On the Role of DAG Structure in Energy-Aware Scheduling with Deep Reinforcement Learning

Anonymous authors
Paper under double-blind review

Abstract

Cloud providers must assign heterogeneous compute resources to workflow DAGs while balancing competing objectives such as completion time, cost, and energy consumption. In this work, we study a single-workflow, queue-free scheduling setting and consider a graph neural network (GNN)–based deep reinforcement learning scheduler designed to minimize workflow completion time and energy usage.

We identify specific out-of-distribution (OOD) conditions under which GNN-based deep reinforcement learning schedulers fail, and provide a principled explanation of why these failures occur. Through controlled OOD evaluations, we demonstrate that performance degradation stems from structural mismatches between training and deployment environments, which disrupt message passing and undermine policy generalization. Our analysis exposes fundamental limitations of current GNN-based schedulers and highlights the need for more robust representations to ensure reliable scheduling performance under distribution shifts.

Keywords: Cloud Computing, Resource Allocation, Deep Reinforcement Learning, Robustness, Interpretability, Job Scheduling

1 Introduction

1.1 Context

Over the past few years, artificial intelligence and large language models have pushed cloud systems much harder than before. According to International Energy Agency (2025), the energy used by data centers keeps growing because of heavier AI training and inference workloads. This means even small improvements in how cloud resources are allocated can save a noticeable amount of time and electricity.

Figure 1 shows the setting. A workflow is a directed acyclic graph. Nodes are tasks. Edges are data or control dependencies. Some stages expose wide parallelism. Others form long serial chains. At the same time, machines in a cloud are heterogeneous. Some machines finish tasks faster but draw more power. Others are slower but use less. The scheduler decides, at each step, which ready task should run on which machine. Each assignment changes the makespan and the total energy used.

This problem has drawn steady research interest. Energy-aware scheduling appeared in HPC and cloud literature well before the recent AI boom. Early work used DVFS to trade performance for power (Zong et al., 2007; Rountree et al., 2009). Recent surveys report steady activity: Multiple systematic reviews published indicate sustained research activity in this domain (Versluis & Iosup, 2020; Ajmera & Tewari, 2024). Machine learning-based cluster management has demonstrated measurable impact, reducing data center energy consumption by up to 15% in Google's production deployments (Google, 2016). Amazon and Microsoft have explored complementary approaches through right-sizing and spot market mechanisms to reduce both operational costs and resource waste (Cortez et al., 2017; Shahrad et al., 2020).

Modern AI workflows add new dimensions to this problem. They mix layers that are very wide with chains that are very long. And they run on machines with big differences in speed and power use. Old scheduling

rules assume the workflow shape stays the same and machines are predictable. When that is not true, performance can drop fast. Because of this, researchers are looking at more advanced schedulers that can adapt. But the problem is we do not know if these learned policies still work when the workflow or machines change. We will go over the current approaches next and then introduce the gap that motivated our study.

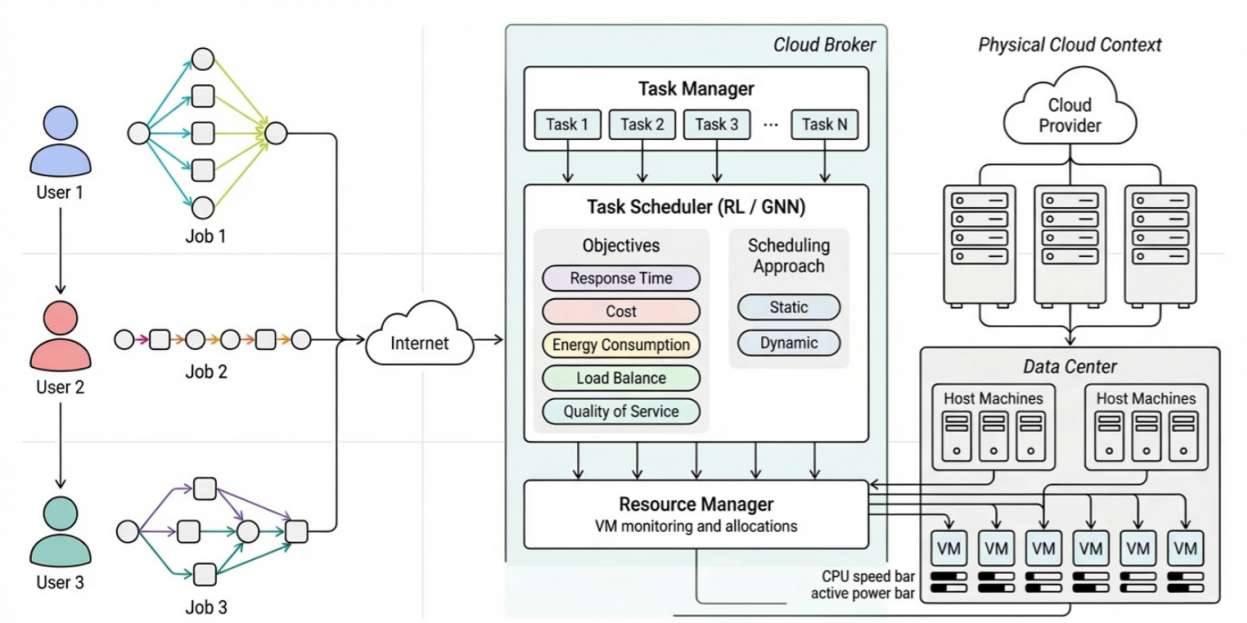

Figure 1: Overview of the workflow scheduling problem in a heterogeneous cloud. The scheduler receives a DAG with ready tasks and a set of machines with different speed and power. It must decide which task runs on which machine, and each choice changes both completion time and energy.

1.2 Related Work

Many workflow applications are naturally written as directed acyclic graphs (DAGs) of tasks with different resource needs and complex dependencies (Deelman et al. (2009), Sakellariou et al. (2010)). Scheduling these DAGs on pools of virtual machines with different speeds and power use is a classic problem in distributed systems and high performance computing (Mao & Humphrey (2012)). This problem is NP hard in general. Foundational work by Pinedo (2012) and the notation of Graham et al. (1979) provide the standard formal basis for makespan minimization under precedence and machine heterogeneity.

1.2.1 Classical Structure Aware DAG Scheduling

A large body of work models applications as weighted DAGs, where nodes are tasks and edges capture precedence and communication constraints on heterogeneous processors. The HEFT and CPOP list schedulers of Topcuoglu et al. (2002) are central references. They rank tasks with path based metrics and map them to heterogeneous processors to reduce makespan. HEFT uses an upward rank that approximates remaining critical path length, then schedules tasks on the processor with the earliest finish time. This exploits DAG depth, fan out, and communication along paths. CPOP identifies a global critical path and assigns its tasks to a single processor to reduce inter processor communication.

Many variants build on this structure aware idea while adding new objectives such as energy or reliability. Energy aware DAG schedulers on heterogeneous and embedded platforms often use critical path and level based slack to decide which tasks can safely be slowed down or consolidated while still meeting deadlines (Hu et al., 2023). In these works, DAG topology is not only a feasibility constraint. It is a direct signal for making decisions. Parallelism per level, critical path length, and slack structure strongly influence task priorities and mapping choices.

In cloud and distributed environments, workflow systems such as Pegasus (Dee) and simulators such as CloudSim (Calheiros et al., 2011) have established DAGs as the standard abstraction for scientific workflows. Energy aware scheduling has been widely studied in this setting. For example, Beloglazov & Buyya (2012) show how power heterogeneity and task structure together shape good placement policies.

Many real world workloads are also multi objective. Schedulers must trade off makespan, cost, energy, and sometimes reliability (Bittencourt & Madeira (2018), da Silva & Bittencourt (2017)). Classical heuristics are usually tuned around a single dominant objective. When new metrics are added, they often need extensive manual retuning and may not transfer well across workloads (Bittencourt & Madeira, 2018). Meta heuristic methods can explore richer trade offs but are often too slow for real time scheduling in dynamic clouds, since they rely on iterative search (da Silva & Bittencourt (2017)).

1.2.2 Deep Reinforcement Learning for Resource Scheduling

These limitations have motivated a shift toward learning based schedulers. Deep reinforcement learning (deep RL), first introduced by Sutton et al. (1998), offers a way to learn scheduling policies directly from experience without manually encoding rules for every scenario. Early work by Mao et al. (2016) showed that deep RL agents could learn to pack tasks and allocate resources in cluster settings, outperforming hand tuned heuristics on job completion time. The idea is to frame scheduling as a sequential decision problem. An agent observes system state, selects actions such as which task to schedule or which machine to use, and receives rewards based on performance metrics like makespan or resource use.

This approach has several advantages. First, the policy can adapt to patterns in the workload without explicit feature engineering. Second, it can handle multi objective trade offs by shaping the reward function. Third, once trained, the policy can make decisions quickly, which is useful in online settings. Several studies have confirmed that deep RL schedulers can match or beat classical heuristics in controlled environments (Zhang et al. (2021), Mao et al. (2016)).

