Cost-Aware Dynamic Cloud Workflow Scheduling using Self-Attention and Evolutionary Reinforcement Learning

Ya Shen✉, Gang Chen, Hui Ma, and Mengjie Zhang

Centre for Data Science and Artificial Intelligence & School of Engineering and Computer Science, Victoria University of Wellington, New Zealand
{ya.shen, aaron.chen, hui.ma, mengjie.zhang}@ecs.vuw.ac.nz

Abstract. As a key cloud management problem, Cost-aware Dynamic Multi-Workflow Scheduling (CDMWS) aims to assign virtual machine (VM) instances to execute tasks in workflows so as to minimize the total costs, including both the penalties for violating Service Level Agreement (SLA) and the VM rental fees. Powered by deep neural networks, Reinforcement Learning (RL) methods can construct effective scheduling policies for solving CDMWS problems. Traditional policy networks in RL often use basic feedforward architectures to separately determine the suitability of assigning any VM instances, without considering all VMs simultaneously to learn their global information. This paper proposes a novel self-attention policy network for cloud workflow scheduling (SPN-CWS) that captures global information from all VMs. We also develop an Evolution Strategy-based RL (ERL) system to train SPN-CWS reliably and effectively. The trained SPN-CWS can effectively process all candidate VM instances simultaneously to identify the most suitable VM instance to execute every workflow task. Comprehensive experiments show that our method can noticeably outperform several state-of-the-art algorithms on multiple benchmark CDMWS problems.

Keywords: Cloud Computing, Cloud Workflow Management, Dynamic Multi-Workflow Scheduling, Self-Attention, Reinforcement Learning

1 Introduction

Numerous computationally intensive and resource-demanding applications (e.g., weather forecasting, tsunami detection) will be submitted daily as workflows to a *cloud broker* for execution [5,7,14]. Each workflow comprises a collection of interdependent tasks that must be efficiently processed by using multiple leased virtual machine (VM) instances provided by major cloud providers, such as Amazon EC2 [2,9]. As a primary challenge, the cloud broker must quickly determine the type and number of VMs required to execute these tasks at the lowest possible cost. The broker then leases VM instances to process these workflows. This challenge is widely known as the *workflow scheduling* (WS) problem [9]. Fig. 1

illustrates how the broker schedules workflows on behalf of its users. Specifically, users submit their workflows to the broker. Each workflow is associated with user-defined *Service Level Agreement* (SLA) [19,24], requesting the broker to meet the execution deadline to avoid any *SLA violation penalties*. Consequently, the broker must dynamically maintain a desirable trade-off between VM rental fee and *SLA violation penalty*, making the WS problem extremely difficult to solve.

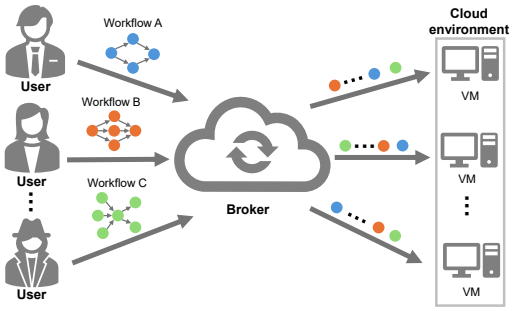


Fig. 1: The diagram of the broker performing the workflow scheduling.

This study addresses the *Cost-aware Dynamic Multi-Workflow Scheduling* (CDMWS) problem, where the broker handles multiple dynamically arriving workflows and makes real-time scheduling decisions. Due to the dynamic and combinatorial nature of the CDMWS problem, it is a well-known NP-hard problem [9]. To tackle this problem, the broker relies on a scheduling policy to make real-time scheduling decisions and instantly respond to newly arriving workflows. However, the suitability of using any scheduling policy may vary significantly across different problem scenarios [3,21]. The broker may fail to achieve its performance commitment and cost objectives upon using poorly designed policies [22]. Designing scheduling policies is hence a major research issue for dynamic multi-workflow scheduling.

Existing studies on scheduling policies can be classified into three categories: *manual approach*, *genetic programming hyper-heuristic* (GPHH), and *reinforcement learning* (RL) (see Section 2). However, most of those approaches use priority rules to rank individual VMs, without considering the status of other VMs. Further, most previously studied policy networks [3,9,10,26] adopt basic feedforward architectures with fixed inputs and outputs. They prioritize each candidate VM for task execution based solely on its own features, lacking the ability to consider features across multiple VMs simultaneously, which is essential to accurately capture workflow status and VM relationships and crucial for effective WS.

To tackle this research issue, we have the goal to propose a novel *Self-attention Policy Network for Cloud Workflow Scheduling* (SPN-CWS) to solve the CDMWS problem. In particular, for a given task to be scheduled, information regarding all candidate VMs are fed together into SPN-CWS, which then processes the global information using the self-attention mechanism. Sub-

sequently, SPN-CWS outputs the priority value with respect to each VM based on the global information and task-specific features. The VM with the highest priority is then selected to execute the task. Additionally, we develop an Evolution strategies-based RL [15] (ERL) system to train SPN-CWS. Compared to gradient-based RL algorithms, ERL is more robust to varied hyperparameter settings and can effectively cope with delayed or sparse rewards [10]. Furthermore, ERL can expedite the training process through parallelization. The contributions of our study are summarized below:

- We design a new SPN-CWS policy network that can effectively utilize global information across all candidate VMs to make informed task scheduling decisions. We adopt the first time in literature the self-attention mechanism in SPN-CWS to scalably handle global relationships among all candidate VMs.
- We develop an ERL system based on our cloud simulator to reliably train SPN-CWS. The trained SPN-CWS is subsequently employed by the broker to execute workflows that arrive dynamically over time.
- We conduct comprehensive experiments to thoroughly examine the performance of SPN-CWS and the accompanying ERL system. Experimental results show that the SPN-CWS trained by ERL can notably outperform multiple state-of-the-art approaches for dynamic WS.

