Proceedings of the 2025 Winter Simulation Conference
E. Azar, A. Djanatliev, A. Harper, C. Kogler, V. Ramamohan, A. Anagnostou, and S. J. E. Taylor, eds.

GNN-HEATMAP AUGMENTED MONTE CARLO TREE SEARCH FOR CLOUD WORKFLOW SCHEDULING

Dingyu Zhou[1], Jiaqi Huang[1], Yirui Zhang[1], and Wai Kin (Victor) Chan [1]

[1]Shenzhen International Graduate School, Tsinghua University, Shenzhen, CHINA

ABSTRACT

This paper addresses the NP-hard cloud workflow scheduling problem by proposing a novel method that integrates Graph Neural Networks with Monte Carlo Tree Search (MCTS). Cloud workflows, represented as Directed Acyclic Graphs, present significant scheduling challenges due to complex task dependencies and heterogeneous resource requirements. Our method leverages Anisotropic Graph Neural Networks to extract structural features from workflow and creates a heatmap that guides the MCTS process during both the selection and simulation phases. Extensive experiments on workflows ranging from 30 to 110 tasks demonstrate that our method outperforms rule-based algorithms, classic MCTS, and other learning-based approaches; more notably, it achieves near-optimal solutions with only a 2.56% gap from exact solutions and demonstrates exceptional scalability to completely unseen workflow sizes. This synergistic integration of neural network patterns with Monte Carlo simulation-based search not only advances cloud workflow scheduling but also offers valuable insights for simulation-based optimization across diverse domains.

1 INTRODUCTION

Cloud infrastructure has revolutionized computational ecosystems, enabling complex distributed applications for data analytics, scientific simulations, and machine learning across various domains (Jalali Khalil Abadi et al. 2024). These computational tasks are often structured into workflows, which are naturally represented as Directed Acyclic Graphs (DAG), where vertices denote computational tasks and directed edges capture their dependencies. As cloud computing proliferates, workflows exhibit increasing heterogeneity in structure and resource demands. The core challenge in cloud workflow scheduling—assigning interdependent tasks to resource containers while minimizing the total completion time (*makespan*)—is an NP-hard problem, and traditional methods struggle as complexity grows (Zhou et al. 2024).

Existing workflow scheduling approaches may be categorized into three paradigms: rule-based, heuristic, and learning-based methods. However, each of these paradigms exhibits significant limitations: rule-based algorithms lack adaptability, heuristic methods struggle with transferability, and learning-based approaches face scale challenges across diverse workflow structures. These limitations call for more robust approaches to cloud workflow scheduling. Monte Carlo Tree Search offers a powerful simulation-based method to explore decision spaces (Kemmerling et al. 2024), but the inefficient exploration strategies might affect algorithm performance. To solve this problem, we propose a novel approach that integrates Graph Neural Networks (GNN) with MCTS, combining GNN's pattern recognition capabilities with MCTS's robust decision-making framework.

Our contributions are as follows: (1) We develop a comprehensive workflow generation framework that captures the structural complexity and resource heterogeneity of modern cloud computing workloads. (2) We demonstrate how Anisotropic Graph Neural Networks can effectively extract critical structural features from workflow DAGs. (3) We propose a novel GNN-Heatmap augmented Monte Carlo Tree Search algorithm, and validate its effectiveness and scalability for the cloud workflow scheduling problem.

2 RELATED WORK

Cloud workflow scheduling has been widely studied due to its significance in modern computing infrastructures. Existing approaches can be broadly categorized into three paradigms: rule-based, heuristic, and learning-based methods.

Rule-based algorithms rely on predefined rules to schedule tasks based on specific criteria. HEFT (Topcuoglu et al. 2002) considers task execution times and communication costs to prioritize tasks, but is limited to heterogeneous resource environments. Graphene (Grandl et al. 2016) focuses on packing tasks into resource slots, but may delay less demanding but dependency-critical tasks. While rule-based algorithms are simple and easy to implement, they require extensive parameter tuning and manual adjustments, often lack adaptability to dynamically changing cloud environments.

Recent years have seen a surge in learning-based approaches that utilize historical data to learn patterns and make informed scheduling decisions. Topoformer (Gagrani et al. 2022) employs diversified topological transforms for effective message passing in DAGs, enhancing scheduling efficiency through adaptive attention mechanisms. GoSU (Lee et al. 2021) integrates GCN and DRL to adaptively schedule DAG tasks by prioritizing complex dependencies, achieving reduced makespan and enhanced efficiency across diverse system configurations. However, these methods lack rigorous guarantees in unseen deployment scenarios.

Monte Carlo Tree Search is a heuristic search algorithm that combines the principles of tree search and Monte Carlo simulation. Beyond its well-documented success in games like Go (Silver et al. 2017), MCTS has demonstrated remarkable versatility across space(Kemmerling et al. 2024) including chemical compound design, robotics, and optimization problems. As for cloud workflow scheduling, researchers (Kung et al. 2022) propose to utilize existing heuristic algorithm results as initial upper bounds for MCTS and implement novel pruning strategies, but these approaches demonstrate inconsistent performance across diverse workflow configurations. Spear (Hu et al. 2019) first integrates a DRL agent to the expansion and simulation phases of MCTS, but fails to leverage the critical topological structure of workflow DAGs. Lore (Peng et al. 2022) employs GCN to learn DAG structures and train a DRL agent to guide MCTS, but requires extensive parameter input and hyperparameter optimization, which is hard to train on large-scale workflows.

