section 5.1 as shown by the MCTS-TRAIN function in Algorithm 1. (we start with a single node in I_0). We use a random scheduler to execute the *simulation* operation. The random scheduler gives us an unbiased estimate of the long-term value function of a leaf-node (line 4 in Alg. 1). After performing the operations ψ number of times, we choose the action S_i corresponding to child node of the root with the highest value-function

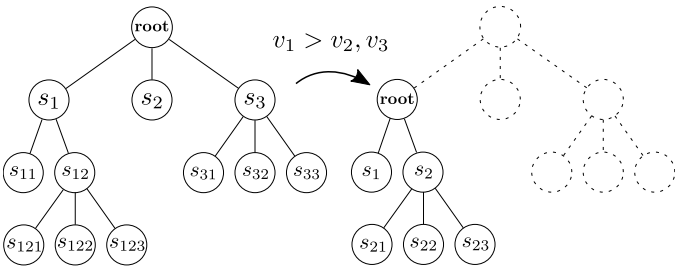


Figure 4. Choosing best action and updating the Monte Carlo Tree after simulations.

Algorithm 1 Training the deep surrogate model

Require:
Number of simulation steps ψ; Rollout parameter ϕ; Dataset used for training Λ; Deep Surrogate Model f

1: Initialize $root$ node
2: **procedure** MCTS-TRAIN($root$)
3: **for** $i \leftarrow 0$ to ψ **do**
4: Recursively select child nodes using Eq. (9)
5: Expand leaf-node $\forall$ possible actions
6: Using random policy, run ϕ-step co-simulations to generate QoS scores
7: Recursively back-propagate scores using Eq. (10)
8: **procedure** UPDATE($root$)
9: $[G, W, S, q, v, n] \leftarrow root$
10: $\{[G_j, W_j, S_j, q_j, v_j, n_j]\}_j \leftarrow$ child nodes of $root$
11: $i = \arg\max_j v_j$
12: Add $(f(G, W, S), O_t + v_i)$ to Λ
13: $root \leftarrow [G_i, W_i, S_i, q_i, v_i, n_i]$
14: **return** S_i
15: **procedure** TRAIN(scheduling interval I_t)
16: MCTS-TRAIN($root$)
17: $S_i \leftarrow$ UPDATE($root$)
18: Tune f using the dataset Λ using (11) as loss
19: **return** S_i

v_i (line 11 in Alg. 1). For training, instead of using the q_i value of this node, we get the QoS score in the next interval I_{t+1} *i.e.* O_t and add the datapoint $(f(G, W, S), O_t + v_i)$ to the dataset (line 12 in Alg. 1). For the next interval, we update the Monte Carlo tree by changing the root node to the selected child node (see Figure 4 and UPDATE function

Algorithm 2 MCDS Scheduler

Require:
Number of simulation steps ψ; Learning rate γ;
Convergence threshold ϵ; Iteration limit L
Pre-trained Deep Surrogate Model f

1: Initialize $root$ node
2: **procedure** MCTS-TEST($root$)
3: **for** $i \leftarrow 0$ to ψ **do**
4: Recursively select child nodes using Eq. (13)
5: Expand leaf-node $\forall$ possible actions
6: Use $f(G, W, S)$ as generate QoS scores
7: Recursively back-propagate scores using Eq. (10)
8: **procedure** GOBIGRAPH(f, G, S, W)
9: Initialize $i = 0$
10: **do**
11: $S \leftarrow S + \gamma \cdot \nabla_S f(G, W, S)$
12: $i \leftarrow i + 1$
13: **while** $|\nabla_S f(G, W, S)| > \epsilon$ and $i \leq L$
14: **return** S
15: **procedure** MCDS(scheduling interval I_t)
16: MCTS-TEST($root$)
17: $S_i \leftarrow$ UPDATE($root$)
18: Tune f with the new datapoint using (11) as loss
19: **return** S_i

in Alg. 1). This allows us to re-use co-simulation data in the subsequent scheduling intervals.

Using the collected dataset, we use experience-replay [39] to decouple training from the execution of the workloads on our edge-cloud setup. We update the weights of the deep surrogate model using the loss:

$$\text{Loss} = \text{MSE}(f(G, W, S), O_t + v_i), \tag{11}$$

where MSE denotes mean-square-error and G, W are the workflow constraints and time-series host characteristics of the root node state. We use a cross-validation style training process with early-stopping as the convergence criterion.

5.3 Scheduling using the Trained Model

Now that we have trained the deep surrogate model, for an input $[G, W, S]$, we can run the recently developed technique of gradient-optimization using propagation to input (GOBI) [7] to update S to maximize the output $f(G, W, S)$ using gradient-ascent rule

$$S \leftarrow S + \gamma \cdot \nabla_S f(G, W, S). \tag{12}$$

Using advances like momentum, annealing and restarts we can reach near global optima quickly [7]. To differentiate our workflow-graph based surrogate model from vanilla feed-forward neural network, we call our optimization strategy as GOBIGRAPH that returns an action in the form of a scheduling decision (see GOBIGRAPH function in Alg. 2). At test time we build the Monte Carlo Tree with slight modifications. In the selection operation, we choose the node using the rule

$$\text{selected node} = \arg\max_i v_i + \sqrt{\frac{c_1 \times \ln n}{n_i}} + c_2 \times \mathbb{1}(S_i = \text{GOBIGRAPH}(f, G, S, W)). \tag{13}$$

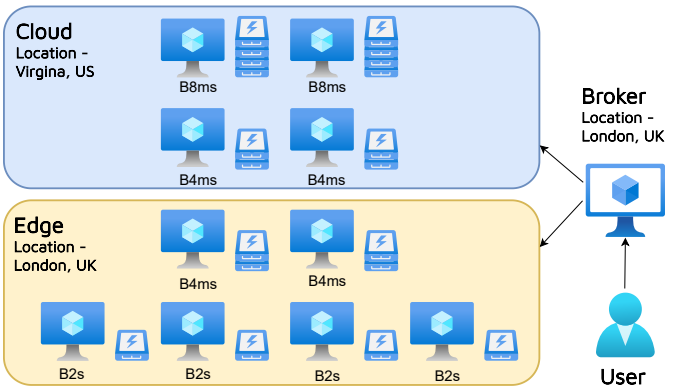


Figure 5. Evaluation Setup.