The remaining of this paper is organized as follows: Section 2 and Section 3 give the related works and problem definitions of the CDMWS. Sections 4 and 5 describe the proposed SPN-CWS and its training method. Section 6 analysis the experiment results and the conclusions of this paper are given in Section 7.

2 Related Work

Research on dynamic WS is gaining increasing attention, driven by their substantial practical importance in diverse applications [9,21,23]. Dynamic WS presents a significant challenge, as it requires making real-time scheduling decisions tailored to the current cloud environment where no prior knowledge exists regarding workflows arriving in the future. While previous studies of dynamic WS [4,13] primarily focused on minimizing VM rental costs, a critical aspect often neglected was the trade-off between VM rental expenses and potential SLA violation penalties [7,20]. In fact, it may prove economically prudent to incur SLA penalties since meeting SLA deadlines often demand for leasing fast but expensive VMs [9,21,23]. Drawing inspiration from several recent studies [8,9,23], this paper embarks on an investigation into the CDMWS problem, aiming to learn scheduling policy to jointly optimize the VM rental fees and the SLA violation penalties.

There are three main categories of algorithms for learning scheduling policies for dynamic WS in cloud: manual approach, GPHH and RL. Manual approach [2,7,20] relies on domain experts to design scheduling policies by leveraging their problem-specific knowledge and empirical experiences. However, the designed policies are typically applied to simplified and static problems [9,16].

GPHH can automatically design WS policies that are highly adaptive to dynamic WS problems. For example, [23] introduced a novel dual-tree policy

representation for GPHH. The performance of GPHH is further enhanced with an adaptive mutation operator in [22]. A multi-tree GPHH method is also proposed in [21] to solve dynamic WS problems in fog computing environments. However, policies learned by GPHH focus solely on local information, considering each VM's features individually. Without explicitly processing the global information among all VMs (i.e., the operating status of all VMs managed by the broker and the relationships among these VMs), the learned policy may fail to identify suitable VMs for executing some workflow tasks, resulting in increased total cost.

In addition to GPHH, recent studies show promise of using RL methods to design policies for dynamic WS [5,10,12]. For instance, [10] developed an RL system to optimize the utilization of green energy while executing workflow tasks. An approach based on deep Q-networks was introduced by [18] with the objective of optimizing both workflow makespan and user costs. In [3], a RL-based collaborative scheduling method is proposed to achieve heterogeneous WS in cloud, aiming to improve overall service quality. However, most of the existing policy networks trained by RL adopt simple feedforward architectures with fixed inputs and outputs. Similar to GPHH, they were designed to process features related to each VM separately, lacking the capability of processing global information across all candidate VMs.

To address this issue, we design the first time in literature a new policy network (SPN-CWS) to simultaneously process global information among an arbitrary number of candidate VMs. Meanwhile, to ensure that the training of SPN-CWS is robust to hyperparameter settings and reward functions, we develop an ERL system to train SPN-CWS using simulated WS problems.

3 Problem Definition

In this section, we formally describe the Cost-aware Dynamic Multi-Workflow Scheduling problem (CDMWS) as illustrated in Fig. 2. This problem is centered around a *broker* that is responsible for dynamically scheduling workflow execution using leased VMs in cloud to minimize the total cost, including both the VM Rental fees and the SLA violation penalties.

Workflow model: In CDMWS, we assume that a sequence of workflows are dynamically submitted by users during T: $W(T) = \{w_i | i = 1, 2, ...\}$. Each workflow w_i contains a set of tasks connected by a *Directed Acyclic Graph* (DAG). A workflow w can be specifically formulated as follows:

$$w = [DAG(w), AT(w), DL(w), \beta(w) | w \in W(T)] \tag{1}$$

where $\beta(w)$ is the user specified SLA penalty rate [25] (see Eq. (9)). $AT(w)$ is the arrival time of w, and $DL(w)$ is the user specified SLA deadline (see Eq. (10)). $DAG(w)$ provides a directed acyclic graph of tasks, representing all the tasks to be executed as part of w as nodes and showing how these tasks are connected together through the directed edges in $DAG(w)$. $DAG(w)$ can be formulated as below:

$$DAG(w) = [Task(w), Edge(w)] \tag{2}$$

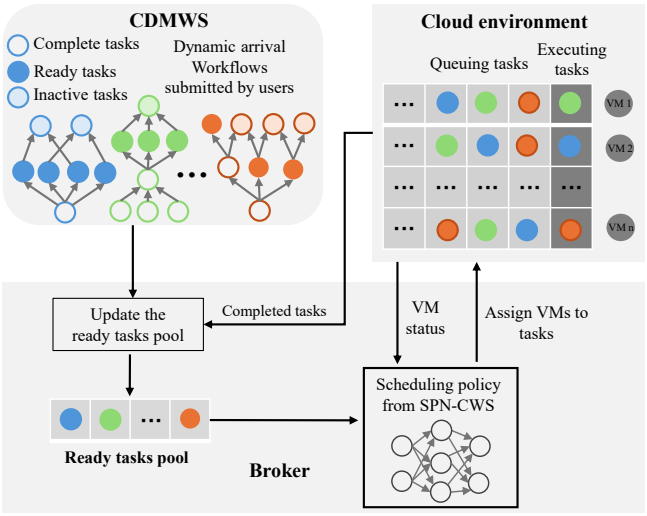


Fig. 2: The diagram of the scheduling of CDMWS.

where $Task(w)$ gives the set of tasks in workflow w. Each task $t \in Task(w)$ has its size determined as $Size(t)$ that quantifies the total amount of computation required for executing it. Each direct edge $(t, t') \in Edge(w)$ connects one task $t \in Task(w)$ to another task $t' \in Task(w)$, indicating that t is a *predecessor* task of t' and t' is a *successor* task of t. Specifically, any task $t \in Task(w)$ will become *ready* for execution only when all of its predecessor tasks are completed or no predecessor tasks exist.