But early deep RL schedulers often treated system state as a flat feature vector. This does not capture the relational structure of workflows or resource topologies. This is where graph neural networks come in.

1.2.3 Graph Neural Networks for Structured Scheduling

Graph neural networks provide a natural way to encode relational structure. In scheduling, both workflows (task dependencies) and resource topologies (machine connectivity or hierarchy) are naturally represented as graphs. GNNs use message passing to let each node aggregate information from its neighbors. This means the learned representation can respect the structure of the problem.

Decima was one of the first systems to combine GNNs with RL for cluster scheduling (Mao et al., 2019). It models each data processing job as a DAG of stages. A GNN embeds this DAG along with per stage features such as remaining work and resource demand. The RL policy then uses these embeddings to decide which stage to schedule and how many executors to assign. Message passing over the DAG lets the model implicitly capture properties like depth, critical paths, and fan in or fan out. Experiments showed that this structure aware policy reduced average job completion time compared to structure agnostic baselines and classical heuristics.

Follow up work extended this idea to other scheduling domains. Park et al. (2021) used GNNs to embed both job DAGs and cluster topology for multi resource scheduling. Others applied similar architectures to workflow scheduling in cloud and edge environments, where tasks have precedence constraints and machines have different speeds or power profiles. In most cases, the GNN based approach improved performance over flat feature representations. These results suggest that exploiting the graph structure provides richer relational information, allowing the policy to generalize more robustly across scenarios.

1.2.4 Generalization and Robustness Challenges

Despite these successes, most studies evaluate learned policies primarily on the same types of workflows and host configurations seen during training. They show that GNN based RL can work well in specific settings,

but they rarely ask how robust these policies are when the workflow structure or resource characteristics change. For example, what happens when a policy trained on shallow parallel workflows is tested on deep sequential workflows? Or when a policy trained on homogeneous machines is deployed on heterogeneous hardware with conflicting speed and power trade offs?

This concern is grounded in known limitations of graph neural networks under distribution shift. Wu et al. (2022) showed at ICLR 2022 that distribution shifts hit GNNs particularly hard because of how nodes connect to each other. When the graph structure changes, the whole representation can break down. This matters for workflow scheduling. A policy trained on one type of DAG may face very different graphs at deployment. Depth can change. Width can change. Branching can change. This motivates a systematic first step: characterize the distribution shift in our setting. We must precisely define and measure how deployment DAGs differ from those seen during training, and then determine how to address it.

1.3 Positioning of Our Work

This paper identifies specific out-of-distribution conditions that cause GNN-based deep RL schedulers to fail, and explains why these failures occur.

We build on two key observations. First, Gu et al. (2021) showed that real workflows from production systems (millions of DAGs from Alibaba batch jobs) cluster into a few structural types. Second, recent RL-based schedulers using GNNs to embed DAGs have shown strong performance on some benchmarks (Mao et al. (2019), Park et al. (2021)), but their robustness across structural types remains unclear.

Our starting point is an empirical pattern reported in Hattay et al. (2024). When a deep RL scheduler is compared with classical heuristics across many workflow instances, it does not simply dominate or fail everywhere. It performs very well on some workflows and host settings and quite poorly on others. Sometimes it is even worse than basic list scheduling rules. With closer examination, we observed that these failures are not random. They concentrate in specific combinations of workflow topology and host heterogeneity. This suggests that learned schedulers have implicit structural domains where they behave coherently and domains where their behavior degrades.

Epistemologically, we follow a Popper style view of scientific progress (Popper, 2005). We are less interested in showing more positive cases for a learned policy and more interested in subjecting it to tests that might break it. We treat the learned policy as a provisional theory about how structure and heterogeneity shape scheduling decisions. We then expose that theory to workflow topologies and host regimes in which it should fail if it is narrow or brittle. The goal is not only to show that the agent can work somewhere, but to reveal what it has actually learned when the surrounding conditions change.

To study this in a controlled way, we define two simple workflow families. *Wide* DAGs are shallow with high parallelism. *Long Critical Path (LongCP)* DAGs have deep dependency chains and little parallel slack. On the resource side, we consider four queue free host regimes that isolate different aspects of heterogeneity: Homogeneous Speed (HS), Homogeneous Power (HP), Heterogeneous Aligned (AL), and Heterogeneous Non Aligned (NA). Each regime creates a different trade off between makespan and active energy. As a result, each regime gives a different incentive for using or ignoring parallelism.

We then train a GNN based actor critic deep RL scheduler on these environments. We use separate agents specialized to Wide workflows and to LongCP workflows. We do not evaluate these agents only on the distributions they were trained on. Instead, we systematically probe cross structure and cross regime generalization. For example, we run a Wide trained agent on LongCP workflows, a LongCP trained agent on Wide workflows, and we test all agents across all four host regimes.

This work therefore investigates how DAG topology and host heterogeneity together shape the behavior and generalization of an RL based scheduler with joint energy and makespan objectives in a queue free, single workflow cloud setting.

Research Questions This setup lets us ask four main questions:

- **Q1:** How do DAG structure (wide vs. long critical path) and host speed/power configurations jointly influence learned policy priorities under mixed energy-makespan objectives? Do these factors induce systematic biases in scheduling strategies despite identical objectives?

- **Q2:** How does cross-structure generalization vary across host configurations (AL, NA, HS, HP)? When do Wide specialists outperform LongCP specialists (and vice versa), and what explains these performance gaps?

- **Q3:** Given generalization patterns, when do simple heuristics suffice versus when learned specialists are needed? What practical approach best handles diverse regime combinations?

Contributions By answering these questions, we make the following contributions:

- **A controlled decomposition of the problem space.** We separate the effects of workflow topology and host heterogeneity by defining two contrasting DAG families (Wide and LongCP) and four host regimes (HS, HP, AL, NA) that each capture a different dimension of heterogeneity.

- **Systematic cross structure and cross regime evaluation.** We train GNN based deep RL schedulers specialized to Wide workflow and to LongCP workflows. We then test all agents across all topology and regime combinations using Wide and LongCP test workflows per configuration.

- **An interpretability focused analysis of generalization domains.** We show that the combined structure of workflow and host naturally divides the problem into domains where a given policy behaves consistently and domains where its generalization breaks down. We analyze these domains using state space geometry and structural statistics to explain when and why policies fail.

In the following sections, we first formalize the problem and objectives (Section 2), then describe our benchmark and GNN based scheduler (Section 3), and finally evaluate and explain its behavior across workflow topologies and host regimes (Section 4).

2 Problem Setup

2.1 Problem Formulation

We adopt the MDP formulation from Chandrasiri & Meedeniya (2025) for workflow scheduling on virtualized clusters, with modifications to support concurrent task execution on multi-core VMs. While we retain the same state space, action space, and transition dynamics, we extend the makespan and active energy calculations to account for overlapping tasks running on a single VM and fractional CPU utilization when computing energy consumption and VM resource liberation times.

We formulate workflow scheduling on a virtualized cluster as a finite-horizon Markov Decision Process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ over decision epochs $k = 0, 1, \ldots, K$, aligned with scheduling decisions.

System Model. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a workflow DAG with tasks $i \in \mathcal{V}$ and precedence edges $(p \to i) \in \mathcal{E}$. Each task has: (i) computational size L_i (e.g., MI), (ii) resource demand vector $\mathbf{d}_i = (\mathrm{cpu}_i, \mathrm{mem}_i)$, (iii) compatibility set $\mathcal{C}_i \subseteq \mathcal{M}$ of admissible VMs. Each VM $m \in \mathcal{M}$ has capacity $\mathbf{c}_m = (C_m^{\mathrm{cpu}}, C_m^{\mathrm{mem}})$, processing speed s_m (e.g., MIPS), and power parameters $(P_m^{\mathrm{idle}}, P_m^{\mathrm{peak}})$.

Decision Epochs and Clock. Let τ_k denote the simulation clock at decision epoch k. Decisions occur when the agent assigns a ready task. Between decisions, the environment advances τ according to scheduled start/finish events implied by previous assignments.

State Space $\mathcal{S}$. A state $s_k \in \mathcal{S}$ summarizes all information needed for optimal control:

$$\begin{aligned} s_k = (\tau_k; &\{\text{task status}_i \in \{\text{not\_ready}, \text{ready}, \text{running}, \text{done}\}\}_{i\in\mathcal{V}}; \\ &\{\text{parent\_ready}_i = \max_{p\in\text{Pa}(i)} c_p\}_{i\in\mathcal{V}}; \{\text{assignment}_i \in \mathcal{C}_i \cup \{\varnothing\}\}_{i\in\mathcal{V}}; \{\text{start}_i,\, c_i\}_{i\in\mathcal{V}}; \\ &\{\text{VM residual capacities and active allocations over } [\tau_k, \infty)\}_{m\in\mathcal{M}}; \{\mathcal{C}_i\}_{i\in\mathcal{V}}). \end{aligned}$$

The ready set at τ_k is $\mathcal{R}_k = \{i : \text{ task status}_i = \text{ready}\}$.