Here, the third term uses the deep surrogate model to bias our selection choice, like a domain knowledge term commonly used in machine learning [37]. $c_2 \in [0, 1]$ is the bias parameter. When performing the simulation operation, we use the value estimate $v = f(G, W, S)$ instead of running expensive co-simulations, reducing the overall scheduling time (line 6 in Alg. 2). Using the new datapoint in each interval, we also fine-tune the deep surrogate model f in MCDS to adapt in dynamic settings (line 18 in Alg. 2).

6 PERFORMANCE EVALUATION

We now describe how we evaluate the MCDS method and compare it against the state-of-the-art baselines: ESVR, DNSGA, IMPSO and Closure as described in Section 2. We also add a vanilla GOBI method as a reference point for our results to compare against the state-of-the-art deep surrogate model based scheduler.

6.1 Evaluation Setup

To test the scheduling strategies, we use two-test beds: physical setup and a simulated platform. The former allows us to perform a precise comparison closer to the real-deployments, while the latter lets us perform large-scale experiments. This is a common testing mechanism in prior work [7], [30].

Physical Testbed: We use Microsoft Azure virtual machines (VMs) to create a heterogeneous PDC edge-cloud testbed[2]. Specifically, we use the B2s. B4ms and B8ms machine types. The configuration consists of 10 VMs, 6 light-weight VMs on the edge and 4 powerful ones on a geographically distant cloud location (see Figure 5). All machine types use a different number of Intel Haswell 2.4 GHz E5-2673 v3 processor cores. To keep storage costs consistent, we keep Azure P15 Managed disk with 125 MB/s disk throughput and 256 GB size[3]. We use the Microsoft Azure pricing calculator to obtain the cost of execution per hour (in US Dollars)[4]. The power consumption models are

2. Azure general purpose B-type VMs: https://docs.microsoft.com/en-us/azure/virtual-machines/sizes-b-series-burstable
3. Azure Managed Disk types https://docs.microsoft.com/en-us/azure/virtual-machines/disks-types#premium-ssd
4. Microsoft Azure pricing calculator for South UK https://azure.microsoft.com/en-gb/pricing/calculator/

taken from the SPEC benchmarks repository[5]. The Million-Instruction-per-Second (MIPS) of all VMs are obtained using the `perf-stat`[6] tool.

The Azure B2s machines consist of two cores (4029 MIPS), B4ms have four (8102 MIPS) and B8ms have eight cores (1601 MIPS) each. Moreover, we have 4GB, 16GB and 32GB RAM sizes in B2s, B4ms and B8ms machines respectively. In our settings, we have considered response time of edge nodes to be 1 ms and that of cloud nodes to be 10 ms based on the empirical studies using the Ping utility in an existing edge-cloud framework namely FogBus [3]. To factor in the possible mobility of the edge nodes we use the `NetLimiter` tool to tweak the communication latency with the broker node using the mobility model described in [40]. Specifically, for the edge nodes in our setup, we use the latency and bandwidth parameters of hosts from the traces generated using the Simulation of Urban Mobility (SUMO) tool [41] that emulates mobile vehicles in a city like environment. SUMO gives us the parameters like `ping` time and network bandwidth to simulate in our testbed using `NetLimiter`. The edge-cloud broker node was a system situated in London, UK and with configuration: Intel i7-10700K CPU, 64GB RAM, and Windows 11 OS.

Simulated Testbed: We consider 50 host machines as a scaled up version of the physical setup. Here, each category has 5 times the instance count, *i.e.*, 30 edge nodes and 20 cloud nodes. This is to test models in a larger-scale edge-cloud environment as considered in prior art [30], [42]. We use a trace driven simulation model that emulates real execution of tasks, but without actual job execution allowing us to perform experiments on scale.

We run all experiments for 200 scheduling intervals, with each interval being 300 seconds long, giving a total experiment time of 16 hours 40 minutes. We average over 5 runs and use diverse workload types to ensure statistical significance in our experiments.

6.2 Workloads

For our physical setup, we use *WFCommons* benchmark applications [43] which are derived from Pegasus workflows [44]. These include the popular non-preemptive scientific applications: BLAST, Cycles and Montage, each having 15 to 100 tasks sampled uniformly at random. The BLAST workflow is a high-throughput compute-intensive application for genome based bioinformatics calculations and has applications in smart health monitoring systems at the edge [45]. The Cycles benchmark performs several agro-ecomonic modelling based calculations and is popular in edge computing based smart-agriculture [46]. Montage is a benchmark derived from astronomical imaging systems and similar to other edge computing based space exploration applications [47]. The Pegasus workflows are popular benchmark applications that are representative of a large number of edge and cloud computing environments. These have been used in several previous works that test workflow scheduling in edge-cloud environments [48], [49], [50]. Due to the these benchmarks being standard in most prior work, we use them in our experiments.

For our simulator, we use the execution traces of the three applications and the precedence graphs from the benchmark specifications. These traces are generated by running the experiments on the 10 node physical setup. The traces consist of data intervals of 5 minutes including the CPU, RAM, Disk and Bandwidth utilization of all tasks. We sample monitoring data uniformly from the three applications as done in the *WFCommons's* workload generator.

At the beginning of each scheduling interval we create $Poisson(\lambda)$ workflows with $\lambda = 1.2$ workflows for physical and $\lambda = 5$ workflows for the simulated setup [7]. These values have been taken directly from traces generated from real-world computing applications deployed on Bitbrains datacenters with 10 and 50 servers [51]. The 1.2 and 5 values have been obtained after normalizing the computational requirements of tasks using the total instructions executed for each task, calculated as the product of average MIPS and execution time. This is done to ensure that the experimental testbed is under similar load as typical real-world physical deployment.