3 PROBLEM DEFINITION AND MODEL

The cloud workflow scheduling problem can be formulated as follows: given a computational workflow represented by a task set $\mathscr{A} = \{1, 2, ..., n+1\}$ deployed on cloud infrastructure, each task $i \in \mathscr{A}$ requires processing duration d_i and consumes resource quantities r_{ik} for each resource type $k \in \mathscr{R}$ during runtime. Our goal is to determine each task's start time S_i within the discrete time horizon $\mathscr{T}$ while satisfying the dependency constraints under the workflow DAG structure and the cloud infrastructure resource capacity constraints, so as to minimize the overall completion time (*makespan*) of the workflow:

$$\begin{aligned}
\text{minimize} \quad & S_{n+1} && && (1)\\
\text{subject to} \quad & S_i \le t + M(1 - x_{it}), && \forall i \in \mathscr{A}, t \in \mathscr{T} && (2)\\
& S_i \ge t - d_i + 1 - M(1 - x_{it}), && \forall i \in \mathscr{A}, t \in \mathscr{T} && (3)\\
& \sum_{t \in \mathscr{T}} x_{i,t} = d_i, && \forall i \in \mathscr{A} && (4)\\
& \sum_{i \in \mathscr{A}} r_{ik} x_{it} \le C_k, && \forall k \in \mathscr{R}, t \in \mathscr{T} && (5)\\
& S_j \ge S_i + d_i, && \forall i \in \mathscr{A}, j \in \mathrm{Succ}(i) && (6)\\
& e_i \le S_i \le T_{\max} - d_i. && \forall i \in \mathscr{A} && (7)
\end{aligned}$$

The objective function in equation (1) minimizes the makespan, represented by the completion time of the last virtual task $n+1$. Equations (2) and (3) define a binary decision variable x_{it}, indicating whether task i is active at time t; these constraints ensure that $x_{it} = 1$ if and only if task i is being processed at time t (i.e., $S_i \leq t < S_i + d_i$). Equation (4) guarantees that each task runs continuously for exactly its required duration. Equation (5) guarantees that the total resource consumption across all active tasks at any time does not exceed the available capacity for each resource type. Equation (6) maintains the workflow's DAG structure by ensuring that successors of task i, i.e., Succ(i), can only begin after task i completes; in particular, the virtual task $n+1$ with no processing time is the successor of any other task and has no successor for itself, which makes it always the last task of the workflow. Equation (7) enforces the earliest possible start time e_i for each task i and ensures the workflow completes within the scheduling horizon $T_{\max}$.

4 METHODOLOGY

In this section, we present our novel GNN-Heatmap augmented Monte Carlo Tree Search algorithm for cloud workflow scheduling. We begin with an overview of classic Monte Carlo Tree Search and identify its limitations in scheduling contexts. Next, we detail the construction of our Graph Neural Network Heatmap that captures workflow dependencies. Finally, we demonstrate how this GNN-Heatmap integrates with MCTS to enhance search efficiency and solution quality.

4.1 Monte Carlo Tree Search

Monte Carlo Tree Search is a simulation-based search algorithm that effectively navigates complex decision spaces by balancing exploration and exploitation. It operates in four iterative phases: *Selection*, *Expansion*, *Simulation*, and *Backpropagation*.

Selection: Starting from the root, the algorithm recursively selects the most promising child node until reaching a leaf. This process is guided by the Upper Confidence Bound for Trees (UCT) policy (Kocsis and Szepesvári 2006), which balances exploiting nodes with high estimated values, $Q(s,a)$, against exploring less-visited nodes:

$$UCT(s,a) = Q(s,a) + c\sqrt{\frac{\ln(N_p)}{N(s,a)}},$$

where N_p is the parent's visit count, $N(s,a)$ is the current node's visit count, and c is the exploration parameter.

Expansion: Once a leaf node is reached, the algorithm expands the node by adding child nodes representing possible actions from the current state. This involves creating new nodes based on the search tree's current state and available actions. In classic MCTS, expansion typically occurs by randomly selecting one available action to generate a new child node.

Simulation: The algorithm performs a simulation from the newly expanded node to estimate the potential outcome of that node. This is done by running a quick rollout. In classic MCTS, it is often done by randomly selecting an available action until a terminal state is reached. The simulation result is used to evaluate the quality of the node and its potential impact on the overall search process.

Backpropagation: After the simulation, the algorithm backpropagates the simulation result up the tree, updating the value of the nodes along the path from the leaf node to the root node. The state value of a node is updated by the simulation results of its child nodes, and the visit count is incremented by one.

Adapting MCTS from game-playing to optimization problems like scheduling requires two key modifications (Xing and Tu 2020):

Value Recording: Instead of tracking average outcomes (e.g., win rates), we record the best value (minimum makespan) found, as optimization prioritizes extreme performance over average results.

Value Normalization: Since makespan values can span arbitrary ranges, rewards are normalized among sibling nodes to fit the UCT framework without extensive parameter tuning. The normalized value $Q_{norm}(s,a)$

is calculated as:

$$Q_{norm}(s,a) = \frac{Q(s,a) - Q_w(s)}{Q_b(s) - Q_w(s)},$$

where $Q_b(s)$ and $Q_w(s)$ are the best (minimum) and worst (maximum) makespan values among the children of state s. This scales values to a [0, 1] range, preserving the exploration-exploitation balance.

However, classic MCTS algorithm mentioned above has some limitations. In the expansion step, the algorithm randomly selects one of the available actions, which may lead to inefficient searches and slow convergence. In the simulation step, the algorithm randomly selects an available action until a terminal state is reached, which may not accurately represent the true value of the node. Thus, we can train a neural network to learn patterns from historical scheduling structures and use this knowledge to guide the MCTS search process, potentially improving convergence speed and solution quality.

4.2 Graph Neural Network Heatmap

4.2.1 Anisotropic Graph Neural Network Encoding

Cloud workflow architectures can be modeled as the DAG denoted by $G = (V,E)$, wherein the vertex set V comprises computational tasks, and the directed edge set E encodes the execution dependencies between these interconnected tasks. Graph Neural Networks have emerged as a powerful framework for learning on graph structured data, enabling the extraction of meaningful representations from complex relational structures. Stacking L GNN layers allows for the aggregation from L-hop neighbors.