Action Space $\mathcal{A}$. An action selects a task–VM pair:

$$a_k = (i, m) \in \mathcal{F}(s_k) \subseteq \mathcal{V} \times \mathcal{M},$$

where the feasible set enforces precedence, compatibility, and capacity:

$$\mathcal{F}(s_k) = \Big\{(i,m) : \ i \in \mathcal{R}_k,\ m \in \mathcal{C}_i,\ \mathbf{d}_i \preceq \text{residual\_capacity}_m(t) \text{ for some } t \geq \text{parent\_ready}_i\Big\}.$$

Operationally, assigning (i, m) schedules task i on VM m at its earliest feasible start time

$$s_i = \min\{t \geq \text{parent\_ready}_i : \ \mathbf{d}_i \preceq \text{residual\_capacity}_m(t)\}, \qquad c_i = s_i + \frac{L_i}{s_m},$$

and updates VM m's capacity timeline accordingly. In contrast to Chandrasiri & Meedeniya (2025), which assumes one task per VM at a time, our formulation allows multiple tasks to execute concurrently on VM m as long as the aggregate resource demands satisfy $\sum_{j\in A_m(t)} \text{cpu}_j \leq C_m^{\text{cpu}}$ and $\sum_{j\in A_m(t)} \text{mem}_j \leq C_m^{\text{mem}}$ for all t. Action masking enforces $\mathcal{F}(s_k)$.

Transition Kernel $\mathcal{P}$. Given s_k and $a_k = (i, m)$, the environment deterministically updates: (i) task i's $(\text{start}_i, c_i, \text{status}_i)$, (ii) VM m's allocation timeline, (iii) descendants' readiness when all parents are completed: task status$_j \leftarrow$ ready if $\forall p \in \text{Pa}(j)$: task status$_p$ = done and $\tau \geq \max_{p\in\text{Pa}(j)} c_p$, (iv) simulation clock to the next decision epoch τ_{k+1}.

Reward $\mathcal{R}$ with Concurrency-Aware Heuristics. To enable effective credit assignment, we define per-step rewards as regret reductions relative to integrated heuristic estimates that account for concurrent task execution on multi-core VMs.

Heuristic Estimates. At each state s, we compute two greedy estimates:

- **Makespan estimate** $\widehat{T}(s)$: Earliest completion time obtained by greedily scheduling remaining tasks using an earliest-completion-time (ECT) policy with full concurrency awareness.
- **Active energy estimate** $\widehat{E}(s)$: Minimum active energy consumption for remaining tasks, accounting for fractional CPU utilization and concurrent execution.

Both heuristics simulate a feasible completion of the workflow by:

1. Building per-VM event timelines from already-scheduled tasks, where each event $(t, \Delta_{\text{mem}}, \Delta_{\text{cores}})$ tracks resource changes at time t.
2. For each unscheduled task i, finding the earliest feasible start time $t \geq t^i_{\text{ready}}$ on each compatible VM $m \in \mathcal{C}_i$ when:

$$\text{used\_mem}_m(t) + \text{mem}_i \leq C_m^{\text{mem}} \quad \text{and} \quad \text{used\_cores}_m(t) + \text{cpu}_i \leq C_m^{\text{cpu}}$$

3. Selecting the VM that minimizes completion time (for makespan) or energy consumption (for energy).

For the energy heuristic, power on VM m at time t is modeled as:

$$P_m(t) = P_m^{\text{idle}} + (P_m^{\text{peak}} - P_m^{\text{idle}}) \cdot U_m(t), \quad U_m(t) = \min\left(1,\ \frac{1}{C_m^{\text{cpu}}} \sum_{j \in A_m(t)} \text{cpu}_j\right)$$

where $A_m(t)$ is the set of tasks active on VM m at time t, and $U_m(t) \in [0, 1]$ is the fractional CPU utilization. This fractional CPU model extends Chandrasiri & Meedeniya (2025) work by accounting for the aggregate core usage of all concurrent tasks, enabling accurate energy estimation when multiple tasks overlap on multi-core VMs. Energy is integrated piecewise-constant over segments bounded by task start/completion events.

Regret-Based Reward. At each decision epoch k, after action a_k transitions $s_k \to s_{k+1}$, we compute normalized regret reductions:

$$\Delta R_k^{\text{mk}} = -\frac{\widehat{T}(s_{k+1}) - \widehat{T}(s_k)}{\max(\widehat{T}(s_{k+1}), \varepsilon)}, \qquad \Delta R_k^{\text{en}} = -\frac{\widehat{E}(s_{k+1}) - \widehat{E}(s_k)}{\max(\widehat{E}(s_{k+1}), \varepsilon)}$$

where $\varepsilon > 0$ is a small constant. A positive value indicates the action reduced the estimated cost-to-go. The combined reward is:

$$r_k = w_T \cdot \Delta R_k^{\text{mk}} + w_E \cdot \Delta R_k^{\text{en}}$$

where (w_T, w_E) are tunable weights that scalarize the multi-objective problem.

Objective. We optimize a stationary policy $\pi_\theta(a \mid s)$ to maximize expected return

$$J(\pi_\theta) = \mathbb{E}_{\pi_\theta, \mathcal{P}}\left[\sum_{k=0}^{K} \gamma^k\, r_k\right],$$

typically with $\gamma \approx 1$ for episodic scheduling. At episode termination, final makespan $T_{\text{mk}} = \max_i c_i$ and total energy

$$E_{\text{tot}} = \sum_{m \in \mathcal{M}} \int_0^{T_{\text{mk}}} P_m(t)\, dt$$

are computed for evaluation.

Constraints and Termination. Feasibility is enforced by $\mathcal{F}(s_k)$: (i) precedence constraints via readiness, (ii) per-VM resource capacities (CPU cores and memory) over time with concurrent task support, (iii) task–VM compatibility. The episode terminates when all tasks are completed.

2.2 Workflow Structure and Host Regime Decomposition

To understand the problem complexity, we first examine what makes one job different from another at a structural level. Since jobs consist of multiple interdependent tasks, their *dependency structure* determines how many tasks can be ready in parallel and how scheduling decisions propagate through time.

Let $G = (V, E)$ be a DAG with task work $\{L_i\}_{i \in V}$. We denote:

- total work $W = \sum_{i \in V} L_i$,
- critical-path length $L_{\text{CP}} = \max_{\pi \in \text{paths}(G)} \sum_{i \in \pi} L_i$,
- depth D and level widths $|\text{level}(\ell)|$ from a standard levelization of the DAG.

A useful scalar summary of intrinsic parallelism is

$$\Phi \;=\; \frac{W}{L_{\text{CP}}}.$$

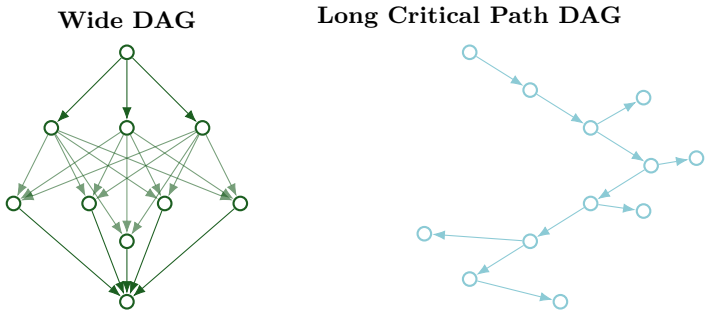

Figure 2: Schematic comparison of a **Wide** DAG (left, shallow with many parallel branches) and a **Long-CP** DAG (right, deep dependency chain with limited side-branch concurrency).

Intuitively, L_{CP} is the amount of work that *must* be done sequentially, while W is the total work. Large Φ indicates ample exploitable parallelism; Φ close to 1 indicates a nearly sequential job.

We focus on two representative structures:

- **Long Critical Path (LongCP).** Deep dependency chains (D large) with small width, so Φ is close to 1–few. Ready sets are typically small and concentrated near occasional side branches. In a queue-free, homogeneous-speed setting, the makespan lower bound $T^\star = L_{\mathrm{CP}}/s$ is dominated by the chain; scheduling primarily ensures no critical-path idling while fitting short side branches.
- **Wide DAG.** Shallow depth (D small) with large level widths, so $\Phi \gg 1$. Ready sets are large and bursty at wide layers, and many task–VM assignments are simultaneously feasible. Scheduling is dominated by packing across VMs (load balance and co-location), with small placement choices causing large utilization changes in a queue-free setting.

Figure 3 shows how workflow structure changes the scheduling problem. Wide DAGs create a rugged landscape with shallow valleys everywhere. When many tasks are ready at once, there are countless ways to assign them to VMs. A small change in the assignment can cause energy consumption to jump around unpredictably.

LongCP DAGs produce a different landscape entirely. Fewer valleys, but deeper ones. The critical path constrains most decisions because dependency chains force a specific order. There's little room to explore alternatives. The main freedom is placing short side branches, which creates a few isolated basins instead of a chaotic surface.

With these distinctions, we address Q1 and Q2 through four host-configuration regimes (AL, NA, HS, HP) in our queue-free setting.

Queue-Free Regime

Across all four regimes we assume a *queue-free*, single-workflow environment. A cluster of V virtual machines (VMs) executes a single DAG workflow. Time is continuous and tasks are non-migratable once assigned to a VM.