6.3 Implementation Details

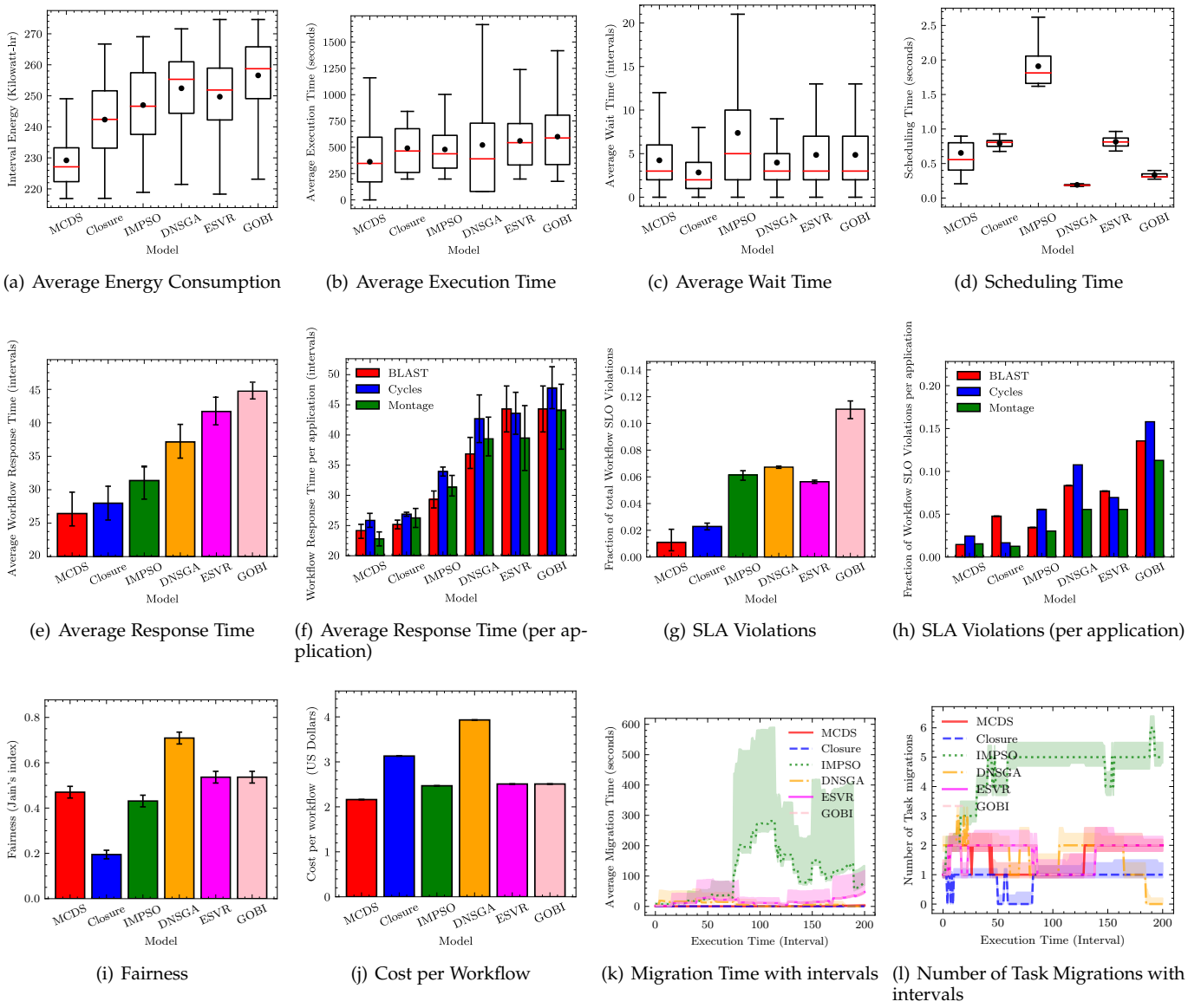
We perform all our experiments on an extended version of the COSCO framework [7] to support workflow DAG models. COSCO [7] is a framework for development and deployment of applications on edge-cloud environments with structured communication and platform independent execution of applications. COSCO connects all edge and cloud nodes in a physical setup using HTTP REST APIs. We deploy our workflows in COSCO with each task mapped to a docker container.

To implement MCDS in the COSCO framework, we extend the `Framework` class. The function `getPlacementPossible()` was modified to also check for precedence constraints when allocating tasks. Moreover, we implemented a data transferring pipeline for broadcasting of inputs for parallel tasks and forwarding of the outputs between two neighboring tasks in a DAG. Finally, the outputs were synchronized and brought to the broker to calculate the inference accuracy and measure the workflow response time. For synchronization of outputs and execution of network splits, we use the HTTP Notification API. Further, the GOBIGraph approach is implemented using the Autograd package in the PyTorch module [52].

To run an experiment, the broker runs the COSCO framework and communicates with each host via HTTP REST APIs [53]. The REST server of the `Framework` class sends operation codes including `GetHostStats`, `GetContainerStats`, `Checkpoint`, `Migrate` and `Restore` to each host to instruct a specific activity. At every worker host, a `DockerClient` service runs with a `Flask` HTTP web-service [54]. Flask is web application REST client that allows interfacing between edge-cloud hosts and the broker. To obtain the resource utilization metrics of the host machines at the start of each scheduling interval, we use the `psutil` utility for memory and disk characteristics, `perf stat` for CPU utilization and `ioping` utility for memory

5. Standard Performance Evaluation Corporation (SPEC) benchmark repository `https://www.spec.org/cloud_iaas2018/results/`

6. `perf-stat` tool `https://man7.org/linux/man-pages/man1/perf-stat.1.html`



(a) Average Energy Consumption (b) Average Execution Time (c) Average Wait Time (d) Scheduling Time

(e) Average Response Time (f) Average Response Time (per application) (g) SLA Violations (h) SLA Violations (per application)

(i) Fairness (j) Cost per Workflow (k) Migration Time with intervals (l) Number of Task Migrations with intervals

Figure 6. Comparison of MCDS against baselines on physical setup with 10 hosts. Box plots: dot represents mean and solid red line represents median. Error bars in the bar plots correspond to 90% confidence-interval. Shaded regions in the line plots show the range of the values over five runs.

and network bandwidth utilization characteristics. For the running tasks, we use the `docker inspect` command with the identifier of the corresponding container instance as an argument. All utilization metrics are instantly saved to an InfluxDB database instance running at the broker node. This additional step of collecting utilization metrics leads to an overhead due to the HTTP REST request and adding a new entry in the database. However, this occurs in a non-blocking fashion, *i.e.*, it does not interrupt the execution of active containers in worker hosts.