We denote the d-dimensional feature vector of node i at layer ℓ as x_i^ℓ, and for each edge connecting nodes i and j, we denote the edge feature as w_{ij}^ℓ. To facilitate efficient message passing and feature aggregation for DAG structure, we introduce *extended edges* and *self-loop edges*, shown in the left of Figure 1. Extended edges allow nodes to propagate information beyond original dependencies, while self-loop edges ensure that each node retains a portion of its original feature during updates. This structure is critical for DAG's architecture, enabling efficient long-range information flow and preserving essential node-specific features during message passing. In this paper, the input features of the node are the task resource requirements and its duration, and the input features of the edge are varying from $[1,0,0]$, $[0,1,0]$ and $[0,0,1]$, representing the type of the edge among *original dependencies*, *extended edges* and *self-loop edges*.

After establishing this enhanced graph structure, we implement the Anisotropic Graph Neural Network encoding recommended by (Joshi et al. 2019) as follows:

$$x_i^{\ell+1} = x_i^\ell + \mathrm{RELU}\Big(\mathrm{BN}\Big(U^\ell x_i^\ell + \mathrm{AGGR}_{j\in\mathcal{N}_i}\Big(\sigma(w_{ij}^\ell)\odot V^\ell x_j^\ell\Big)\Big)\Big), \tag{8}$$

$$w_{ij}^{\ell+1} = w_{ij}^\ell + \mathrm{RELU}\Big(\mathrm{BN}\Big(R^\ell w_{ij}^\ell + S^\ell x_i^\ell + T^\ell x_j^\ell\Big)\Big), \tag{9}$$

where $U^\ell, V^\ell, R^\ell, S^\ell, T^\ell \in \mathbb{R}^{d\times d}$ are learnable parameters, $\mathcal{N}_i$ represents the neighborhood of node i, AGGR is the neighborhood aggregation function, σ is the sigmoid function that gates the importance of edge features, $\odot$ is the element-wise multiplication, and BN is the batch normalization function that stabilizes training. On the right of Figure 1, we show the message passing process of the Anisotropic GNN. In each update step, nodes aggregate information from their neighbors (yellow arrows) while edges dynamically adjust their features based on node interactions (red arrows). This bidirectional updating process allows nodes and edges to refine their representations iteratively, capturing complex dependencies in the workflow graph structure.

4.2.2 Process of Decoding

The decoding process transforms the final GNN edge features, w_{ij}^L, into a heatmap that provides probability scores for each edge, guiding the scheduling process. This transformation is achieved through a multi-layer

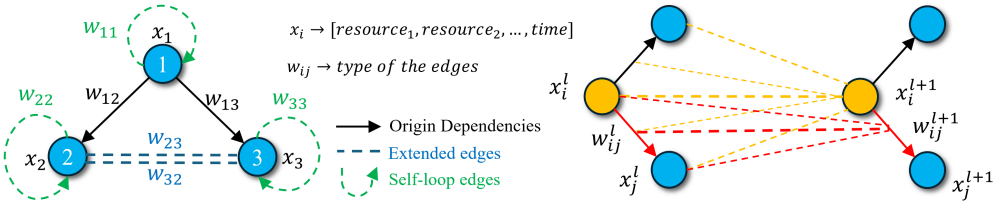

Figure 1: Left: original dependencies, extended edges and self-loop edges for DAG; Right: message passing of Anisotropic GNN.

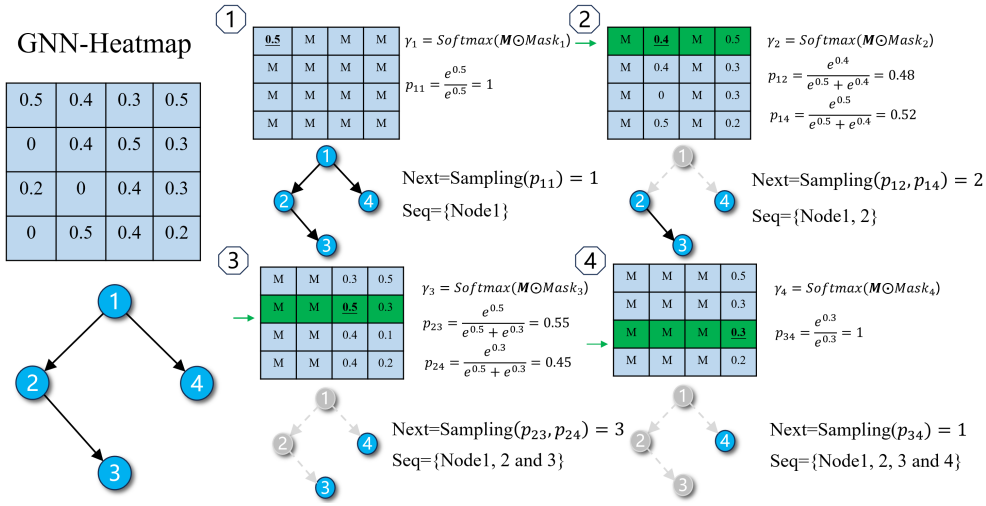

Figure 2: Description of the solution construction process of GNN-Heatmap using sampling methods.

fully connected network. First, the edge features w_{ij}^L are transformed into the initial heatmap layer M^0. This is followed by $H-1$ hidden layers processed with RELU activation functions. Finally, an output layer with a Sigmoid function compresses the tensor into a two-dimensional edge score matrix $M^H \in \mathbb{R}^{n\times n\times 1}$, as follows:

$$\begin{aligned} M^0 &= \text{transform}\left(w_{ij}^L\right), \quad M^0 \in \mathbb{R}^{n\times n\times d} \\ M^h &= \text{RELU}\left(W_f^h M^{h-1} + b_f^h\right), \quad h = 1,\ldots,H-1 \\ M^H &= \sigma\left(W_f^H M^{H-1} + b_f^H\right). \quad M^H \in \mathbb{R}^{n\times n\times 1} \end{aligned} \tag{10}$$

After obtaining the Anisotropic Graph Neural Network Heatmap (**GNN-Heatmap**), we need to use it to generate the scheduling solution. Figure 2 shows an example of stochastic sampling methods. It starts from the root node, and iteratively selects the next task to be scheduled based on the probability scores of the GNN-Heatmap. The Mask matrix is used to ensure that the selected task is valid and satisfies the dependency constraints. Other sampling methods such as greedy search and beam search will be discussed in the Section 5.2.