Cloud environment: the cloud environment contains a set of VMs with varied VM types. In this study, the types of VM are limited, but the quantity available for each type is unlimited, meaning that the broker can lease an arbitrary number of VM instances of any type on demand. A VM instance denoted as v in the cloud can be characterized as follows:

$$v = [Type(v), Capacity(v), Price(v)] \tag{3}$$

where $Type(v)$ indicates the VM type; $Capacity(v)$ gives the computation capability per time unit of v [7] ; $Price(v)$ is the rental fee per hour incurred as a result of using/leasing v. According to several existing works [7,9,23], the rental time for a fraction of an hour is charged as one hour.

Execution time: Let the VM instance chosen by policy π to execute task $t \in Task(w)$ of workflow w be denoted as $v_{t,w,\pi}$. The required execution time of t, i.e., $EXT(t, w, \pi)$, is determined as:

$$EXT(t, w, \pi) = \frac{Size(t)}{Capacity(v_{t,w,\pi})} \tag{4}$$

We further denote $AVM(v|t, w, \pi)$ as the set of all candidate VM instances that can be used to execute task $t \in Task(w)$ of workflow w, including all the

currently rented VMs as well as VMs that can be newly leased from the cloud. Clearly $v_{t,w,\pi} \in AVM(v|t,w,\pi)$.

In this study, workflows arrive at the broker dynamically across time, demanding π to make its scheduling decisions in real time. In order to reduce the VM rental fees and the SLA penalty, every ready task must be scheduled for execution immediately whenever it becomes ready [9]. The following formulation captures the completion time of a *ready* task $t \in Task(w)$ of workflow w:

$$CT(t,w,\pi) = ST(t,w,\pi) + EXT(t,w,\pi) \tag{5}$$

where $CT(t,w,\pi)$ and $ST(t,w,\pi)$ are the completion time and start time of $t \in Task(w)$ upon using π, respectively. Hence, the completion time of workflow w under policy π is:

$$WCT(w,\pi) = \max_{t \in Task(w)} CT(t,w,\pi) \tag{6}$$

VM rental fees: CDMWS focuses on processing multiple dynamically arriving workflows, each consisting of a collection of tasks to be executed on multiple VMs. When all workflows are completed, the total rental fees of all leased VMs will be calculated. We use T to represent the full operation period associated with a CDMWS problem. This time period starts from time t_s and ends at time t_e. The duration of T depends on the scheduling policy π used to execute all workflow tasks. During T, each VM instance v is used for a certain time period, as defined below:

$$RP(v,\pi,T) = [t_s(v,\pi,T), t_e(v,\pi,T)] \tag{7}$$

where $t_s(v,\pi,T) \geq t_s$ is the start time for using v, and $t_e(v,k,\pi,T) \leq t_e$ is the time at which v is no longer used. Thus, the total rental fees of all VMs under the scheduling policy π during T can be calculated as follows:

$$VMFee(\pi,T) = \sum_{v \in Set(\pi,T)} \left(Price(v) \times \left\lceil \frac{t_e(v,\pi,T) - t_s(v,\pi,T)}{3600} \right\rceil \right) \tag{8}$$

where $Set(\pi,T)$ refers to the set of all VM instances leased to execute workflow tasks during T. The ceiling function in Eq. (8) converts the time period that a VM instance v is used to its corresponding renting time in integer hours.

SLA penalty: Apart from VM rental fees, the SLA penalty presents a major source of cost to be minimized in CDMWS. According to [9,19,23], the SLA penalty of a workflow w can be calculated by the following:

$$SLA_{Penalty}(w,\pi) = \beta(w) \times \max\{0, [WCT(w,\pi) - DL(w)]\} \tag{9}$$

In Eq. (9), the SLA deadline specified by users, i.e., $DL(w)$, is determined as:

$$DL(w) = AT(w) + \gamma \times MinMakespan(w) \tag{10}$$

where $MinMakespan(w)$ is the theoretical shortest time duration required for processing w, achievable by using the fastest VM to process all tasks of workflow

w without any delay. γ is the relaxation coefficient. Increasing γ results in more relaxed deadline for w.

In CDMWS, we aim to find an effective (optimal) scheduling policy π to minimize the total cost, as formulated below:

$$\arg\min_{\pi} TotalCost(\pi) = \arg\min_{\pi} \left\{ VMFee(\pi, T) + \sum_{w \in W(T)} SLA_{Penalty}(w, \pi) \right\} \tag{11}$$

4 Self-Attention Policy Network for Cloud Workflow Scheduling (SPN-CWS)

4.1 State information

The scheduling policy plays a critical role for CDMWS. Whenever a task (i.e., rt) of any workflow (i.e., w_{rt}) becomes ready for execution at time t, π is utilized to process the state information obtained from the CDMWS problem as its input, and then produces a VM selected to execute rt as its action output. Therefore, we first introduce the state representation in association with the CDMWS problem being solved, which will serve as the input of SPN-CWS.

In line with [9,23], we design the state representation by considering both the *task-info* of the ready task rt and the *VM-info* of all the VMs in $AVM(v|rt, w_{rt}, \pi)$ that can be utilized for executing rt. We specifically list all the features employed to build the state input with respect to *task-info* and *VM-info* in Table 1. Whenever there is a task rt ready to be processed, the *task-info* of rt and the *VM-info* of all VM instances in $AVM(v|rt, w_{rt}, \pi)$ jointly form the representation of the current state as the input of SPN-CWS.