The dataset generator enforces queue-freedom: for each DAG we scale task memory and CPU requirements so that the peak-width layer fits within aggregate cluster capacity. If $\mathcal{L}_{\max}$ is the peak layer,

$$\sum_{i\in\mathcal{L}_{\max}} \mathrm{req\_mem}_i \;\le\; \sum_{v=1}^{V} \mathrm{mem}_v, \qquad \sum_{i\in\mathcal{L}_{\max}} \mathrm{req\_cores}_i \;\le\; \sum_{v=1}^{V} \mathrm{cores}_v.$$

Thus, whenever a task becomes ready, there exists a feasible placement that does not violate capacity. The four regimes below differ only in how VM *speed* and *power* are parameterized.

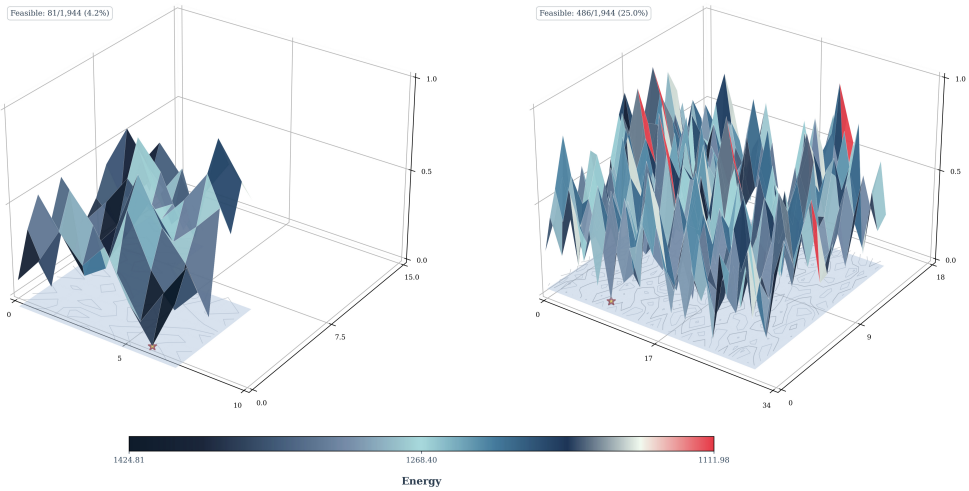

Figure 3: **Fitness landscape of the scheduling search space.** We exhaustively enumerate all feasible action sequences for a small scenario and project them onto a 2D plane using a Hilbert curve. The Z-axis shows energy consumption (lower is better). Left: LongCP DAG. Right: Wide DAG .

Queue-Free, Homogeneous-Speed Regime

In the homogeneous-speed regime, all VMs process work at the same rate s (e.g., MIPS). If a task i has computational length L_i (in MI), its processing time is $\tau_i = L_i/s$, independent of which VM executes it.

Because speeds are identical and the instance is queue-free, any non-pathological schedule that keeps critical-path tasks busy attains the same makespan $T^\star = L_{\mathrm{CP}}/s$, where L_{CP} is the critical-path length of the DAG. Makespan is determined by DAG structure, not by VM assignment choices.

We focus on power-heterogeneous but speed-homogeneous hosts. At approximately fixed makespan, the only remaining degree of freedom is which VMs are active and for how long. This means the policy can only shape active energy by controlling VM power profiles.

Queue-Free, Homogeneous-Power Regime

In the homogeneous-power regime, all VMs share the same active power P^{act} but differ in speed $s(v)$. Active energy is approximated as $E \approx \int P^{\mathrm{act}}\, dt$. For a fixed task workload, this makes energy largely insensitive to speed (ignoring second-order utilization effects), while makespan still depends on $s(v)$ through $\tau = L/s(v)$.

The effective objective becomes primarily time-dominated: faster machines shorten makespan, but active energy remains approximately unchanged across VM choices since P^{act} is constant. This reduces the multi-objective problem to a single-objective problem focused on minimizing completion time.

Queue Free, Heterogeneous Aligned Regime

In the heterogeneous aligned regime, speed and energy efficiency move in the same direction. Faster machines are also more power efficient for the same amount of work. Formally, for two VMs v_1 and v_2 with speeds $s(v_1) < s(v_2)$, the faster VM has lower energy. Moving a task to a faster VM therefore tends to reduce both completion time and total energy, so the two objectives are mostly aligned.

Queue Free, Heterogeneous Non Aligned Regime

In the heterogeneous non aligned regime, we break this link between speed and efficiency. Some VMs are fast but energy hungry, while others are slow but energy cheap. In other words, higher speed does not imply lower energy per unit work, and in some cases it can be strictly worse.

This creates genuine conflicts between local choices. A faster VM may reduce completion time but increase total energy, while a slower and more efficient VM may save energy at the cost of longer makespan. The trade off surface becomes non monotone, with several locally attractive choices depending on the exact speed and power pairing.

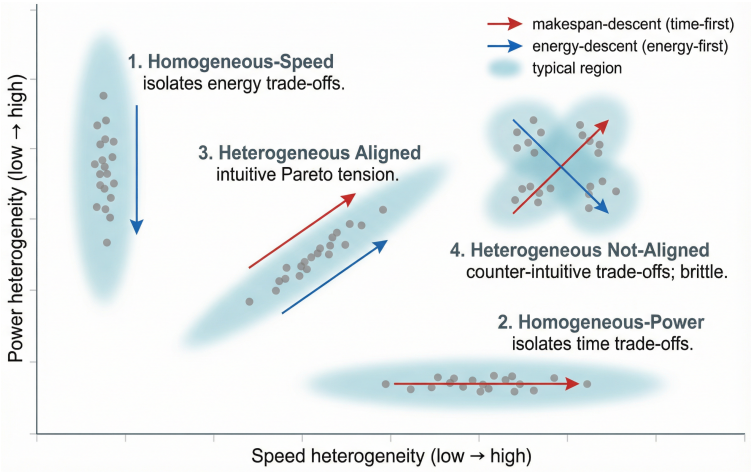

Figure 4: **Speed–power host regimes.** Illustration of the four queue-free host regimes studied in this paper. Homogeneous-Speed: all VMs share the same processing speed but differ in power, isolating energy trade-offs at fixed makespan. Homogeneous-Power: all VMs share the same active power but differ in speed, isolating time trade-offs. Heterogeneous Aligned: speed and power are monotonically aligned, inducing an intuitive Pareto frontier between makespan and active energy. Heterogeneous Non-Aligned: speed and power are not aligned, creating counter-intuitive trade-offs and stronger generalization challenges for learned schedulers.

3 Model Architecture

Our scheduler is a deep actor–critic architecture built around a shared graph neural network (GNN) backbone that embeds both workflow tasks and virtual machines (VMs). Figure 5 summarizes the deep reinforcement learning architecture used throughout this work. We follow the standard actor–critic decomposition : an actor (policy network) outputs a stochastic policy $\pi_\theta(a \mid s)$ over scheduling actions, while a critic (value network) estimates the state value $V_\phi(s)$ and is used as a learned baseline to reduce the variance of the policy-gradient estimator (Konda & Tsitsiklis (1999); Mnih et al. (2016); Schulman et al. (2017)). Both heads share the same GIN backbone so that policy and value are learned from a common structural representation of the workflow and VM pool.

Input representation. At each decision step the environment provides a structured observation encoding the current workflow DAG and the VM pool. Tasks are represented by: (i) scheduled/ready flags, (ii) remaining work and completion time, (iii) CPU and memory requirements. VMs are represented by: (i) completion times and current utilization, (ii) speed, core count and available cores, (iii) memory capacity and free memory, (iv) host idle and peak power. Compatibility edges link tasks to VMs on which they can run, and dependency edges encode the workflow DAG (parent–child relations). The resulting structure forms a bipartite-plus-dependency graph with heterogeneous node types and two distinct edge families.

Task and VM encoders. We map raw task and VM features into a common latent space using two separate multi-layer perceptrons (MLPs). The task encoder consumes a 6-dimensional feature vector and outputs a d-dimensional embedding. The VM encoder consumes a 12-dimensional feature vector and outputs an embedding in the same space. Both encoders use batch normalization and ReLU nonlinearities, followed by a final linear projection. The encoders are shared between actor and critic.