4.2.3 Training Strategy

In this paper, we use Reinforcement Learning to train the GNN-Heatmap. The training process is based on the policy gradient, which aims to minimize the expected makespan C given the workflow instance d:

$$\mathscr{L}(\theta \mid d) = \mathbb{E}_{\pi\sim\pi_\theta}\left[C(\pi)\right],$$

where π_θ is the policy network parameterized by θ. We can calculate the policy gradient using the REINFORCE algorithm (Williams 1992):

$$\nabla_\theta \mathscr{L}(\theta \mid d) = \mathbb{E}_{\pi\sim\pi_\theta}\left[(C(\pi) - b(d)) \cdot \nabla_\theta \log \pi_\theta\right],$$

where $b(d)$ is the baseline function, which is used to reduce the variance of the policy gradient. In practice, we use the Monte Carlo method to estimate the expected makespan $C(\pi)$ and the baseline function $b(d)$. First we stochastically sample T trajectories τ from the GNN-Heatmap, and then we calculate the expected makespan $C(\tau)$ and the baseline function $b(d)$ using the greedy search method. To eliminate the mean differences between batches and reduce the variance, we adopt the central self-critical form recommended by (Ma et al. 2019) and (Xiao et al. 2024):

$$\nabla \mathscr{L}(\theta) = \frac{1}{B}\sum_{i=1}^{B}\left[(C(\pi) - b(d) - \omega) \cdot \nabla \log \pi_\theta\right],$$
$$\omega = \frac{1}{B}\sum_{i=1}^{B}(C(\pi) - b(d)),$$

where B is the batch size. The overall training process is shown in Algorithm 1.

Algorithm 1 Training Process for GNN-Heatmap

Require: Training iterations I, steps per iteration S, mini-batch size B, validation size V

Initialize model parameters θ, set best validation loss $\mathscr{L}^* \leftarrow +\infty$
Generate validation instances D_{val} with size V
for iteration $i = 1$ **to** I **do**
    for optimization step $k = 1$ **to** S **do**
        Sample training batch D_{tr} with B instances
        Compute node features x_i^ℓ and edge features w_{ij}^ℓ with equation (8) and (9)
        Obtain the GNN-Heatmap M with the decoding process in equation (10)
        Compute stochastic policy π_θ, trajectories $\tau \leftarrow T$ times stochastic sampling(M, T)
        Generate baseline solution $s^* \leftarrow$ Greedy Search(M)
        Calculate policy gradient $\nabla_\theta \mathscr{L}(\theta) \leftarrow \frac{1}{B}\sum_{i=1}^{B}[(C(\tau) - b(d) - \omega) \cdot \nabla_\theta \log \pi_\theta]$
        Update parameters via $\theta \leftarrow$ Optimizer$(\theta, \nabla_\theta \mathscr{L}(\theta))$
    Evaluate on validation set: $M_{\text{val}} \leftarrow f_\theta(D_{\text{val}})$
    Compute validation solutions $s_{\text{val}} \leftarrow T$ times stochastic sampling(M_{val}, T)
    if $\mathbb{E}[C(s_{\text{val}})] < \mathscr{L}^*$ **then**
        Update $\mathscr{L}^* \leftarrow \mathbb{E}[C(s_{\text{val}})]$
        Store best parameters $\theta^* \leftarrow \theta$

4.3 Augmented Monte Carlo Tree Search

After obtaining the GNN-Heatmap, we can use it to augment the classic MCTS mentioned in Section 4.1. In the *selection* step, we can use the GNN-Heatmap to guide the search process. Inspired by methods that incorporate prior knowledge into the search policy (Silver et al. 2017), we integrate the GNN-Heatmap's probability score $P(s,a)$ into the UCT formula:

$$UCT_{GNN-Heatmap}(s,a) = Q(s,a) + cP(s,a)\sqrt{\frac{\ln(N_p)}{N(s,a)}},$$

where $P(s,a)$ is the prior probability provided by the GNN-Heatmap. This modified policy balances exploitation by favoring nodes with high estimated values ($Q(s,a)$) and exploration by prioritizing moves that are both promising (high $P(s,a)$) and less-visited. In the *simulation* step, instead of randomly selecting an available action in each step of the rollout, we can use the GNN-Heatmap to sample the available action with the higher probability score. This can help accurately represent the true value of the node, thus leading the search process to the optimal solution. The overall framework of GNN-Heatmap augmented MCTS is illustrated in Figure 3.