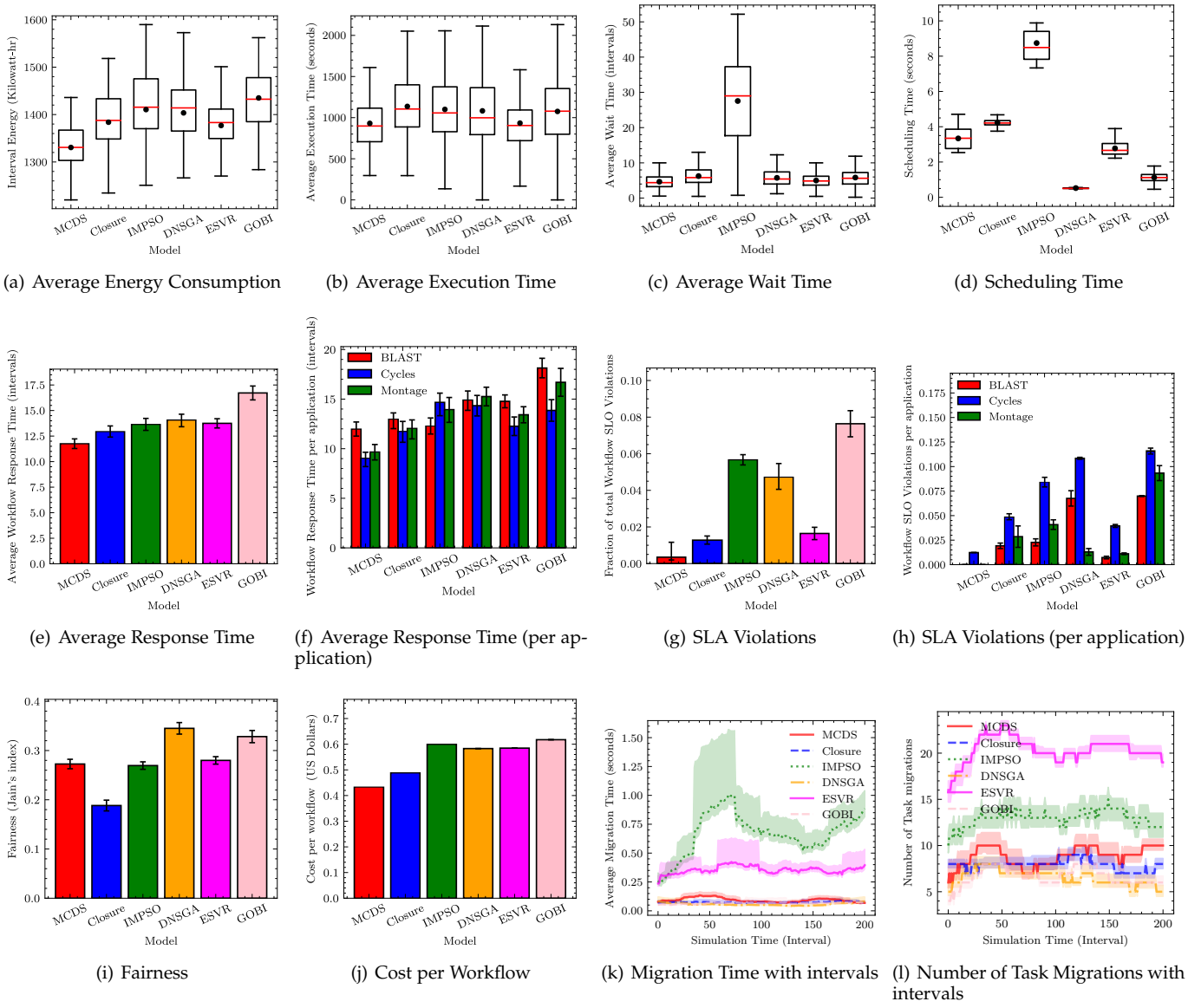
6.4 Model and Experimental Assumptions

To optimize the QoS parameters, we consider a standard objective function that focuses on energy consumption and response time as is done in prior work [7]. For interval I_t,

$$\mathcal{O}_t = 1 - \alpha \cdot AEC_t - (1-\alpha) \cdot ART_t, \qquad (14)$$

where AEC_t and ART_t are the normalized average energy consumption of the environment in I_t and average response time of workflows leaving at the end of I_t. Here, α is a hyper-parameter that can be set by users as per the application requirements. We choose response time as a metric to optimize the service quality by reducing the latency of the executed workflow applications and subsequently the SLA violation rates [2]. Energy is another important metric, both for edge and cloud nodes. At the edge, devices are typically mobile, relying on battery operated hardware. For long-lasting services and reducing the average down-times of edge nodes, supreme energy efficiency is crucial. For cloud service providers, higher energy consumption leads to increased operation expenses, both for running the cloud servers as well as the cooling equipment [32]. The α parameter is chosen the user as per the application demands.

For our experiments, we keep $\alpha = 0.5$ in (14) as it is a characteristic representative of typical fog environments [7], [40]. We also set $\phi = 10$ (number of actions in simulations for model training) and $\psi = 50$ (number of MCTS itera-



(a) Average Energy Consumption (b) Average Execution Time (c) Average Wait Time (d) Scheduling Time

(e) Average Response Time (f) Average Response Time (per application) (g) SLA Violations (h) SLA Violations (per application)

(i) Fairness (j) Cost per Workflow (k) Migration Time with intervals (l) Number of Task Migrations with intervals

Figure 7. Comparison of MCDS against baselines on simulated setup with 50 hosts.

tions). We also use $c_1 = 0.1, c_2 = 0.5$. All hyper-parameter values are obtained using grid-search. We get similar trends for other values. We present a sensitivity analysis on ψ later in Section 6.9. To train our deep surrogate model, we use we the AdamW optimizer [55] with learning rate as 0.0001.

6.5 Evaluation Metrics

To compare the QoS, we consider metrics like energy consumption and response time. We also compare the SLA violation rates. The SLA of a workflow is violated if its response time is greater than the deadline. We consider the relative definition of SLA (as in [7]) where the deadline is the 98^{th} percentile response time for the same application (BLAST/Cycles/Montage) on the state-of-the-art baseline *Closure*. To ensure sufficient sample size, we run the *Closure* baseline for 1000 scheduling intervals and evaluate the 98^{th} percentile response times for each application. We also consider scheduling fairness, for which we use Jain's fairness index over the IPS of the running tasks in the system. We also compare the average migration time which is the average time for task (container) migrations. We also compare the scheduling time, execution cost and average waiting time (number scheduling intervals spent in the wait queue) of all executed workflows.