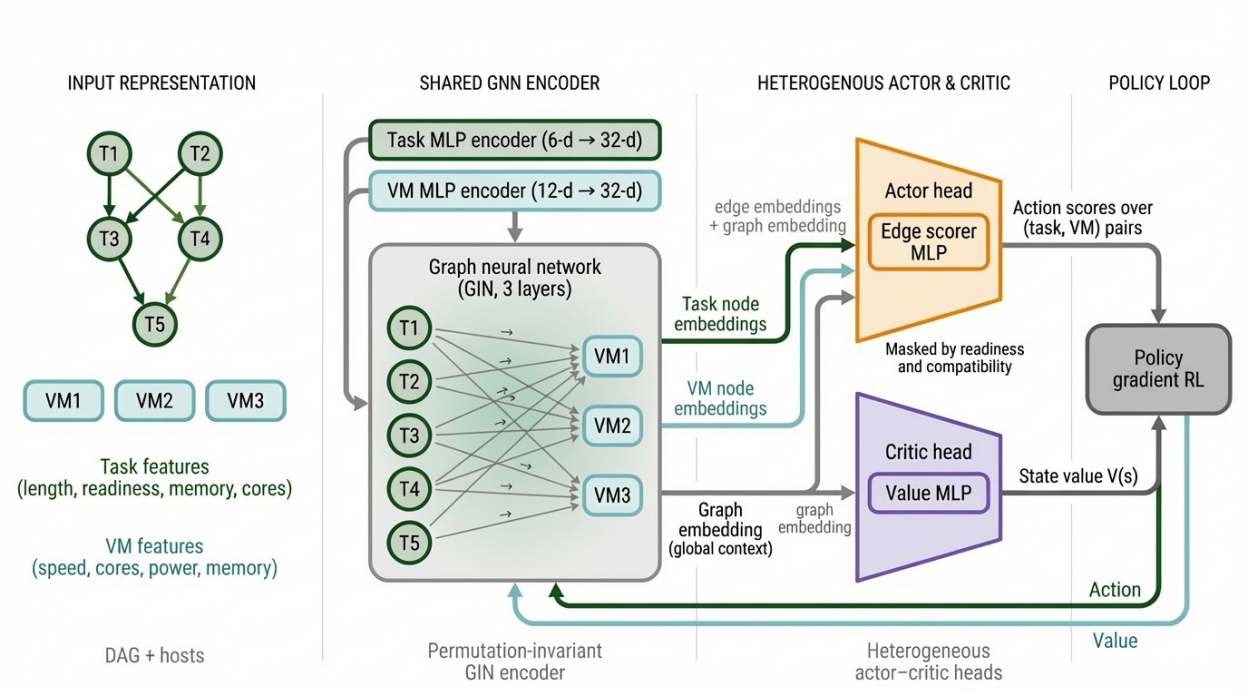

Figure 5: Overview of the proposed GIN-based actor–critic scheduler architecture.

GIN backbone. We concatenate the encoded task and VM nodes into a single set and process them using a three-layer Graph Isomorphism Network (GIN) (Xu et al. (2019)) with hidden dimension h. The edges capture both task–VM compatibility links and task–task dependency links. Each GIN layer performs neighborhood aggregation followed by an MLP update, producing type-agnostic node embeddings that capture both the workflow structure and the VM context. A global mean-pooling operation over all nodes is then applied to compute a *graph embedding*, providing a compact summary of the current scheduling state.

Actor head. The actor network uses the learned node and graph embeddings to evaluate possible scheduling decisions. For each task that can run on a given VM, we build an *edge embedding* by combining the embeddings of the task node, the VM node, and the overall graph representation. This combined vector is passed through a small multilayer perceptron (the *edge scorer*) that outputs a single numerical score. All scores are then arranged into a task×VM matrix, where invalid entries that corresponds to tasks that are not ready, already scheduled, or incompatible are masked out. After flattening the remaining entries, a softmax function converts the scores into a probability distribution over all valid (task, VM) pairs. The agent's policy $\pi_\theta(a \mid s)$ is defined by sampling or selecting the highest-scoring action from this distribution.

Critic head. The critic shares the same GIN backbone and encoders but operates only on the global graph embedding. A two-layer MLP maps the graph embedding to a scalar value estimate $V_\phi(s)$, representing the expected return from the current state under the policy.

Training. Actor and critic parameters are jointly optimized with a policy-gradient algorithm using the mixed objective described in Section 2.1. The actor receives policy-gradient updates based on advantage estimates that combine makespan and active-energy, while the critic is trained with a regression loss toward bootstrapped returns. Gradients flow through the encoders and GIN backbone, so the resulting node, edge, and graph embeddings become specialized for structure-aware scheduling decisions across the host regimes described in the next section.

4 Experimental Methodology, Results and Analysis

This section shows the experiments and results that really answer Q1 (how DAG structure and host setups affect scheduling choices) and Q2 (how well policies work across different structures). It also touches on Q3 (when to use simple rules versus learned ones) in a low-key way at the end.

4.1 Experimental Methodology

We conduct experiments using the two workflow families (Wide and LongCP) and four host regimes (HS, HP, AL, NA) defined in Section 2.2. For each topology class, we define disjoint training and evaluation seed sets to generate independent workflow instances.

4.1.1 Training Setup

We train the GNN-based actor–critic agent described in Section 3 using standard on-policy policy gradients with advantage estimation.

Episodes and environment dynamics. Each episode executes a complete workflow from source tasks to sink completion. The environment implements the transition kernel $\mathcal{P}$ from Section 2.1, simulating task execution, updating readiness based on DAG dependencies, and integrating power over time.

Reward formulation. We use the mixed objective from Section 2.1:

$$r_k = w_T \, \Delta r_k^{\mathrm{mk}} + w_E \, \Delta r_k^{\mathrm{en}},$$

with per-decision incremental improvements in makespan and energy. Coefficients (w_T, w_E) are fixed as (1,1) within experiments.

Training distributions. For each host regime, we train three specialist policies:

1. **Wide-only:** episodes contain only Wide workflows.
2. **LongCP-only:** episodes contain only LongCP workflows.

Training hyperparameters. All agents use the same PPO configuration: 2,000,000 total timesteps with 10 parallel environments, batch size 2560 (256 steps per environment), 4 minibatches, 4 update epochs per batch, learning rate 2.5×10^{-4}, GAE with $\gamma = 0.99$ and $\lambda = 0.95$, and clip coefficient 0.2.

We provide the training script and configurations to reproduce all results at `https://github.com/DAGAwareRL?tab=repositories`.

4.1.2 Evaluation Protocol

In-distribution and cross-structure evaluations. For each trained agent, we evaluate under all combinations of:

- training topology: Wide-only vs. LongCP-only.,
- test topology: Wide vs. LongCP,
- host regime: HS, HP, AL, NA.

This produces in-distribution conditions (e.g., Wide-trained agent on Wide workflows) as well as cross-structure conditions (e.g., Wide-trained agent on LongCP workflows), under each host regime.

Metrics. For every configuration we report:

- average makespan per workflow.
- average active energy per workflow;
- the empirical Pareto relationship between energy and makespan across different agents and baselines.

4.2 Results and Analysis

4.2.1 Results

Table 1 summarizes cross-domain performance of the wide and LongCP heterogeneous specialists across the NAL, HS, and HP host configurations.

Homogeneous-Speed (HS). In the homogeneous-speed regime, all VMs have the same processing rate. This means makespan depends almost entirely on DAG structure and keeping the critical path busy. The main remaining choice is which power profile to use when: the scheduler can reduce active energy by concentrating work on lower-power VMs. Table 1 shows that both specialists achieve identical makespan on their respective evaluation domains (1.51 on LongCP, 0.44 on wide), but the wide specialist achieves substantially lower active energy (32.22 vs 37.23 on LongCP, 50.66 vs 57.24 on wide). This means the wide specialist wins on energy while maintaining the same makespan, even when tested on LongCP configurations where it was never trained. The reason is that training on wide parallel structures teaches the agent to spread work efficiently across VMs with different power profiles. When speed is fixed, this power-aware load balancing becomes the primary optimization lever, and the wide specialist has learned exactly this skill. The LongCP specialist, trained on sequential bottlenecks, focuses more on keeping the critical path busy and pays less attention to power selection. But here's the thing: a simple power-aware heuristic (which always prefers less power-hungry VMs when feasible) consistently achieves lower or comparable active energy at similar makespans, while being much cheaper to compute. In HS, this low-complexity heuristic should be preferred in practice.

Homogeneous-Power (HP). In the homogeneous-power regime, all VMs share the same active power and differ only in speed. This makes the problem simpler: energy becomes primarily time-dominated, with only weak sensitivity to per-VM choices beyond their impact on completion time. Table 1 shows that the LongCP specialist has a consistent advantage under HP. On both evaluation domains, it attains lower makespan (2.62 vs 2.67 on LongCP, 0.85 vs 0.87 on wide) and lower active energy (16.74 vs 16.74 on LongCP, 25.53 vs 25.56 on wide). This happens because training on long dependency chains teaches the agent to prioritize critical path optimization and assign important tasks to faster VMs. When power is fixed, this speed-aware scheduling becomes the dominant factor for both makespan and energy. The wide specialist, trained on more parallel structures, learns a more exploratory policy that spreads work broadly but misses the structured prioritization that matters when speed varies. But the same conclusion holds here: a simple time-aware heuristic (which prioritizes faster VMs to minimize makespan) should be sufficient and is a better practical choice than either learned agent.

Heterogeneous Aligned (AL). Table 1 shows that the wide specialist beats the LongCP specialist on both objectives in both evaluation domains for the AL host configuration. On LongCP tasks (where it never trained), the wide specialist gets 22% lower makespan (2.24 vs 2.88) and 26% lower energy (28.31 vs 38.28). On wide tasks (its training domain), it stays ahead with 17% lower makespan (0.73 vs 0.88) and 28% lower energy (43.39 vs 60.67). This raises a simple question: why does the wide specialist transfer to LongCP structures, but not the other way around?