Table 1: Task information (*task-info*) and VM information (*VM-info*).

task-info	***Number of Successors***	It gives the number of tasks that depend on rt in w_{rt}, i.e., $\|Suc(w_{rt}, rt)\|$, where $\|\cdot\|$ stands for the set cardinality.
	Completion ratio	It calculates the workflow completion ratio of w_{rt} through $Com_{tasks}/Total_{tasks}$, where Com_{tasks} is the number of completed tasks in w_{rt} and $Total_{tasks}$ is the total number of tasks in w_{rt}.
	Arrival rate	It estimates the workflow arrival rate for future workflows according to the current execution situation of w_{rt} [9].
VM-info	***Task deadline***	It indicates whether using the VM can meet the task deadline of rt. The task deadline is calculated using the method in [20]. Depending on the task size, the workflow deadline is assigned to each task. Large size task will be assigned a larger task deadline.
	Incurred cost	It calculates the corresponding VM rental fee and ***Task deadline*** violation penalty to be incurred upon using the VM to execute rt.
	Remaining time	It counts the remaining VM rental time after executing rt on this VM.
	Fittest VM	It indicates whether the VM being considered for executing rt has the lowest ***Incurred cost*** among all candidate VMs while meeting the ***Task deadline***.

4.2 Architecture Design of SPN-CWS

To process global information across all VMs in $AVM(v|rt, w_{rt}, \pi)$ for effective scheduling of the ready task rt, we design a novel SPN-CWS deep model for the scheduling policy π in CDMWS, as shown in Fig. 3. Specifically, we introduce the time-tested *Multi-Head Self-Attention* mechanism (MHSA) [17], as highlighted in the ***Global information learning*** component of Fig. 3, to enhance the capability for policy π to process global VM information effectively and scalably (see the detailed experiments in Section 6.2). SPN-CWS mainly consists of five key components as described below.

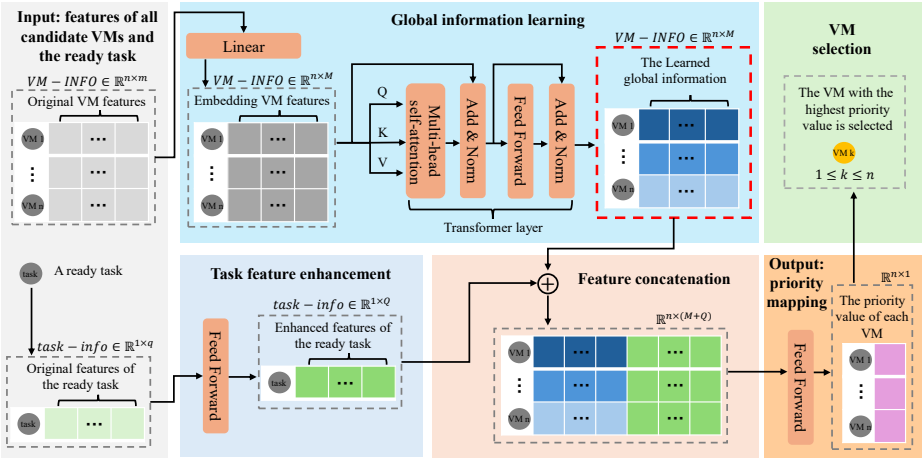


Fig. 3: The structure of SPN-CWS.

Global information learning: This component aims to learn global information among VMs. Firstly, we combine the *VM-info* of all VMs in $AVM(v|rt, w_{rt}, \pi)$ to construct the *VM-INFO* indicated in Fig. 3. Then, we employ a linear layer to map *VM-INFO* from the original $\mathbb{R}^{n\times m}$ space to a high-dimensional $\mathbb{R}^{n\times M}$ space ($M > m$ and $n = |AVM(v|rt, w_{rt}, \pi)|$, i.e., the total number of VM instances in $AVM(v|rt, w_{rt}, \pi)$). Then, the Transformer layer processes the projected VM features to learn the relationships among all VMs. In more details, first we use MHSA to compute a weighted sum of all VM features, where the weights are determined by the similarity between every pair of VMs. The weighted features are further processed by Add & Norm and Feed-Forward Networks (FFN) to learn more complex feature representations. MHSA allows features of each VM instance to attend to features of other VM instances, effectively capturing inter-VM relationships and allowing SPN-CWS to focus on the most relevant VM instances.

Task feature enhancement: This component processes and transforms task features. In this component, a FFN is used to map *task-info* from the $\mathbb{R}^{1\times q}$ space to a high-dimensional $\mathbb{R}^{1\times Q}$ space ($Q > q$) to build high-level feature representations of the ready task rt being scheduled. FFN refines the task features,

ensuring that the task representation captures important information that is essential for the subsequent concatenation with VM features.

Feature concatenation: This component concatenates task features of the ready task with the features from each VM instance. In order to perform priority mapping on every VM instance, in this component, the processed *task-info* is concatenated separately with the *VM-INFO* associated with each candidate VM instance.

Priority mapping: This component maps concatenated task and VM features to priority values that dictate the importance of using any specific VM instance to execute the ready task. For example, using the *VM-INFO* features with respect to v_1 and the features in the *task-info*, the priority value of VM instance v_1 is determined through a FFN included in this component, as shown in Fig. 3.

VM Selection: This component selects the VM instance with the highest priority value to execute the ready task rt.