6.6 Results

We show the results of our experiments on the physical and simulated testbeds in Figures 6 and 7 respectively. Figures 6(a) and 7(a) show that the MCDS model has the lowest energy consumption compared to the baseline methods. MCDS scheduler gives an average interval energy consumption of 228.21 KW-hr for 10 hosts and 1326.98 KW-hr for 50 hosts. As the number of hosts increases so does the power consumption of the setup. However, MCDS has up to 6.13% lower energy consumption than the second lowest value of the Closure scheduler. The major reason for this is the low response time of each application, which leads to lower number of active tasks in the system at any point

giving low CPU utilization and consequently low energy consumption. Figures 6(b) and 7(b) show the execution times of the workflows, *i.e.*, the time these workflows spend in computation. All schedulers have similar execution times for both setups. Figures 6(c) and 7(c) show the waiting times of workflows. Here, IMPSO has the highest waiting times as using this scheduler leads to the maximum number of resource contentions in the system. The scheduling times of all schedulers are less than 10 seconds (3.33% of the scheduling interval duration of 5 minutes), with DNSGA and GOBI having the lowest scheduling times (Figures 6(d) and 7(d)). MCDS has 13.19% lower scheduling time than the best performing baseline, *i.e.*, Closure. A higher scheduling time compared to the meta-heuristic approach DNSGA is due to the multiple Monte-Carlo iterations. However, the model is faster than other approaches like Closure and IMPSO, showing that MCDS is scalable to larger testbeds. More details on scalability with workload size in Section 6.8.

Figures 6(e) and 6(e) show the average response times for both setups. Among the baselines, Closure has the lowest average response time of 27.21 intervals in the physical setup and 12.67 intervals in the simulated case (lower response time in the simulated setup due to the larger parallelization capacity). MCDS has a response time up to 4.56% lower than Closure. This is due to the long-term optimization in MCDS, unlike other methods. Myopic optimization in prior work typically prevents them to consider future system states that could cause an increase in response time due to resource contention or migrations. This allows MCDS to be parsimonious in terms of number of preemptive migrations (Figs. 6(l) and 7(l)). Comparing the number of migrations across baselines, we see that IMPSO and ESVR very aggressively migrate tasks, whereas the other methods try to minimize migration overheads to optimize QoS. Low response times of the MCDS scheduler also results in low SLA violation rates (here the service level objectives are defined as deadlines as discussed in Section 6.5) as seen in Figures 6(g) and 7(g). MCDS has an SLA violation percentage of 0.25%-1.12%, lower than the best values among the baselines, which are 1.12%-2.04% for the Closure scheduler. IMSO has a high SLA violation rates due to the high waiting and migration times as seen in Figures 6(k) and 7(k). This is due to the aggressive migration in IMPSO as seen in Figures 6(l) and 7(l). ESVR has relatively low SLA violation rates for its response times due to the deadline first prioritization in the scheduling decisions. Figures 6 and 7 also show the response times and SLA violation rates for each application type (BLAST, Cycles and Montage). The SLA violation rates for each application is shown in Figures 6(h) and 7(h). The deviations from the 2% mark in these figures is due to the stochasticity in the workloads, motivating us to fine-tune the model at each interval (line 18 in Alg. 2).

Comparing the fairness indices of the schedulers (Figures 6(i) and 7(i)), we see that the DNSGA has the maximum fairness index of 0.34-0.72. MCDS has a fairness index of 0.28-0.52. This is due to the long-term optimization in MCDS, preventing resource contention and allocation of resources to tasks corresponding to their requirements. As MCDS is able to complete the maximum number of workloads in the duration of 200 intervals, the amortized cost per workflow of MCDS is lowest among all models (0.43 and 2.21 USD/workflow). MCDS gives 16.12%-30.71% lower cost than the baselines with ten hosts (Figure 6(j)). Similar trends are seen in the simulated setup with 50 hosts (7.62%-70.14%).

Table 1
Ablation Analysis. Units: Energy - *KW-Hr*, Response Time - scheduling intervals, SLA Violations - percentage, Scheduling Time (cumulative) - seconds.

Model	Energy	Resp. Time	SLA Violation	Sched. Time
MCDS	**1326**	**12.08**	**0.468**	3.3627
w/o MCTS	1479	14.21	4.283	**0.3123**
w/o DSM	1401	13.20	3.027	7.4231
w/o Exploration	1392	12.98	0.832	2.6283
w/o Domain Kn.	1532	19.41	0.930	2.0318

6.7 Ablation Analysis

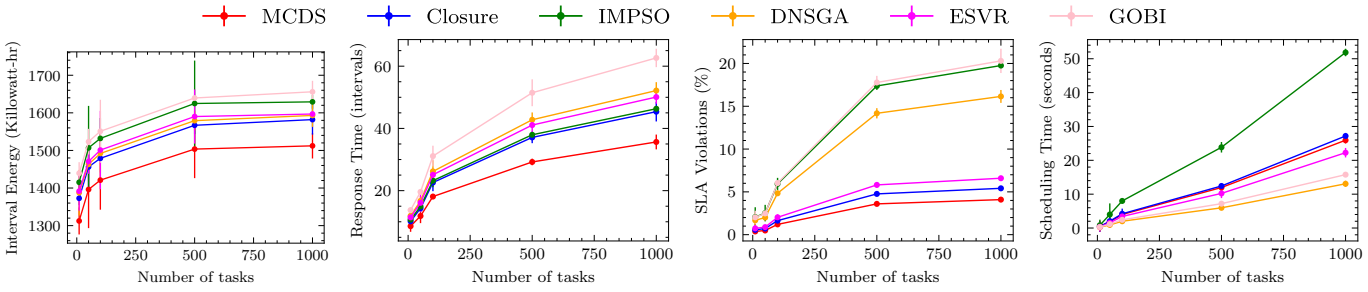
To study the relative importance of each component of the model, we exclude every major one and observe how it affects the performance of the scheduler. An overview of this ablation analysis is given in Table 1. First, we consider the MCDS scheduler without the Monte-Carlo tree search, *i.e.*, only using the pre-trained deep surrogate model to get the GOBIGraph output (*w/o MCTS* model). Second, we consider a model without the deep surrogate model, *i.e.*, we only use MCTS simulations with a random scheduler to get the QoS estimates and generate the scheduling decision at each interval (*w/o DSM* model). The other two models we consider are modifications in the selection formula. In the *w/o Exploration* model we ignore the UCB exploration metric. In the *w/o Domain Kn.* we ignore the GOBIGraph based indicator term in (13). We reach the following findings:

- Without the MCTS search scheme, the model performs a myopic optimization and suffers the maximum increase in the SLA violation rate. This demonstrates the need for a long-term objective estimation and bootstrapping is crucial to reach optimal QoS.
- Without the deep-surrogate model, the scheduling time significantly increases. This is due to the large number of co-simulation steps being performed in the MCTS iterations. It also affects the QoS values due to the inaccurate estimates obtained from the simulations based on the random scheduler.
- Without the exploration term the model is unable to find possibly better scheduling decisions. Also, the domain knowledge term, *i.e.*, the GOBIGraph based indicator term in (13) helps the model to use the deep surrogate model for exploration. Both these terms help improve the scheduling decisions and reach better QoS values.