As shown later in Figure 8, the LongCP specialist exhibits near-perfect makespan–energy correlation under AL, while the wide specialist shows a weaker correlation. In LongCP topologies, a dominant critical path ties the two objectives together and the multi-objective problem behaves almost like a single objective. This narrows the training signal and leads to a simple policy that breaks when the critical-path assumption no longer holds.

Table 1: Cross-domain evaluation of heterogeneous agents across host configurations. Best results for each evaluation domain within a host configuration are highlighted in bold.

Host cfg	Method	Eval Domain	Makespan	Active Energy (10^7 J)
HS host configuration				
HS	long_cp	long_cp	**1.51**	37.23
HS	wide	long_cp	**1.51**	**32.22**
HS	long_cp	wide	**0.44**	57.24
HS	wide	wide	**0.44**	**50.66**
HP host configuration				
HP	long_cp	long_cp	**2.62**	**16.74**
HP	wide	long_cp	2.67	16.74
HP	long_cp	wide	**0.85**	**25.53**
HP	wide	wide	0.87	25.56
AL host configuration				
AL	long_cp	long_cp	2.88	38.28
AL	wide	long_cp	**2.24**	**28.31**
AL	long_cp	wide	0.88	60.67
AL	wide	wide	**0.73**	**43.39**
NA host configuration				
NA	long_cp	long_cp	3.14	**24.39**
NA	wide	long_cp	**2.40**	37.18
NA	long_cp	wide	0.93	**42.59**
NA	wide	wide	**0.76**	56.43

Wide topologies behave differently. They generate states where many tasks can be scheduled at the same time and no single path dominates. In these states, the agent has to learn when to spread work across machines to reduce makespan and when to pack work to save energy. The weaker correlation between objectives means the agent is forced to learn real trade offs between them, rather than a hidden single objective rule.

The state space picture in Figure 6 adds an important piece. The t-SNE shows that LongCP states sit mostly inside the cloud of wide states. This means the two domains share much of the same observation space, but they explore it in different ways. The wide agent sometimes sees long chain structures as rare subgraphs during training, so these states are still in distribution at test time. The LongCP agent almost never sees highly parallel states, so wide structures are effectively out of distribution for it. This asymmetry in state coverage, combined with the richer training signal in the wide domain, explains why the wide specialist transfers well while the LongCP specialist overfits to its narrow training regime.

Heterogeneous Non-Aligned (NA). In the non-aligned configuration, speed and power are negatively correlated: faster VMs consume more power. This creates a genuine trade-off between makespan and energy. Table 1 shows that neither agent uniformly dominates. On the LongCP domain, the wide specialist achieves lower makespan (2.40 vs 3.14), but the LongCP specialist is more energy-efficient (24.39 vs 37.18). On the wide domain, the wide specialist is again faster (0.76 vs 0.93), while the LongCP specialist achieves lower active energy (42.59 vs 56.43).

Figure 7 confirms the headline trade-offs: the wide specialist tends to dominate the low-makespan region, while the LongCP specialist tends to dominate the low-energy region. This mirrors the averages and shows the biases across the whole distribution.

The diagnostic referenced earlier now appears in Figure 8. Panel (a) (AL, Long-CP) shows near-perfect coupling (r=0.993): makespan and active energy move almost linearly together. Panel (b) (AL, Wide) remains strongly coupled (r=0.953), though with a slightly wider band. Now , under NA, the link weakens:

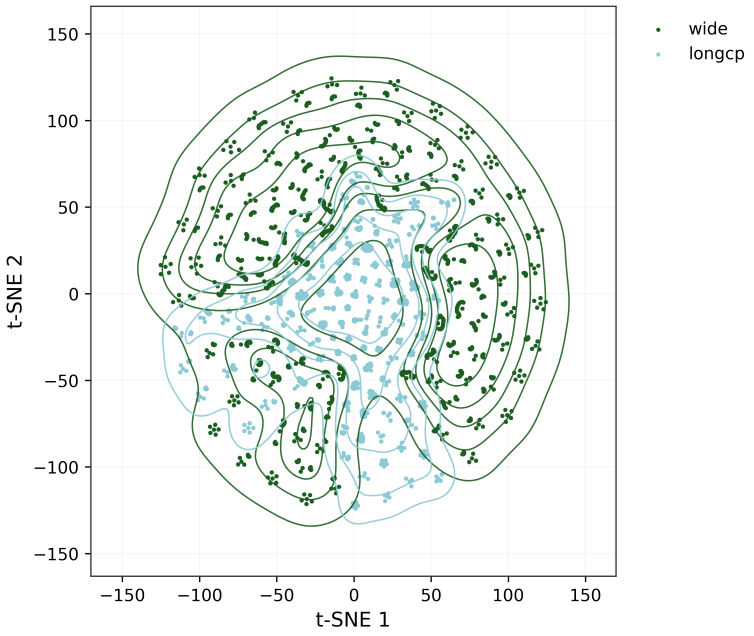

Figure 6: State space coverage under random agents on wide (green) and LongCP (teal) topologies. Points are 2D t-SNE embeddings of collected observations with thin outlines showing local density.

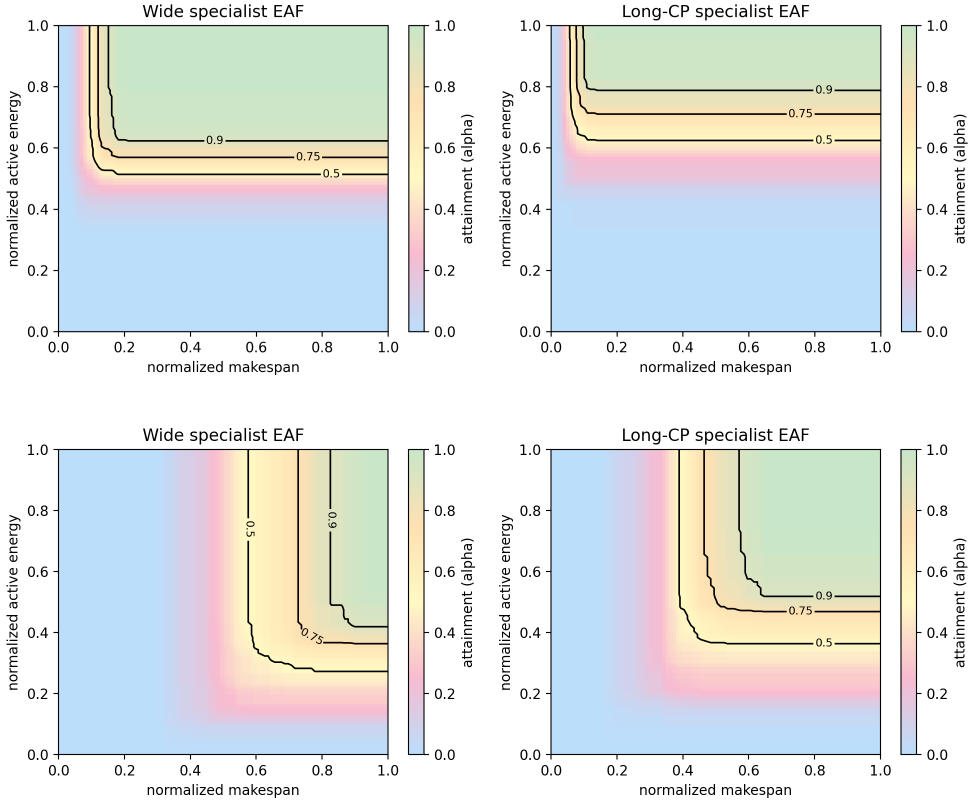

Figure 7: Empirical attainment functions (EAFs) over 100 test jobs for the wide and LongCP specialists on the wide configuration (top) and the LongCP configuration (bottom). Each panel shows the distribution of the trade-off between normalized makespan (x-axis) and normalized active energy (y-axis), with color indicating the attainment level α and black contours marking $\alpha \in \{0.5, 0.75, 0.9\}$.