5 Training SPN-CWS via ERL

The performance of gradient-based RL can be highly sensitive to its hyperparameter settings as well as the design of the reward function. ERL can alleviate these issues and achieve robust learning performance [1,11,15]. Therefore, in this study, we develop an ERL system based on our cloud simulator to train SPN-CWS designed above for CDMWS, which is denoted as π_{sc}. The pseudo-code of the training algorithm is presented in Algorithm 1. Specifically, the training process of π_{sc} using ERL is described as follows:

1. Let $\hat{\theta}$ denote the current policy parameter of π_{sc}. During each generation (i.e., each iteration of the ERL), ERL samples a population of N individuals, where each individual $(\theta_i|i = 1, 2, ...N)$ is sampled from an isotropic multivariate Gaussian distribution with $\hat{\theta}$ as its mean vector and $\sigma^2 I$ as its co-variance matrix. Hence, $\theta_i \sim \mathcal{N}(\hat{\theta}, \sigma^2 I)$, which is equivalent to $\theta_i = \hat{\theta} + \sigma\epsilon_i$ with $\epsilon_i \sim \mathcal{N}(0, I)$. θ_i indicates the parameters of π_i.
2. The fitness value of θ_i (i.e., $F(\theta_i)$) equals the total cost (as described in Section 3) achieved by using π_i to solve any CDMWS problem used for training. Since ERL aims to learn the policy parameters that minimize the total cost, we define the fitness function as follows:

$$F(\theta_i) = F(\hat{\theta} + \sigma\epsilon_i) = -TotalCost(\pi_i) \tag{12}$$

3. With Eq. (12) as the objective function, ERL updates $\hat{\theta}$ to maximize the expected value of the objective function (i.e., $\mathbb{E}_{\theta_i \sim \mathcal{N}(\hat{\theta}, \sigma^2 I)} F(\theta_i)$), thereby minimizing the total costs. Specifically, ERL updates $\hat{\theta}$ using policy gradients estimated below [15]:

$$\begin{aligned} \nabla_{\hat{\theta}} \mathbb{E}_{\theta_i \sim \mathcal{N}(\hat{\theta}, \sigma^2 I)} F(\theta_i) &= \nabla_{\hat{\theta}} \mathbb{E}_{\epsilon_i \sim \mathcal{N}(0, I)} F(\hat{\theta} + \sigma\epsilon_i) \\ &= \frac{1}{\sigma} \mathbb{E}_{\epsilon_i \sim \mathcal{N}(0, I)} \{F(\hat{\theta} + \sigma\epsilon_i)\epsilon_i\} \end{aligned} \tag{13}$$

Algorithm 1: ERL for training SPN-CWS (π_{sc})

Input : Population size: N, max number of generation: Gen, initial parameters of π_{sc}: $\hat{\theta}$, initial learning rate: α, and the Gaussian standard noise deviation: σ

Output: Scheduling policy: π_{sc} (the trained SPN-CWS)

While the current number of generation <= Gen: **do**
  Randomly generate a CDMWS training problem from our simulator: Pro
  For each individual (i=1,2,...) **in** N: **do**
    Sample a $\epsilon_i \sim \mathcal{N}(0, I)$
    The parameters of π_i represented by individual i: $\theta_i = \hat{\theta} + \sigma\epsilon_i$
    Evaluate the fitness value of $F(\theta_i)$ using Eq. (12) based on Pro
  End for
  Estimate the policy gradient $\nabla_{\hat{\theta}}\mathbb{E}_{\theta_i \sim \mathcal{N}(\hat{\theta}, \sigma^2 I)}F(\theta_i)$ using Eq. (13)
  Update parameters of π_{sc}: $\hat{\theta} \leftarrow \hat{\theta} + \alpha\nabla_{\hat{\theta}}\mathbb{E}_{\theta_i \sim \mathcal{N}(\hat{\theta}, \sigma^2 I)}F(\theta_i)$
End while

6 Experiments

6.1 Simulation Environment Configuration

In this section, we present the CDMWS simulation environment for training SPN-CWS using ERL as well as all the competing algorithms. The simulation environment is specifically described as follows.

VM types and workflow patterns: We configure the VMs used in CDMWS based on Amazon EC2[1]. In line with existing studies [9,22,23], we use six types of VMs with their respective configurations summarized in Table 2. Following [9,22], each type of VM can be leased on demand with unlimited number of instances. As summarized in Table 3, the workflows experimented consist of four popular patterns (i.e., CyberShake, Montage, Inspiral and SIPHT) that are commonly used in recent studies [9,21,22]. Workflows of the same pattern become more complex and difficult to solve as the number of tasks in the workflows increases. Based on the workflow size (i.e., number of workflow tasks), workflows are grouped into three categories: *Small*, *Medium*, and *Large*.

Table 2: All VM types used in this study.

VM name	vCPU	Memory (GB)	Cost ($ per hour)
m5.large	2	8	0.096
m5.xlarge	4	16	0.192
m5.2xlarge	8	32	0.384
m5.4xlarge	16	64	0.768
m5.8xlarge	32	128	1.536
m5.12xlarge	48	192	2.304

CDMWS problem instance: Following [9,22], each CDMWS problem instance consists of 30 randomly sampled workflows corresponding to all the four

[1] https://aws.amazon.com/ec2/pricing/on-demand/

workflow patterns in Table 3. All workflows in this study are dynamically generated according to a Poisson distribution with $\lambda = 0.01$ to simulate workflows submitted by users across time. We set the SLA deadline coefficient[2] in Eq. (10) as: $\gamma \in \{1.00, 1.25, 1.50, 1.75, 2.00, 2.25\}$. SLA deadlines become more relaxed as γ increases, enabling the broker to rent cheaper VMs to execute newly arrived workflows. $\beta(w)$ in Eqs. (1) and (9) is set to \$0.24/*hour* according to [9,22].

Table 3: The three workflow sets used in this study.

Workflow set	**Name of pattern(number of task)**			
Small	CyberShake(30)	Montage(25)	Inspiral(30)	SIPHT(30)
Medium	CyberShake(50)	Montage(50)	Inspiral(50)	SIPHT(60)
Large	CyberShake(100)	Montage(100)	Inspiral(100)	SIPHT(100)

CDMWS problem scenarios: Small-scenario CDMWS problem instances are generated from *small* workflow set. Meanwhile, medium- and large-scenario CDMWS problem instances are generated from *Medium* and *Large* workflow sets, respectively. For each generation of Algorithm 1, we sample a small-scenario CDMWS problem instance for fitness evaluation. During testing, we use 30 small-scenario, 30 medium-scenario, and 30 large-scenario CDMWS instances to jointly evaluate the performance of the trained SPN-CWS. Notably, the medium- and large-scenario CDMWS problems instances are not used during training for computation efficiency reasons. They are only employed to assess the generalization capability of SPN-CWS during testing. In each test scenario, the scheduling policy's performance is calculated based on the average of 30 CDMWS problem instances.