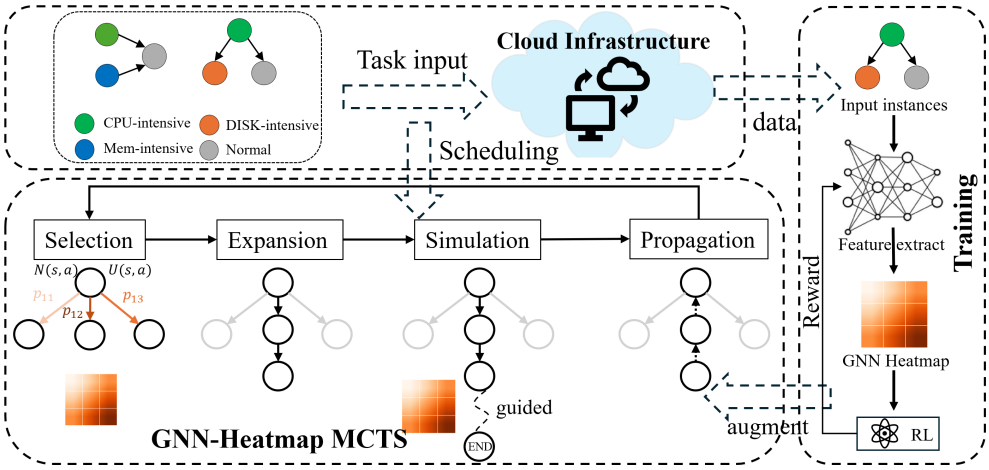

Figure 3: GNN-Heatmap augmented Monte Carlo Tree Search.

5 EXPERIMENTS

5.1 Generation of Workflow

In this paper, we adopt and implement the workflow generation methods from (Peng et al. 2022) and (Arabnejad and Barbosa 2014) to generate training data and test data for evaluating the performance of the proposed method. The generation rules are as follows:

n: Reflects the number of tasks in the DAG.

fat: This parameter governs the structural proportions of the DAG and is selected from {0.5,1,1.5,2,3}. It influences the task distribution across DAG layers, where the number of tasks per level approximates a normal distribution with mean $\mu = \sqrt{\text{n}}/\text{fat}$. The DAG's height (total number of layers) expands until all **n** tasks have been incorporated into the structure. Larger **fat** values produce DAGs with greater breadth and enhanced parallelization opportunities, while smaller values create taller, more sequential structures with constrained parallel execution possibilities.

density: Determines the maximum out-degree of a task, selected from {2, 3, 4, 5}. This parameter controls the number of edges between consecutive levels, with lower values resulting in sparse connectivity and higher values creating dense dependency networks.

regularity: Controls the uniformity of task distribution across levels, represented as the variance σ^2 of the normal distribution that generates the number of tasks per level. Values range from {0.5,1,2,3,4}, where lower values indicate heterogeneous levels and higher values produce more uniform task distributions.

Here we give an example of a DAG generated by the above parameters in Figure 4.

We consider three typical cloud computing resources (CPU, memory, and disk storage) with capacity limits $C_{CPU} = 24$, $C_{MEM} = 64$ GB, and $C_{DISK} = 1000$ GB, simulating modern cloud infrastructure configurations. Resource requirements are modeled based on four distinct task types, with equal probability: compute-intensive, memory-intensive, storage-intensive, and non-intensive tasks. Intensive tasks are as-

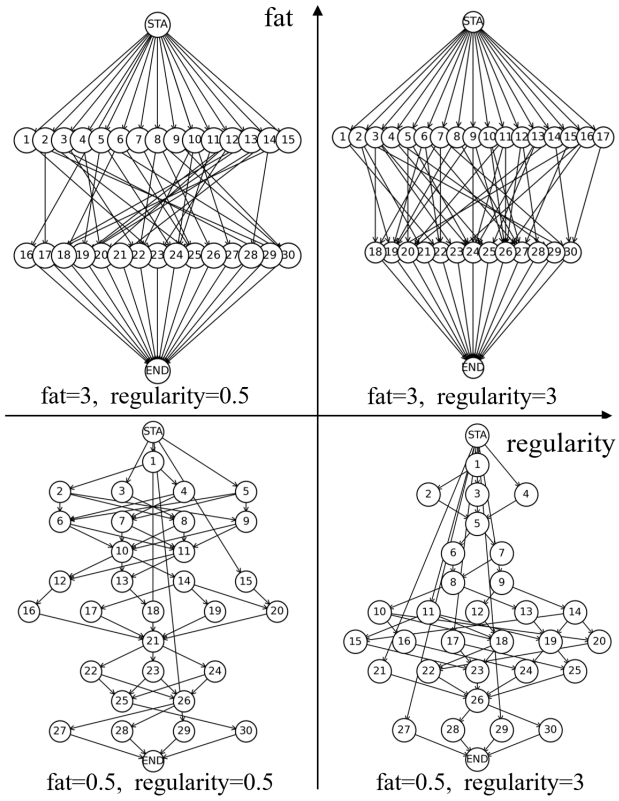

Figure 4: Example of a parameterized workflow DAG. The *STA* and *END* nodes represent dummy source and sink nodes, to standardize workflow entry and exit points.

signed randomly from 25% to 50% of the corresponding resource capacity, while non-intensive tasks are assigned from 2% to 10%. The task duration is uniformly distributed between 3 and 12 minutes, simulating realistic task complexities.

5.2 Baselines

The proposed method aimed at minimizing the *makespan* is compared with the following baselines:

CP: The Critical Path (CP) method, a classic heuristic that prioritizes tasks on the longest execution path of the workflow. To implement this, we adopt the upward ranking value from the Heterogeneous Earliest Finish Time (HEFT) algorithm (Topcuoglu et al. 2002):

$$priority(v_i) = w_i + \max_{v_j \in succ(v_i)} \{priority(v_j)\},$$

where w_i represents the execution time of task v_i, and $succ(v_i)$ denotes the set of immediate successor tasks of v_i. HEFT schedules tasks in descending order of priority to ensure most critical tasks are allocated first.

Graphene: A scheduling algorithm that focuses on the complex dependency structures of the workflow and heterogeneous resource requirements (Grandl et al. 2016). It first identifies "troublesome tasks" - those with long execution times or complex resource requirements that could become bottlenecks. These are identified using LongScore (task duration divided by maximum task duration) and FragScore (minimum ratio of resource capacity to task demand across all resource types). The algorithm then partitions the workflow DAG into four distinct subsets and schedules them in a carefully determined order that ensures optimal resource utilization while preventing potential execution bottlenecks.