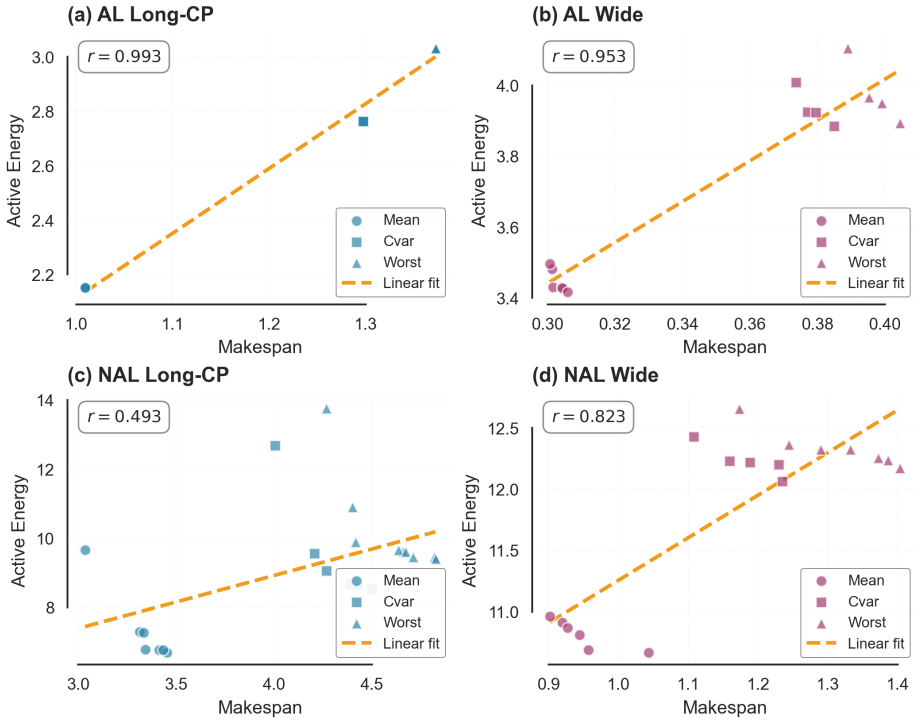

Figure 8: Correlation between makespan and active energy across Pareto checkpoints for AL (top row) and NA (bottom row), with LongCP and wide specialists.

panel (c) (NA, Long-CP) shows only moderate correlation with substantial spread (r=0.493), while panel (d) (NA, Wide) is stronger but still looser than AL (r=0.823) and exhibits more variance at lower makespans.

AL creates a strong link between time and energy, so improving one usually improves the other. NA loosens this connection, especially for Long CP workflows, allowing real trade offs between time and energy. This pattern also shows up in the EAFs: under NA, different specialists dominate different parts of the frontier, rather than a single agent leading everywhere

These patterns raise a natural question: why do two agents trained with the same mixed objective end up with such different preferences over time and energy? The answer lies in how DAG structure shapes the states the agent visits during training.

Let π be a policy over observations s (task requirements and readiness, VM utilization, energy rate features), and let $d_\pi(s)$ denote its state occupancy measure. We can rule out one simple explanation: maybe the specialists just encounter unfamiliar states when tested on the other topology. Figure 6 shows that LongCP states fall largely within the outer envelope of wide states. So the performance gaps are not because agents see completely out-of-distribution observations.

Given that the raw state distributions overlap, the question becomes: how do trained policies actually use this shared state space? Figure 10 explains the answer. The x-axis shows a parallelism index (normalized ready tasks per level / DAG width) and the y-axis shows an energy-intensity index (aggregate active power rate while busy).

Actually, that both agents are trained with the same mixed objective we have described earlier in Section 2.1

$$J(\pi) = \alpha\,\mathbb{E}[E_{\text{active}}] + \beta\,\mathbb{E}[\text{Makespan}], \qquad \alpha, \beta > 0,$$

but they visit different parts of the state space because of their training DAGs.

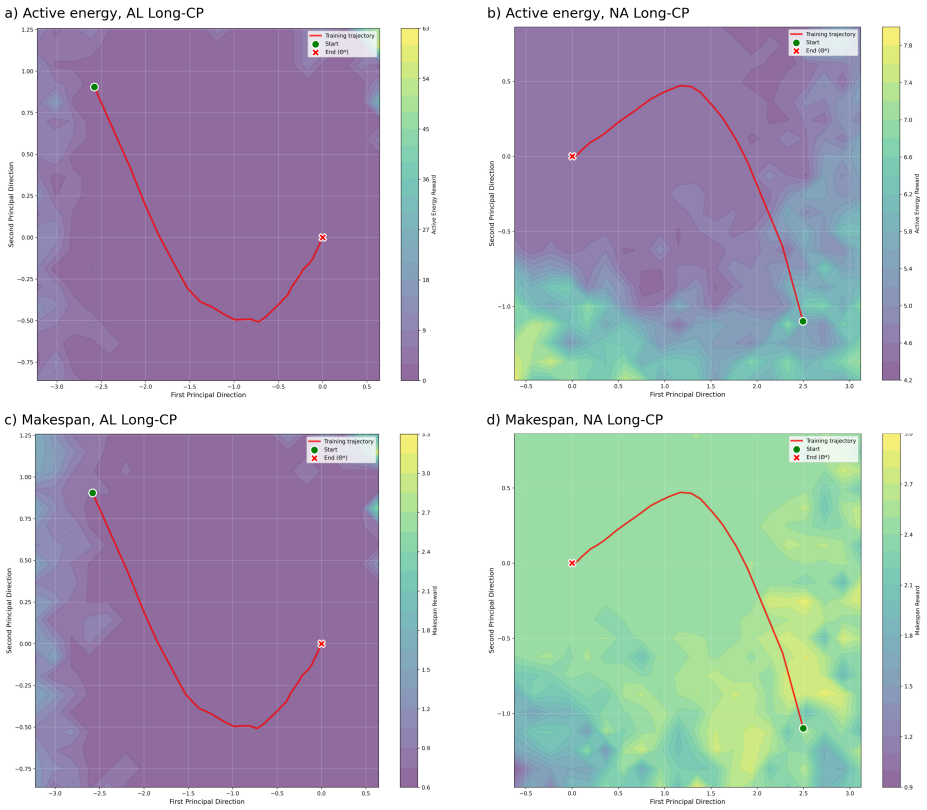

Figure 9: Panels: a) Active energy, case AL; b) Active energy, case NA; c) Makespan, case AL; d) Makespan, case NA.

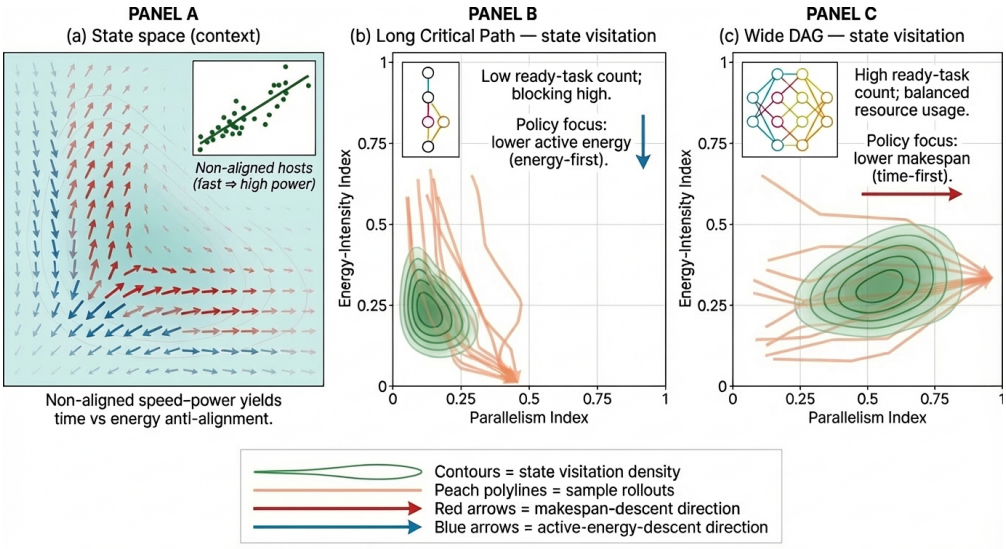

Figure 10: **State visitation under a fixed mixed objective and non-aligned speed–power.** (a) Conceptual state-space diagram with anti-aligned red (makespan) and blue (energy) descent directions, plus an inset showing non-aligned host speed–power. (b) Long Critical Path (LongCP) training: KDE contours and rollouts concentrate in low-parallelism regions; annotation highlights an energy-first policy bias. (c) Wide DAG training: visitation shifts toward higher-parallelism states; annotation highlights a time-first bias.

Here's the thing: both agents optimize the same $J(\pi)$. The difference is that DAG structure changes which states are reachable and frequently visited. Long critical paths (Panel 10b) compress the feasible manifold toward low parallelism, so d_π places more mass in low-parallelism regions where energy-reducing choices matter. Wide DAGs (Panel 10c) expand the feasible trajectory set $\mathcal{T}(\text{DAG}, \text{hosts})$ in parallel directions, so d_π shifts toward high-parallelism states where time-reducing choices are available. Structure rotates which parts of the trade-off surface are accessible during training.

Figure 9 provides additional evidence by showing how the learned value landscapes change with actor parameters during training. In the aligned case (AL), both makespan and energy gradients point in similar directions because faster VMs also consume less power. But in the non-aligned case (NA), the gradients point in opposite directions, creating a genuine trade-off. The LongCP specialist's landscape shows stronger gradients toward low active energy. This reflects the different biases learned from different $d_\pi(s)$ patterns.

This explains the cross-evaluation results in Table 1. When the LongCP specialist is tested on wide configurations, it still tries to minimize energy. When the wide specialist is tested on LongCP configurations, it still tries to minimize makespan because that's what it learned to prioritize when it had choices. Neither strategy is wrong, they just learned different priorities from different training distributions. The DAG structure during training fundamentally shapes which features the policy learns to care about, even when the objective function stays the same. Policy gradients are weighted by $d_\pi(s)$, so structure and host regime determine which features and transitions are emphasized during learning.