We observe that Monte-Carlo learning and deep surrogate model have the maximum impact to the performance. UCB exploration and domain knowledge are additional features that help improve QoS.

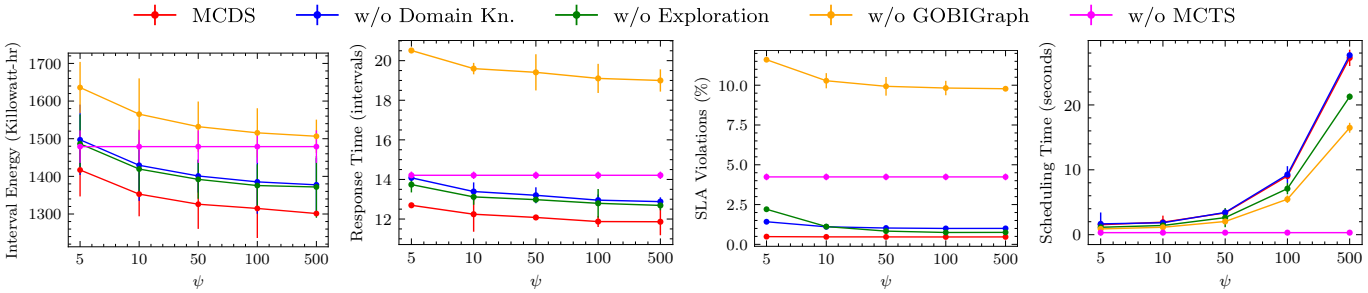
6.8 Scalability Analysis

The *WfGen* utility within *WFCommons* benchmark suite allows us to change the number of tasks in each workflow



(a) Energy Consumption (b) Response Time (c) SLA Violations (d) Scheduling Time

Figure 8. Scalabilty Analysis for MCDS and baseline methods with the number of tasks in each workflow as 10, 50, 100, 500 and 1000.



(a) Energy Consumption (b) Response Time (c) SLA Violations (d) Scheduling Time

Figure 9. Sensitivity Analysis for ablated models with number of Monte Carlo iterations (the ψ parameter in our formulation).

DAG.[7] In our main experiments we sample the number of jobs uniformly at random between 15 and 100. We now use this feature to test the scalability of the MCDS model and compare it against the baselines on the simulated setup with 50 hosts. Figure 8 shows the energy consumption, response time, SLA violations and scheduling time when we vary the number of tasks in each workflow from 10 to 1000. Comparing across models, the DNSGA, ESVR and IMPSO do not scale well in terms of SLA violation rates as the number of tasks grows. Only models that use deep surrogate networks, *i.e.*, MCDS and GOBI, and the attack-defense based scalable baseline Closure are able to scale with increasing task count. Similarly, other parameters, *i.e.*, energy consumption, response time and scheduling time increase with number of tasks. In terms of energy consumption, response time, SLA violation rate and scheduling time, MCDS not only has better metric scores, but also does not face from excessive performance drop in cases where workflows have a large number of tasks, thanks to its gradient graph convolution based adaptive normalization. The deep surrogate model in MCDS is able to handle large-scale workflows also due to the *Bahdanau* attention that brings down the dimensional size of the input size windows.

6.9 Sensitivity Analysis

We check the sensitivity of the performance parameters to the ψ parameter (number of Monte-Carlo iterations), as shown in Figure 9. The figure shows the energy consumption, response time, SLA violations and scheduling time when ψ varies from 5 to 500. As the number of iterations increases, the model can cover larger state-action space and get a much better look-ahead estimate. Thus, the energy consumption, response time and SLA violations decrease as ψ increases (except the *w/o MCTS* model which is unaffected as MCTS is not performed here). Although, after $\psi = 50$ the improvement in the QoS performance is minimal. However, increasing the simulation steps has an adverse impact on the scheduling time of the model. We choose $\psi = 50$ in our experiments to balance the trade-off between QoS and scheduling overhead.

7. WfGen - WfCommons `https://wfcommons.org/generator`.

7 Conclusions and Future Work

We have presented MCDS, a novel scheduler for efficiently scheduling demanding workflow applications on mobile edge-cloud computing environments. Unlike prior work that performs myopic optimization of QoS using reinforcement learning, evolutionary methods or heuristics, MCDS trains a long-term QoS estimation surrogate model. To do this, MCDS uses a modified version of the Monte-Carlo-Tree-Search that uses UCB exploration and gradient-optimization based biased exploitation. This allows MCDS to manage workflows to maintain high inference objective scores on an average and reduce the scheduling time as it does not need to perform several multi-step simulations at test time. All these contributions allow MCDS to be scalable and outperform the state-of-the-art models in terms of energy consumption, response time, SLA violation rate and cost by up to 6.13%, 4.56%, 45.09% and 30.71% respectively in a heterogeneous mobile edge-cloud environment with real-world workloads.

In some application scenarios, tasks in the same workflow DAG use the same input files for processing. Thus, allocating such tasks on the same host could help reduce data transfer overheads. In the future, we wish to extend

the proposed model to also consider data transfer overheads when scheduling workflows. We also propose to tune the MCDS approach for more recent IoT based workloads that are based on augmented reality, video processing and Big Data analytics applications [56].

Software Availability

The code is available at `https://github.com/imperial-qore/MCDS`. The Docker images used in the experiments are available at `https://hub.docker.com/u/shreshthtuli`.

Acknowledgments

Shreshth Tuli is supported by the President's PhD scholarship at Imperial College London.

References

[1] R. Xu, Y. Wang, Y. Cheng, Y. Zhu, Y. Xie, A. S. Sani, and D. Yuan, "Improved particle swarm optimization based workflow scheduling in cloud-fog environment," in *International Conference on Business Process Management*. Springer, 2018, pp. 337–347.