IP: Integer Programming. In this paper, we use the Gurobi solver to solve the Integer Programming model defined in Section 3.

Classic MCTS: Classic Monte Carlo Tree Search algorithm, which is defined in the Section 4.1.

Random search: A random search algorithm that randomly selects an available task at each step. It is not guided by any heuristic or prior knowledge. This process is inherently parallelized.

Stochastic sampling: Based on the probability scores of the GNN-Heatmap, the stochastic sampling algorithm randomly selects an available task at each step, with the probability of selecting each task proportional to its score. It allows for exploration of different scheduling configurations, potentially leading to diverse solutions. This process is inherently parallelized.

Greedy search: Based on the probability scores of the GNN-Heatmap, the greedy search algorithm selects an available task with the highest score at each step.

Beam search: Based on the probability scores of the GNN-Heatmap, the beam search algorithm maintains a fixed number of the best candidate solutions at each step. It explores multiple paths simultaneously, allowing for a more comprehensive search of the solution space. This process is inherently parallelized.

End2End: a neural network-based approach, which uses an Encoder-Decoder architecture to generate scheduling solutions directly from the workflow DAG. In this paper, the Encoder part is based on the Anisotropic Graph Neural Network encoding method described in Section 4.2.1, and for the Decoder part, we refer to the procedures in (Kool et al. 2019) and (Cai et al. 2024):

At each step t, the Decoder constructs an initial context vector $\hat{x}_t^C$ by concatenating the embeddings of the last scheduled task ($x^L_{\pi_{t-1}}$), available tasks (x_{ava}), and the global graph (x_G). This vector then serves as the query in a standard Multi-Head Attention (MHA) operation:

$$x_i^C = \text{MHA}(Q = \hat{x}_t^C, K = \{x_1^L, \ldots, x_n^L\}, V = \{x_1^L, \ldots, x_n^L\}),$$

where Q, K, and V are the query, key, and value inputs for the M-headed attention mechanism ($M = 8$). The resulting context-aware embedding x_i^C is used to compute attention scores z_{ij}, which are then normalized into probabilities $\hat{p}_{ij}$:

$$z_{ij} = \begin{cases} C \cdot \tanh\left(\frac{(W_Q x_i^C)^T (W_K x_j^L)}{\sqrt{d}}\right) & \text{if task } j \text{ is available} \\ -\infty & \text{otherwise} \end{cases},$$

$$\hat{p}_{ij} = \text{Softmax}(z_{ij}/T).$$

A clipping coefficient C (typically 10) prevents exploding gradients. The temperature T controls the distribution's sharpness, with higher values ($T = 1$) for exploration during training and lower values ($T = 0.6$) for exploitation during evaluation.

5.3 Experimental Settings

In the encoding process, we set the number of GNN layers $L = 8$ and the dimension of the GNN model $d = 64$. In the decoding process, we set the number of fully connected layers $H = 3$. In the training process, we set the batch size $B = 16$, the validation size $V = 20$, the training iterations $I = 150$, and the steps per iteration $S = 16$. We use the Adam optimizer with a learning rate of 3×10^{-4}. As for the MCTS, we set the number of simulations in each layer $N = 300$ and the coefficient $c = 1.4$.

6 RESULTS

6.1 Makespan Performance

We compare the performance of the proposed GNN-Heatmap augmented MCTS with the baselines on the workflow scheduling problem. We conduct our experiments on the generated workflows mentioned in

Section 5.1, with different sizes of 30, 50, 70, and 110 tasks. For each size of the workflow, we generate 500 instances for training and 50 instances for testing.

Figure 5 illustrates the average makespan performance across different workflow sizes. For these learning-based methods (greedy search, beam search, stochastic sampling, End2End), they perform better than rule-based methods (CP and Graphene). The Anisotropic GNN demonstrates strong feature extraction capability from DAG structures, enabling these methods to obtain near-optimal solutions in a short time. However, they are not as good as the classic MCTS. Our proposed method, which leverages the Anisotropic GNN-Heatmap to augment the classic MCTS, consistently outperforms other baseline methods across all workflow sizes. For small workflows (30 tasks), our method shows very competitive performance compared with the IP. As workflow size increases to 50, 70, and 110, our method demonstrates the best performance, achieving the lowest makespan among all methods.

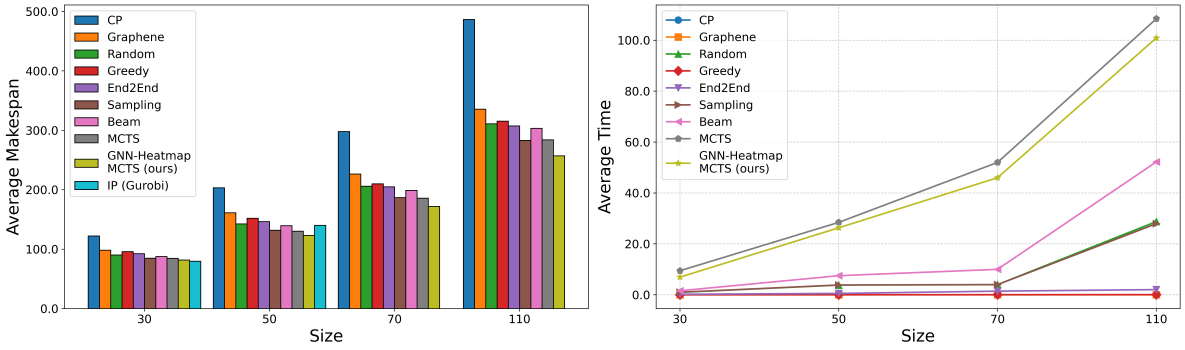

Figure 5: Performances of different methods. Left: Average makespan performances. Random, sampling and beam search are run with $300 \times n$ parallel searches. We solve the IP model using Gurobi for the sizes of 30 and 50. Right: Average evaluation time. We set the time limit of Gurobi for 15 minutes and 30 minutes for the sizes of 30 and 50 respectively, with 5% tolerance relative gap.