Regime-aware routing. The detailed cross-regime evaluation reveals what works in each extreme setting. We tested four corner cases : homogeneous speed (HS), homogeneous power (HP), aligned speed and power (AL), and anti-aligned speed and power (NA) and found that (i) simple heuristics can outperform learned policies when only one resource dimension varies, (ii) the LongCP specialist dominates when time and energy are strongly coupled, and (iii) each specialist learns a different inductive bias when the objectives conflict. Real cloud environments typically lie between these extremes, but understanding the boundary cases suggests a practical regime-aware routing strategy. We propose to first classify the current environment by computing variance of speeds, variance of powers, and their correlation, then route to the appropriate expert: in HS, a power-minimizing heuristic; in HP, a makespan-minimizing heuristic; in AL, the LongCP specialist. NA exhibits a different structure because the two specialists have complementary strengths along the Pareto frontier: the wide specialist achieves lower makespan, while the LongCP specialist achieves lower energy. We therefore suggest deploying both and selecting based on operational priority. Formally, given a user-specified or time-varying preference weight $\lambda \in [0, 1]$, the system would route to the wide specialist when $\lambda_{\text{time}} > 0.5$ and to the LongCP specialist otherwise. For mixed or intermediate regimes, a learned gating function $g : (\text{DAG features}, \text{host statistics}) \to \{\text{wide}, \text{LongCP}\}$ could predict which specialist maximizes expected utility, replacing hand-crafted rules with adaptive routing across the full configuration space.

5 Conclusion

The challenge of efficiently scheduling complex, dependency-driven workflows in modern cloud environments requires balancing completion time (makespan) against energy consumption. Deep Reinforcement Learning (deep RL) schedulers with Graph Neural Network (GNN) backbones are a promising alternative to classical heuristics, but their robustness and generalization across workflow structures and hardware regimes remain open questions. In this work, we addressed this by constructing a controlled benchmark that factorizes the problem into two contrasting DAG topologies : Wide Parallel and Long Critical Path (LongCP) , and four host regimes: Homogeneous Speed (HS), Homogeneous Power (HP), Aligned (AL), and Non-Aligned (NA). By training specialized agents and subjecting them to systematic cross-structure and cross-regime evaluation, we have successfully answered the three research questions posed in Section 1 and made several key contributions to the understanding of learned cloud scheduling:

Our results show, first, that DAG topology fundamentally shapes the policy's learned priorities, even under a fixed energy–makespan objective: agents trained on different topologies implicitly attend to different structural and resource features. Second, generalization failures are structured rather than random: performance drops concentrate in specific out-of-distribution combinations of topology and host regime, with the NA

configuration exposing the sharpest trade-off between speed and energy. Third, these findings highlight the brittleness of a single, monolithic deep RL scheduler: the learned policy behaves like a theory tuned to its training domain and can fail systematically when that domain shifts.

Motivated by this, we argued for a structure- and regime-aware routing approach: instead of enforcing a universally robust agent, first characterize the environment (in terms of DAG topology and host heterogeneity) and then route jobs to specialized policies or heuristics that are best suited to that region of the space. This embraces specialization rather than fighting it, and uses it to maintain high performance across diverse scenarios.

This research opens several avenues for future investigation. The most immediate is the formalization and implementation of the proposed routing mechanism. This would involve developing a robust, lightweight classifier capable of accurately identifying the structural domain of an incoming workflow and the characteristics of the target host environment, allowing for dynamic selection of the optimal specialist policy.

Furthermore, future work should explore domain-agnostic training techniques. This could involve investigating meta-learning or domain randomization approaches that explicitly force the deep RL agent to learn invariant features across different topologies and host regimes, potentially leading to a single, more robust policy that can adapt its behavior dynamically.

Finally, our study only looked at a single workflow with no queue. Real systems are more complex. They run many workflows at once and must deal with contention and priorities. Extending this analysis to a queue based setting is essential. In that case, structure and system load will interact in new ways. The results here show why structure matters, and they give a solid starting point for studying these more realistic scenarios.

References

Karan Ajmera and T. K. Tewari. A systematic literature review on contemporary and future trends in virtual machine scheduling techniques in cloud and multi-access computing. *Frontiers in Computer Science*, 6, 2024. doi: 10.3389/fcomp.2024.1288552.

Anton Beloglazov and Rajkumar Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation*, 2012.

Luiz Fernando Bittencourt and Edmundo Roberto Mauro Madeira. Hcoc: A cost optimization algorithm for workflow scheduling in hybrid clouds. *IEEE Transactions on Cloud Computing*, 6(3):649–662, 2018.

Rodrigo Calheiros et al. Cloudsim: A toolkit for modeling and simulation of cloud computing environments. *Software: Practice and Experience*, 2011.

Sunera Chandrasiri and Dulani Meedeniya. Energy-efficient dynamic workflow scheduling in cloud environments using deep learning. *Sensors*, 25(5):1428, 2025.

Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pp. 153–167, 2017.

Daniel G. da Silva and Luiz Fernando Bittencourt. Learning-based workflow scheduling in cloud computing: A survey. *Journal of Cloud Computing*, 6(1):1–20, 2017.

Ewa Deelman, Dennis Gannon, Matthew Shields, and Ian Taylor. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009.

Google. Machine learning applications for data center optimization. Technical report, Google, 2016. URL `https://deepmind.google/discover/blog/`

deepmind-ai-reduces-google-data-centre-cooling-bill-by-40/. DeepMind AI Reduces Google Data Centre Cooling Bill by 40%.

R. L. Graham et al. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 1979.

Zhaochen Gu, Sihai Tang, Beilei Jiang, Song Huang, Qiang Guan, and Song Fu. Characterizing job-task dependency in cloud workloads using graph learning. In *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 288–297, 2021. doi: 10.1109/IPDPSW52791.2021.00052.

Anas Hattay, Fred Ngole Mboula, Eric Gascard, and Zakaria Yahouni. Evaluating energy-aware cloud task scheduling techniques: A comprehensive dialectical approach. In *2024 IEEE/ACM 17th International Conference on Utility and Cloud Computing (UCC)*, pp. 109–118. IEEE, 2024.

Biao Hu, Xincheng Yang, and Mingguo Zhao. Online energy-efficient scheduling of dag tasks on heterogeneous embedded platforms. *Journal of Systems Architecture*, 140:102894, 2023.

International Energy Agency. Data centres and data transmission networks. Technical report, IEA, 2025. URL https://www.iea.org/energy-system/buildings/data-centres-and-data-transmission-networks.

Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. In *Advances in Neural Information Processing Systems*, volume 12, 1999.

Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pp. 50–56, 2016.

Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *ACM SIGCOMM*, pp. 270–288, 2019.

Mingwei Mao and Marty Humphrey. A survey of dynamic resource management in cloud computing. *IEEE Communications Surveys & Tutorials*, 14(4):1101–1117, 2012.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *ICML*, pp. 1928–1937, 2016.

Junyoung Park, Jaehyeong Chun, Sang Hun Kim, Youngkook Kim, and Jinkyoo Park. Learning to schedule job-shop problems: representation and policy learning using graph neural network and reinforcement learning. *International journal of production research*, 59(11):3360–3377, 2021.

Michael Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 2012.

Karl Popper. *The logic of scientific discovery*. Routledge, 2005.

Barry Rountree, David K. Lownenthal, Bronis R. de Supinski, Martin Schulz, Vincent W. Freeh, and Tyler Bletsch. Adagio: Making dvs practical for complex hpc applications. *ACM International Conference on Supercomputing*, pp. 460–469, 2009.

Rizos Sakellariou, Henan Zhao, and Ewa Deelman. Mapping workflows on grid resources: Experiments with the montage workflow. In *Grids, P2P and services computing*, pp. 119–132. Springer, 2010.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Mohammad Shahrad, Rodrigo Fonseca, Íñigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In *2020 USENIX Annual Technical Conference*, pp. 205–218, 2020.

Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Haluk Topcuoglu, Salim Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, 2002.

Laurens Versluis and Alexandru Iosup. A survey and annotated bibliography of workflow scheduling in computing infrastructures: Community, keyword, and article reviews. *arXiv preprint arXiv:2004.10077*, 2020.

Qitian Wu, Hengrui Zhang, Junchi Yan, and David Wipf. Handling distribution shifts on graphs: An invariance perspective. In *International Conference on Learning Representations (ICLR)*, 2022.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.

Kaixiang Zhang, Wenbo Li, Kai Zhang, Zenglin Li, and Zhaohui Qin. Deepjs: Job-shop scheduling with deep reinforcement learning. *IEEE Transactions on Industrial Informatics*, 17(3):2083–2093, 2021.

Zhiling Zong, Adam Manzanares, Xiaojun Ruan, and Xiao Qin. Ead and pebd: Two energy-aware duplication scheduling algorithms for parallel tasks on homogeneous clusters. In *IEEE Transactions on Computers*, volume 56, pp. 1661–1675, 2007.