Baseline algorithms: Four competing methods are included in our experiments. Among them, **ProLis** [20] and **GRP-HEFT** [7] are two popular heuristic methods designed manually by domain experts and are frequently studied in existing works [5,21,22]. **DSGP** [6] is a GPHH-based approach that learns a heuristic rule as scheduling policy through evolutionary search in the hyper-heuristic space. **ES-RL** [9] adopts an RL-based algorithm designed by OpenAI[3] to train a scheduling policy modeled as a feedforward neural network.

Parameter settings: The parameter settings of all competing algorithms strictly follow their original papers. For Algorithm 1, N, Gen, α, and σ are set to 40, 3000, 0.01, and 0.05, respectively, which are identical to the settings adopted in [9] for a fair comparison. According to Table 1, m and q of Fig. 3 are 4 and 3, respectively, while M and Q are set to 16 in this study. In SPN-CWS, the size of feedforward hidden layers in the *Global information learning*, *Task feature enhancement*, and *Priority mapping* components are set to 64, 32, and 32, respectively, with $ReLU$ as the activation function.

[2] ES-RL and SPN-CWS are trained with $\gamma = 5$ and tested with $\gamma \in \{1.00, 1.25, 1.50, 1.75, 2.00, 2.25\}$ to evaluate their performance under tight SLA deadline coefficients.

[3] https://github.com/openai

6.2 Main Results

The performance of all competing algorithms are summarized in Table 4, which clearly indicates that SPN-CWS significantly outperforms ProLis and GRP-HEFT across all scenarios, confirming the importance of designing scheduling policies automatically. Furthermore, compared to DSGP and ES-RL, thanks to its capability of processing global information among all candidate VMs, SPN-CWS achieved significantly better overall performance. Particularly, on majority of medium and large scenarios, SPN-CWS significantly outperforms DSGP and ES-RL, with scenario $\langle 2.00, M\rangle$ as the only exception. This shows that SPN-CWS, while being trained on small problems, can achieve reliable generalization performance on large problems.

Meanwhile, we have bolded the best performance results in Table 4, which have been thoroughly verified through the Wilcoxon ranked sum test. The corresponding p-values associated with the bolded results are consistently less than 0.05. For example, on the scenario $\langle 1.50, M\rangle$, SPN-CWS can manage to reduce the total costs by 5.29% and 56.9% respectively, compared to the total costs realized by DSGP and ES-RL.

Moreover, Table 4 reveals that, upon increasing γ, the total costs achieved by SPN-CWS consistently exhibit a downward trend across all test scenarios. This suggests that a more relaxed SLA deadline coefficient encourages SPN-CWS to utilize cheaper VM instances, thereby effectively lowering the total costs.

Table 4: Average (standard deviation) total cost of each algorithm over 30 independent runs.

Scenarios	ProLis [20]	GRP-HEFT [7]	DSGP [6]	ES-RL [9]	SPN-CWS (our)
$\langle 1.00, S\rangle$	773.69	1685.01(-)	**142.88**(16.35)(+)(+)	215.83(50.34)(+)(+)(-)	**145.53**(14.96)(+)(+)(≈)(+)
$\langle 1.00, M\rangle$	1829.25	2867.64(-)	244.14(66.39)(+)(+)	456.87(125.70)(+)(+)(-)	**238.68**(51.38)(+)(+)(+)(+)
$\langle 1.00, L\rangle$	3641.79	5873.20(-)	422.49(96.27)(+)(+)	1199.20(264.65)(+)(+)(-)	**348.19**(87.92)(+)(+)(+)(+)
$\langle 1.25, S\rangle$	923.15	1967.31(-)	**128.24**(11.12)(+)(+)	301.46(77.59)(+)(+)(-)	131.05(12.67)(+)(+)(-)(+)
$\langle 1.25, M\rangle$	2068.78	3578.04(-)	232.34(57.15)(+)(+)	568.56(173.47)(+)(+)(-)	**212.72**(35.50)(+)(+)(+)(+)
$\langle 1.25, L\rangle$	4213.29	6868.22(-)	438.23(80.27)(+)(+)	1278.11(287.33)(+)(+)(-)	**348.33**(83.90)(+)(+)(+)(+)
$\langle 1.50, S\rangle$	901.40	1963.39(-)	**126.12**(14.49)(+)(+)	249.40(54.74)(+)(+)(-)	**127.76**(12.18)(+)(+)(≈)(+)
$\langle 1.50, M\rangle$	2035.52	3567.97(-)	224.21(47.68)(+)(+)	492.13(164.01)(+)(+)(-)	**212.34**(34.98)(+)(+)(+)(+)
$\langle 1.50, L\rangle$	4066.84	6863.62(-)	405.98(94.18)(+)(+)	913.00(265.89)(+)(+)(-)	**343.76**(83.37) (+)(+)(+)(+)
$\langle 1.75, S\rangle$	804.33	1959.71(-)	**123.14**(13.49)(+)(+)	227.23(46.17)(+)(+)(-)	**122.58**(10.14)(+)(+)(≈)(+)
$\langle 1.75, M\rangle$	1977.10	3560.22(-)	210.46(37.68)(+)(+)	455.73(136.39)(+)(+)(-)	**209.54**(33.87)(+)(+)(+)(+)
$\langle 1.75, L\rangle$	3898.42	6854.40(-)	398.29(84.18)(+)(+)	818.91(259.21)(+)(+)(-)	**348.22**(80.40)(+)(+)(+)(+)
$\langle 2.00, S\rangle$	789.71	1957.25(-)	**120.74**(12.47)(+)(+)	206.94(39.40)(+)(+)(-)	**119.17**(9.73)(+)(+)(≈)(+)
$\langle 2.00, M\rangle$	1921.42	3554.07(-)	**204.36**(37.68)(+)(+)	409.84(135.83)(+)(+)(-)	**205.62**(32.80)(+)(+)(≈)(+)
$\langle 2.00, L\rangle$	4024.03	6847.49(-)	388.64(104.18)(+)(+)	748.58(246.18)(+)(+)(-)	**349.51**(95.20)(+)(+)(+)(+)
$\langle 2.25, S\rangle$	710.18	1954.71(-)	**115.28**(13.13)(+)(+)	184.78(31.46)(+)(+)(-)	**116.93**(10.61)(+)(+)(≈)(+)
$\langle 2.25, M\rangle$	1924.27	3551.31(-)	197.96(35.28)(+)(+)	362.83(144.59)(+)(+)(-)	**194.06**(27.23)(+)(+)(+)(+)
$\langle 2.25, L\rangle$	3957.08	6842.88(-)	373.35(94.76)(+)(+)	611.03(218.16)(+)(+)(-)	**343.97**(78.32)(+)(+)(+)(+)