Table 1 presents the performance relative gap of each algorithm across different workflow sizes, where the gap is calculated as the difference between the solution makespan and optimal makespan, normalized by the optimal makespan. Based on IP solution tolerance gaps below 5%, we categorize instances into optimal (*Opt.*), non-optimal (*Not Opt.*), and all the instances (*All*). For size 30 instances with optimal IP solutions (3.36% tolerance), our method achieves a mere 1.99% gap from IP solutions. For size 50 instances with optimal IP solutions (3.49% tolerance), our method maintains a 3.30% gap; remarkably, when IP solutions are suboptimal (28.49% tolerance), our method outperforms IP. For larger instances (70 and 110 tasks) where IP fails to converge, our method consistently achieves optimal performance against all baselines. These results demonstrate that our approach reliably converges to high-quality solutions across both optimally solved and challenging instances, indicating superior robustness and convergence capability.

Table 1: Performance relative gap, with lower values indicating better performance. Best performance (0%) is marked in bold.

size	**IP Status (Average tolerance relative gap)**	**Average Performance Relative Gap**									
		CP	**Graph-ene**	**Random**	**Samp-ling**	**Greedy**	**Beam**	**End2 End**	**MCTS**	**GNN-MCTS**	**IP**
30	Opt. (3.36%)	50.88%	21.27%	11.86%	5.62%	19.07%	8.76%	14.87%	5.39%	1.99%	**0.00%**
	Not Opt. (7.93%)	66.20%	33.97%	19.76%	10.57%	27.21%	15.60%	21.75%	9.36%	4.89%	**0.00%**
	All (4.18%)	53.09%	23.10%	13.00%	6.34%	20.25%	9.75%	15.86%	5.96%	2.56%	**0.00%**
50	Opt. (3.49%)	59.32%	29.00%	15.92%	8.71%	25.37%	14.97%	20.07%	7.23%	3.30%	**0.00%**
	Not Opt. (28.49%)	74.49%	26.66%	18.38%	8.15%	24.35%	14.21%	17.37%	6.77%	**0.00%**	29.51%
	All (17.99%)	64.89%	30.83%	15.64%	7.00%	23.21%	13.08%	18.75%	5.59%	**0.00%**	13.60%
70	All (-)	73.26%	31.76%	19.77%	8.72%	22.09%	15.70%	19.19%	8.14%	**0.00%**	-
110	All (-)	89.21%	30.52%	20.96%	10.09%	22.64%	17.95%	19.51%	10.46%	**0.00%**	-

6.2 Scalability

We also evaluate the scalability of the proposed method on unseen workflow sizes. Specifically, we construct a mixed training dataset consisting of 250 instances of size 30 workflows and 250 instances of size 50 workflows (500 instances total), and evaluate its performance on completely unseen workflow sizes: 60, 70, and 90 tasks, with each size containing 50 instances.

Table 2 shows the performance relative gap of the proposed method and the baselines on unseen workflow sizes. The performance relative gap of the Graphene, classic MCTS, and parallelized searching methods (Random, Sampling) is stable across different workflow sizes. However, the performance relative gap of the learning-based methods (greedy search, beam search, End2End) increases significantly. For example, the greedy search algorithm's gap increases from 23.87% at size 60 to 30.10% at size 90, while beam search deteriorates from 20.42% to 26.86%. In contrast, our method consistently maintains optimal performance across all unseen workflow sizes, demonstrating superior scalability compared to pure learning-based approaches, which struggle with structural variations in larger workflows.

Table 2: Performance relative gap demonstrating the scalability. The training dataset is composed of workflow sizes 30 and 50 only. Best performance (0%) is marked in bold.

size	CP	Graphene	Random	Sampling	Greedy	Beam	End2End	MCTS	GNN-MCTS
60	82.13%	28.60%	18.35%	9.33%	23.87%	20.42%	21.70%	7.38%	**0.00%**
70	73.58%	30.27%	18.65%	9.85%	26.00%	21.38%	24.84%	8.40%	**0.00%**
90	87.60%	29.51%	21.96%	11.47%	30.10%	26.86%	28.13%	10.37%	**0.00%**

7 CONCLUSION AND DISCUSSION

In this paper, we propose a GNN-Heatmap augmented Monte Carlo Tree Search algorithm, to solve the cloud workflow scheduling problem in heterogeneous environments. The GNN-Heatmap is designed to learn the complex dependencies of the workflow and augment the classic MCTS in the selection and simulation steps. The experimental results show that our proposed method is superior to the baselines in terms of makespan performance and better scalability on unseen workflow sizes.

However, there are still many potential extensions for this work. First, while our method demonstrates robust performance across synthetic workflows with varying structural parameters, future work could explore GNN-Heatmap behavior across distinct DAG topologies (sparse vs. dense, layered vs. irregular) and analyze failure cases where neural guidance misguides MCTS to enhance reliability. Second, to further strengthen the practical impact of our findings beyond synthetic distributions, future research could focus on validating our approach using actual workflows sourced from cloud providers.

Fundamentally, our approach advances Monte Carlo simulation techniques for complex optimization across diverse domains, offering valuable insights for network traffic orchestration, manufacturing systems, and healthcare operations modeling.

ACKNOWLEDGMENTS

This research was funded by the Guangdong Pearl River Plan (2019QN01X890).

REFERENCES

Arabnejad, H., and J. G. Barbosa. 2014. "List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table". *IEEE Transactions on Parallel and Distributed Systems*:682–694 https://doi.org/10.1109/TPDS.2013.57.