[2] S. S. Gill, S. Tuli, M. Xu, I. Singh, K. V. Singh, D. Lindsay, S. Tuli, D. Smirnova, M. Singh, U. Jain *et al.*, "Transformative effects of IoT, Blockchain and Artificial Intelligence on cloud computing: Evolution, vision, trends and open challenges," *Internet of Things*, vol. 8, pp. 100–118, 2019.

[3] S. Tuli, R. Mahmud, S. Tuli, and R. Buyya, "Fogbus: A blockchain-based lightweight framework for edge and fog computing," *Journal of Systems and Software*, vol. 154, pp. 22–36, 2019.

[4] P. Wang, Y. Lei, P. R. Agbedanu, and Z. Zhang, "Makespan-driven workflow scheduling in clouds using immune-based pso algorithm," *IEEE Access*, vol. 8, pp. 29 281–29 290, 2020.

[5] G. Ismayilov and H. R. Topcuoglu, "Neural network based multi-objective evolutionary algorithm for dynamic workflow scheduling in cloud computing," *Future Generation computer systems*, vol. 102, pp. 307–322, 2020.

[6] M. Adhikari, T. Amgoth, and S. N. Srirama, "A survey on scheduling strategies for workflows in cloud environment and emerging trends," *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–36, 2019.

[7] S. Tuli, S. R. Poojara, S. N. Srirama, G. Casale, and N. R. Jennings, "COSCO: Container Orchestration Using Co-Simulation and Gradient Based Optimization for Fog Computing Environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 1, pp. 101–116, 2022.

[8] R. Ding, X. Li, X. Liu, and J. Xu, "A cost-effective time-constrained multi-workflow scheduling strategy in fog computing," in *International Conference on Service-Oriented Computing*. Springer, 2018, pp. 194–207.

[9] K. Matrouk and K. Alatoun, "Scheduling algorithms in fog computing: A survey," *International Journal of Networked and Distributed Computing*, vol. 9, no. 1, pp. 59–74, 2021.

[10] A. Machen, S. Wang, K. K. Leung, B. J. Ko, and T. Salonidis, "Migrating running applications across mobile edge clouds: poster," in *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, 2016, pp. 435–436.

[11] A. J. Ben Ali, Z. S. Hashemifar, and K. Dantu, "Edge-slam: Edge-assisted visual simultaneous localization and mapping," in *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, 2020, pp. 325–337.

[12] J. Haussmann, W. Blochinger, and W. Kuechlin, "Cost-optimized parallel computations using volatile cloud resources," in *International Conference on the Economics of Grids, Clouds, Systems, and Services*. Springer, 2019, pp. 45–53.

[13] Y. Choi, S. Park, and H. Cha, "Optimizing energy efficiency of browsers in energy-aware scheduling-enabled mobile devices," in *The 25th Annual International Conference on Mobile Computing and Networking*, 2019, pp. 1–16.

[14] J. Yu, R. Buyya, and C. K. Tham, "Cost-based scheduling of scientific workflow applications on utility grids," in *First International Conference on e-Science and Grid Computing (e-Science'05)*. Ieee, 2005, pp. 8–pp.

[15] T.-P. Pham and T. Fahringer, "Evolutionary multi-objective workflow scheduling for volatile resources in the cloud," *IEEE Transactions on Cloud Computing*, 2020.

[16] Y. Wang, Y. Guo, Z. Guo, T. Baker, and W. Liu, "Closure: A cloud scientific workflow scheduling algorithm based on attack–defense game model," *Future Generation Computer Systems*, vol. 111, pp. 460–474, 2020.

[17] E. N. Alkhanak, S. P. Lee, and S. U. R. Khan, "Cost-aware challenges for workflow scheduling approaches in cloud computing environments: Taxonomy and opportunities," *Future Generation Computer Systems*, vol. 50, pp. 3–21, 2015.

[18] T. A. Genez, L. F. Bittencourt, and E. R. Madeira, "Workflow scheduling for SaaS/PaaS cloud providers considering two SLA levels," in *2012 IEEE Network Operations and Management Symposium*. IEEE, 2012, pp. 906–912.

[19] M. Xu, L. Cui, H. Wang, and Y. Bi, "A multiple qos constrained scheduling strategy of multiple workflows for cloud computing," in *2009 IEEE International Symposium on Parallel and Distributed Processing with Applications*. IEEE, 2009, pp. 629–634.

[20] S. Saeedi, R. Khorsand, S. G. Bidgoli, and M. Ramezanpour, "Improved many-objective particle swarm optimization algorithm for scientific workflow scheduling in cloud computing," *Computers & Industrial Engineering*, vol. 147, p. 106649, 2020.

[21] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[22] Z. Song and E. Tilevich, "Win with what you have: Qos-consistent edge services with unreliable and dynamic resources," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2020, pp. 530–540.

[23] E.-K. Byun, Y.-S. Kee, J.-S. Kim, and S. Maeng, "Cost optimized provisioning of elastic resources for application workflows," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1011–1026, 2011.

[24] W. Chen, G. Xie, R. Li, Y. Bai, C. Fan, and K. Li, "Efficient task scheduling for budget constrained parallel applications on heterogeneous cloud computing systems," *Future Generation Computer Systems*, vol. 74, pp. 1–11, 2017.

[25] A. M. Chirkin, A. S. Belloum, S. V. Kovalchuk, M. X. Makkes, M. A. Melnik, A. A. Visheratin, and D. A. Nasonov, "Execution time estimation for workflow scheduling," *Future generation computer systems*, vol. 75, pp. 376–387, 2017.

[26] V. Singh, I. Gupta, and P. K. Jana, "A novel cost-efficient approach for deadline-constrained workflow scheduling by dynamic provisioning of resources," *Future Generation Computer Systems*, vol. 79, pp. 95–110, 2018.

[27] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in *2010 24th IEEE international conference on advanced information networking and applications*. IEEE, 2010, pp. 400–407.

[28] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE transactions on parallel and distributed systems*, vol. 13, no. 3, pp. 260–274, 2002.

[29] S. P. Ahuja, S. Mani, and J. Zambrano, "A survey of the state of cloud computing in healthcare," *Network and Communication Technologies*, vol. 1, no. 2, p. 12, 2012.