* $\langle 1.00, S\rangle$ denotes that $\gamma = 1.00$ and algorithms are tested on the small-scenario CDMWS instance.
* ProLis and GRP-HEFT are deterministic heuristics, therefore have no standard deviation.
* (+), (-) or (≈) indicates that the result is significantly better, worse or equivalent to the corresponding algorithm based on Wilcoxon test with a significance level of 0.05.

Fig. 4 presents the VM fees and SLA penalties for each algorithm. We can see that GRP-HEFT tends to treat the SLA deadline as a hard constraint (the SLA penalty is 0), leading to excessively high VM rental fees. Similarly, ProLis, DSGP, and ES-RL also tend to produce excessively high VM rental fees. In contrast, SPN-CWS can better balance the trade-off between VM rental fees and

SLA penalties, thanks to its capability of using global information to select suitable VM instances. SPN-CWS incurs higher SLA penalties compared to DSGP and ES-RL, but it can significantly reduce the total costs by using cheap VMs, especially on medium and large CDMWS problem scenarios.

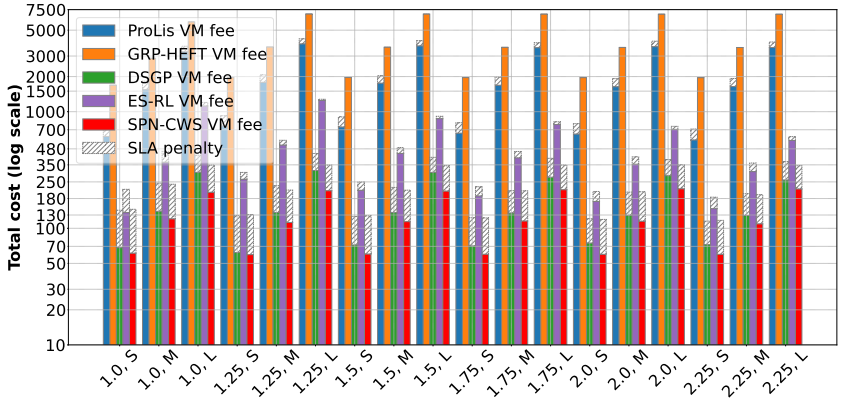
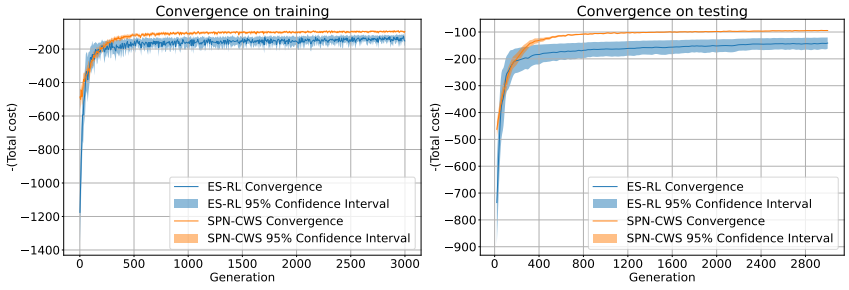


Fig. 4: The average of VM fees and SLA penalties of all algorithms.

6.3 Convergence Analysis

Since both ES-RL and SPN-CWS are RL-based algorithms, we compare the convergence behaviors of ES-RL and SPN-CWS during training and testing in Fig. 5. As demonstrated in this figure, SPN-CWS achieves lower total costs compared to ES-RL during training and testing. Notably, despite of using complex network architectures, SPN-CWS achieved competitive convergence speed as that of ES-RL. It also showed faster convergence speed than ES-RL on the testing problems. Furthermore, compared to ES-RL, SPN-CWS enjoys clearly smaller confidence intervals during both training and testing, suggesting that the trained SPN-CWS can perform more consistently and reliably.



(a) (b)

Fig. 5: The convergence on small-scenario CDMWS instance with $\gamma = 5$.

7 Conclusions

This paper introduced a Self-Attention Policy Network (SPN-CWS) for cloud workflow scheduling, capable of capturing global information across all VMs. We developed an ERL system with a cloud simulator to train SPN-CWS efficiently. The trained SPN-CWS selects the most suitable VM for each workflow task by processing all candidate VMs as input. Experimental results demonstrated that our approach significantly outperforms state-of-the-art algorithms in CDMWS and exhibits good convergence speed and stability. Future work will explore online reinforcement learning to enhance adaptability to dynamic workflow changes.