Cai, H., Y. Bian, and L. Liu. 2024. "Deep Reinforcement Learning for Solving Resource Constrained Project Scheduling Problems with Resource Disruptions". *Robotics and Computer-Integrated Manufacturing* 85 https://doi.org/10.1016/j.rcim.2023.102628.

Gagrani, M., C. Rainone, Y. Yang, H. Teague, W. Jeon, R. Bondesan, *et al*. 2022. "Neural Topological Ordering for Computation Graphs". *Advances in Neural Information Processing Systems*:17327–17339.

Grandl, R., S. Kandula, S. Rao, A. Akella, and J. Kulkarni. 2016. "GRAPHENE: Packing and Dependency-Aware Scheduling for Data-Parallel Clusters". In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 81–97.

Hu, Z., J. Tu, and B. Li. 2019. "Spear: Optimized Dependency-Aware Task Scheduling with Deep Reinforcement Learning". In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 2037–2046: IEEE https://doi.org/10.1109/ICDCS.2019.00201.

Jalali Khalil Abadi, Z., N. Mansouri, and M. M. Javidi. 2024. "Deep Reinforcement Learning-Based Scheduling in Distributed Systems: A Critical Review". *Knowledge and Information Systems*:5709–5782 https://doi.org/10.1007/s10115-024-02167-7.

Joshi, C. K., T. Laurent, and X. Bresson. 2019. "An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem". *arXiv preprint arXiv:1906.01227* https://doi.org/10.48550/arXiv.1906.01227.

Kemmerling, M., D. Lütticke, and R. H. Schmitt. 2024. "Beyond Games: A Systematic Review of Neural Monte Carlo Tree Search Applications". *Applied Intelligence*:1020–1046 https://doi.org/10.1007/s10489-023-05240-w.

Kocsis, L., and C. Szepesvári. 2006. "Bandit Based Monte-Carlo Planning". In *Machine Learning: ECML 2006*, 282–293 https://doi.org/10.1007/11871842_29.

Kool, W., H. Van Hoof, and M. Welling. 2019. "Attention, Learn to Solve Routing Problems!". *arXiv preprint arXiv:1803.08475* https://doi.org/10.48550/arXiv.1803.08475.

Kung, H.-L., S.-J. Yang, and K.-C. Huang. 2022. "An Improved Monte Carlo Tree Search Approach to Workflow Scheduling". *Connection Science*:1221–1251 https://doi.org/10.1080/09540091.2022.2052265.

Lee, H., S. Cho, Y. Jang, J. Lee, and H. Woo. 2021. "A Global DAG Task Scheduler Using Deep Reinforcement Learning and Graph Convolution Network". *IEEE Access*:158548–158561 https://doi.org/10.1109/ACCESS.2021.3130407.

Ma, Q., S. Ge, D. He, D. Thaker, and I. Drori. 2019. "Combinatorial Optimization by Graph Pointer Networks and Hierarchical Reinforcement Learning". *arXiv preprint arXiv:1911.04936* https://doi.org/10.48550/arXiv.1911.04936.

Peng, H., C. Wu, Y. Zhan, and Y. Xia. 2022. "Lore: A Learning-Based Approach for Workflow Scheduling in Clouds". In *Proceedings of the Conference on Research in Adaptive and Convergent Systems*, 47–52 https://doi.org/10.1145/3538641.3561487.

Silver, D., J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, *et al*. 2017. "Mastering the Game of Go without Human Knowledge". *Nature*:354–359 https://doi.org/10.1038/nature24270.

Topcuoglu, H., S. Hariri, and M.-Y. Wu. 2002. "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing". *IEEE Transactions on Parallel and Distributed Systems*:260–274 https://doi.org/10.1109/71.993206.

Williams, R. J. 1992. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". *Machine Learning*:229–256 https://doi.org/10.1007/BF00992696.

Xiao, Y., D. Wang, B. Li, H. Chen, W. Pang, X. Wu, *et al*. 2024. "Reinforcement Learning-Based Nonautoregressive Solver for Traveling Salesman Problems". *IEEE Transactions on Neural Networks and Learning Systems*:1–15 https://doi.org/10.1109/TNNLS.2024.3483231.

Xing, Z., and S. Tu. 2020. "A Graph Neural Network Assisted Monte Carlo Tree Search Approach to Traveling Salesman Problem". *IEEE Access*:108418–108428 https://doi.org/10.1109/ACCESS.2020.3000236.

Zhou, G., W. Tian, R. Buyya, R. Xue, and L. Song. 2024. "Deep Reinforcement Learning-Based Methods for Resource Scheduling in Cloud Computing: A Review and Future Directions". *Artificial Intelligence Review*:124 https://doi.org/10.1007/s10462-024-10756-9.

AUTHOR BIOGRAPHIES

DINGYU ZHOU is a master's student at the Shenzhen International Graduate School (SIGS), Tsinghua University. His research interests include learning-based methods for cloud computing scheduling. His email address is zdy23@mails.tsinghua.edu.cn.

JIAQI HUANG is a Ph.D. candidate at the Shenzhen International Graduate School (SIGS), Tsinghua University. Her research interests include scheduling algorithms for cloud computing and manufacturing systems. Her email address is hjq23@mails.tsinghua.edu.cn.

YIRUI ZHANG is a master's student at the Shenzhen International Graduate School (SIGS), Tsinghua University. Her research interests include heuristic methods for cloud computing scheduling. Her email address is yr-zhang23@mails.tsinghua.edu.cn.

WAI KIN (VICTOR) CHAN is a Professor at the Shenzhen International Graduate School (SIGS), Tsinghua University. He is the corresponding author of this paper. His primary research focuses on computational modeling methodologies including discrete-event simulation and agent-based approaches, including social network analysis, service optimization, intelligent transportation, and advanced manufacturing processes. His email address is chanw@sz.tsinghua.edu.cn.