[30] D. Basu, X. Wang, Y. Hong, H. Chen, and S. Bressan, "Learn-as-you-go with megh: Efficient live migration of virtual machines," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 8, pp. 1786–1801, 2019.

[31] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[32] S. Tuli, S. S. Gill, M. Xu, P. Garraghan, R. Bahsoon, S. Dustdar, R. Sakellariou, O. Rana, R. Buyya, G. Casale *et al.*, "HUNTER: AI based Holistic Resource Management for Sustainable Cloud Computing," *Journal of Systems and Software*, 2021.

[33] D. Bahdanau, K. H. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *3rd International Conference on Learning Representations, ICLR 2015*, 2015.

[34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 6000–6010.

[35] J. B. Lee, R. A. Rossi, S. Kim, N. K. Ahmed, and E. Koh, "Attention models in graphs: A survey," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 13, no. 6, pp. 1–25, 2019.

[36] X. Huang and S. Belongie, "Arbitrary style transfer in real-time with adaptive instance normalization," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1501–1510.
[37] D. Silver and G. Tesauro, "Monte-carlo simulation balancing," in *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009, pp. 945–952.
[38] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. USA: Prentice Hall Press, 2009.
[39] D. Rolnick, A. Ahuja, J. Schwarz, T. P. Lillicrap, and G. Wayne, "Experience replay for continual learning," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019, pp. 350–360.
[40] S. Tuli, S. Ilager, K. Ramamohanarao, and R. Buyya, "Dynamic Scheduling for Stochastic Edge-Cloud Computing Environments using A3C learning and Residual Recurrent Neural Networks," *IEEE Transactions on Mobile Computing*, 2020.
[41] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of sumo-simulation of urban mobility," *International journal on advances in systems and measurements*, vol. 5, no. 3&4, 2012.
[42] A. Ahmed and G. Pierre, "Docker container deployment in fog computing infrastructures," in *2018 IEEE International Conference on Edge Computing (EDGE)*. IEEE, 2018, pp. 1–8.
[43] T. Coleman, H. Casanova, L. Pottier, M. Kaushik, E. Deelman, and R. F. da Silva, "Wfcommons: A framework for enabling scientific workflow research and development," *arXiv preprint arXiv:2105.14352*, 2021.
[44] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of scientific workflows," in *2008 third workshop on workflows in support of large-scale science*. IEEE, 2008, pp. 1–10.
[45] W. Rafique, L. Qi, I. Yaqoob, M. Imran, R. U. Rasool, and W. Dou, "Complementing IoT services through software defined networking and edge computing: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1761–1804, 2020.
[46] M. O'Grady, D. Langton, and G. O'Hare, "Edge computing: A tractable model for smart agriculture?" *Artificial Intelligence in Agriculture*, vol. 3, pp. 42–51, 2019.
[47] A. Semenov, "Elastic computing self-organizing for artificial intelligence space exploration," in *Journal of Physics: Conference Series*, vol. 1925, no. 1. IOP Publishing, 2021, p. 012071.
[48] B. Lin, F. Zhu, J. Zhang, J. Chen, X. Chen, N. N. Xiong, and J. L. Mauri, "A time-driven data placement strategy for a scientific workflow combining edge computing and cloud computing," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 7, pp. 4254–4265, 2019.
[49] Y. Shao, C. Li, Z. Fu, L. Jia, and Y. Luo, "Cost-effective replication management and scheduling in edge computing," *Journal of Network and Computer Applications*, vol. 129, pp. 46–61, 2019.
[50] Y. Xie, Y. Zhu, Y. Wang, Y. Cheng, R. Xu, A. S. Sani, D. Yuan, and Y. Yang, "A novel directional and non-local-convergent particle swarm optimization based workflow scheduling in cloud–edge environment," *Future Generation Computer Systems*, vol. 97, pp. 361–378, 2019.
[51] S. Shen, V. van Beek, and A. Iosup, "Statistical characterization of business-critical workloads hosted in cloud datacenters," in *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2015, pp. 465–474.
[52] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.
[53] A. Yates, K. Beal, S. Keenan, W. McLaren, M. Pignatelli, G. R. Ritchie, M. Ruffier, K. Taylor, A. Vullo, and P. Flicek, "The ensembl rest api: Ensembl data for any language," *Bioinformatics*, vol. 31, no. 1, pp. 143–145, 2015.
[54] M. Grinberg, *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2018.
[55] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations*, 2018.
[56] S. Yi, C. Li, and Q. Li, "A survey of fog computing: concepts, applications and issues," in *Proceedings of the 2015 workshop on mobile big data*, 2015, pp. 37–42.



Shreshth Tuli is a President's Ph.D. Scholar at the Department of Computing, Imperial College London, UK. Prior to this he was an undergraduate student at the Department of Computer Science and Engineering at Indian Institute of Technology - Delhi, India. He has worked as a visiting research fellow at the CLOUDS Laboratory, School of Computing and Information Systems, the University of Melbourne, Australia. He is a national level Kishore Vaigyanik Protsahan Yojana (KVPY) scholarship holder from the Government of India for excellence in science and innovation. His research interests include Fog Computing and Deep Learning. For further information, visit `https://shreshthtuli.github.io/`.



Giuliano Casale joined the Department of Computing at Imperial College London in 2010, where he is currently a Reader. Previously, he worked as a research scientist and consultant in the capacity planning industry. He teaches and does research in performance engineering and cloud computing, topics on which he has published more than 100 refereed papers. He has served on the technical program committee of over 80 conferences and workshops and as co-chair for several conferences in the area of performance and reliability engineering, such as ACM SIGMETRICS/Performance and IEEE/IFIP DSN. His research work has received multiple awards, recently the best paper award at ACM SIGMETRICS. He serves on the editorial boards of IEEE TNSM and ACM TOMPECS and as current chair of ACM SIGMETRICS.



Nicholas R. Jennings is the Vice-Chancellor and President of Loughborough University. He is an internationally-recognised authority in the areas of AI, autonomous systems, cyber-security and agent-based computing. He is a member of the UK government's AI Council, the governing body of the Engineering and Physical Sciences Research Council, and chair of the Royal Academy of Engineering's Policy Committee. Before Loughborough, he was the Vice-Provost for Research and Enterprise and Professor of Artificial Intelligence at Imperial College London, the UK's first Regius Professor of Computer Science (a post bestowed by the monarch to recognise exceptionally high quality research) and the UK Government's first Chief Scientific Advisor for National Security.