References

1. Ajani, O.S., Mallipeddi, R.: Adaptive evolution strategy with ensemble of mutations for reinforcement learning. Knowledge-Based Systems **245**, 108624 (2022)
2. Arabnejad, V., Bubendorfer, K., Ng, B.: Budget and deadline aware e-science workflow scheduling in clouds. IEEE Transactions on Parallel and Distributed systems **30**(1), 29–44 (2018)
3. Chen, G., Qi, J., Sun, Y., Hu, X., Dong, Z., Sun, Y.: A collaborative scheduling method for cloud computing heterogeneous workflows based on deep reinforcement learning. Future Generation Computer Systems **141**, 284–297 (2023)
4. Chen, H., Zhu, X., Liu, G., Pedrycz, W.: Uncertainty-aware online scheduling for real-time workflows in cloud service environment. IEEE Transactions on Services Computing **14**(4), 1167–1178 (2018)
5. Dong, T., Xue, F., Xiao, C., Zhang, J.: Workflow scheduling based on deep reinforcement learning in the cloud environment. Journal of Ambient Intelligence and Humanized Computing **12**(12), 10823–10835 (2021)
6. Escott, K.R., Ma, H., Chen, G.: Genetic programming based hyper heuristic approach for dynamic workflow scheduling in the cloud. In: Database and Expert Systems Applications: 31st International Conference, DEXA 2020, Bratislava, Slovakia, September 14–17, 2020, Proceedings, Part II 31. pp. 76–90. Springer (2020)
7. Faragardi, H.R., Sedghpour, M.R.S., Fazliahmadi, S., Fahringer, T., Rasouli, N.: Grp-heft: A budget-constrained resource provisioning scheme for workflow scheduling in iaas clouds. IEEE Transactions on Parallel and Distributed Systems **31**(6), 1239–1254 (2019)
8. Hoseiny, F., Azizi, S., Shojafar, M., Tafazolli, R.: Joint qos-aware and cost-efficient task scheduling for fog-cloud resources in a volunteer computing system. ACM Transactions on Internet Technology **21**(4), 1–21 (2021)
9. Huang, V., Wang, C., Ma, H., Chen, G., Christopher, K.: Cost-aware dynamic multi-workflow scheduling in cloud data center using evolutionary reinforcement learning. In: International Conference on Service-Oriented Computing. pp. 449–464. Springer (2022)
10. Jayanetti, A., Halgamuge, S., Buyya, R.: Multi-agent deep reinforcement learning framework for renewable energy-aware workflow scheduling on distributed cloud data centers. IEEE Transactions on Parallel and Distributed Systems (2024)
11. Khadka, S., Tumer, K.: Evolution-guided policy gradient in reinforcement learning. Advances in Neural Information Processing Systems **31** (2018)

12. Li, H., Huang, J., Wang, B., Fan, Y.: Weighted double deep q-network based reinforcement learning for bi-objective multi-workflow scheduling in the cloud. Cluster Computing **25**(2), 751–768 (2022)
13. Liu, J., Ren, J., Dai, W., Zhang, D., Zhou, P., Zhang, Y., Min, G., Najjari, N.: Online multi-workflow scheduling under uncertain task execution time in iaas clouds. IEEE Transactions on Cloud Computing **9**(3), 1180–1194 (2019)
14. Masdari, M., ValiKardan, S., Shahi, Z., Azar, S.I.: Towards workflow scheduling in cloud computing: a comprehensive analysis. Journal of Network and Computer Applications **66**, 64–82 (2016)
15. Salimans, T., Ho, J., Chen, X., Sidor, S., Sutskever, I.: Evolution strategies as a scalable alternative to reinforcement learning. arxiv 2017. arXiv preprint arXiv:1703.03864 (2017)
16. Silver, E.A.: An overview of heuristic solution methods. Journal of the operational research society **55**(9), 936–956 (2004)
17. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)
18. Wang, Y., Liu, H., Zheng, W., Xia, Y., Li, Y., Chen, P., Guo, K., Xie, H.: Multi-objective workflow scheduling with deep-q-network-based multi-agent reinforcement learning. IEEE access **7**, 39974–39982 (2019)
19. Wu, L., Garg, S.K., Versteeg, S., Buyya, R.: Sla-based resource provisioning for hosted software-as-a-service applications in cloud computing environments. IEEE Transactions on services computing **7**(3), 465–485 (2013)
20. Wu, Q., Ishikawa, F., Zhu, Q., Xia, Y., Wen, J.: Deadline-constrained cost optimization approaches for workflow scheduling in clouds. IEEE Transactions on Parallel and Distributed Systems **28**(12), 3401–3412 (2017)
21. Xu, M., Mei, Y., Zhu, S., Zhang, B., Xiang, T., Zhang, F., Zhang, M.: Genetic programming for dynamic workflow scheduling in fog computing. IEEE Transactions on Services Computing **16**(4), 2657–2671 (2023)
22. Yang, Y., Chen, G., Ma, H., Hartmann, S., Zhang, M.: Dual-tree genetic programming with adaptive mutation for dynamic workflow scheduling in cloud computing. IEEE Transactions on Evolutionary Computation (2024)
23. Yang, Y., Chen, G., Ma, H., Zhang, M.: Dual-tree genetic programming for deadline-constrained dynamic workflow scheduling in cloud. In: International Conference on Service-Oriented Computing. pp. 433–448. Springer (2022)
24. Yang, Y., Chen, G., Ma, H., Zhang, M., Huang, V.: Budget and sla aware dynamic workflow scheduling in cloud computing with heterogeneous resources. In: 2021 IEEE Congress on Evolutionary Computation (CEC). pp. 2141–2148. IEEE (2021)
25. Youn, C.H., Chen, M., Dazzi, P.: Cloud broker and cloudlet for workflow scheduling. Springer (2017)
26. Zhou, B., Cheng, L.: Deep reinforcement learning-based scheduling for same day delivery with a dynamic number of drones. In: International Conference on Service-Oriented Computing. pp. 34–41. Springer (2023)