From Each Job According to Its Structure: DAG Topology and the Generalization of Energy-Aware Cloud Schedulers

Anonymous authors
Paper under double-blind review

Abstract

Cloud providers must allocate heterogeneous computing resources to many users workflows demands while balancing speed, cost, and energy use. Each workflow can be represented as a directed acyclic graph, that is, a dependency graph of tasks. We study a single workflow, queue-free setting with a graph-based learning scheduler that jointly optimizes completion time and energy consumption. To isolate the effect of structure, we build a controllable benchmark that systematically varies graph width, depth, and critical-path length (for example, "wide" versus "long critical path"). We also sweep multiple host configurations, including non-aligned speed–power regimes that create strong trade-offs between time and energy.

Our evaluation spans hundreds of workflows per domain and measures generalization across both changes in graph topology and changes in host configuration. We show empirically that graph topology induces distinct state distributions and action choices for the scheduler. Policies trained in one regime can degrade under cross-topology and cross-host shifts, and non-aligned hosts amplify tensions between completion time and energy. These findings highlight that what the scheduler observes and what actions are useful are largely dictated by task-graph structure and machine heterogeneity.

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 the International Energy Agency (2025) report, the energy used by data centers keeps growing because of heavier AI training and inference workloads. And these jobs do not look like simple batch scripts. They come with huge variations in parallel work and deep dependency chains. This means even small improvements in how cloud resources are allocated can save a noticeable amount of time and electricity. Figure 1 shows the basic problem. A workflow has many tasks that depend on each other. It is usually written as a directed acyclic graph. Some stages have a lot of parallel tasks. Others form long chains. At the same time, cloud machines are all different. Some run fast but draw more power. Others run slow but use less. The scheduler has to match each ready task to a machine. Every match changes the total time the job will take and how much power the system will burn. There is a lot of older work on workflow scheduling. Many systems use heuristics such as HEFT and CPOP to pick a machine for each task Topcuoglu et al. (2002a). These rules often work well when the workflow shape and machine speeds are known ahead of time. But modern AI jobs break those assumptions. They have a mix of very wide layers and very long chains. And cloud providers now offer many VM types with very different speed and power numbers Beloglazov & Buyya (2012b). So the same heuristic can work on one workload and fall apart on another. Because of this, some groups have turned to learning based schedulers. They use deep reinforcement learning or graph neural networks to learn a policy from past runs Mao et al. (2019a). These

schedulers can do better than simple rules in fixed settings. While promising, learned schedulers raise a crucial question: how does the structure of the workflow and the heterogeneity of the hosts shape what the policy learns and how robustly those learned behaviors transfer across different scenarios? That is the gap this paper tries to study.

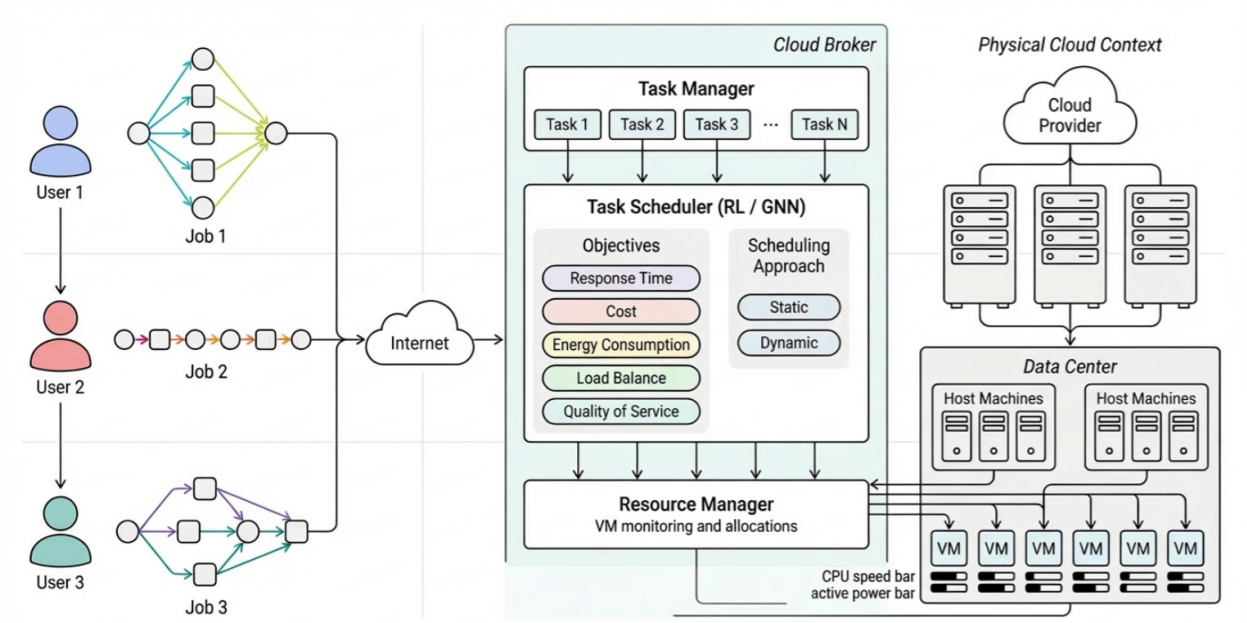


Figure 1: Overview of the workflow scheduling problem in a heterogeneous cloud. The scheduler receives a DAG with ready tasks and a set of machines with different speed and power. It must decide which task runs on which machine, and each choice changes both completion time and energy.

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. (2002b) 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 core 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 (2012a) 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 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 DRL agents could learn to pack tasks and allocate resources in cluster settings, outperforming hand tuned heuristics on job completion time. The core 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 DRL schedulers can match or beat classical heuristics in controlled environments (Zhang et al. (2021), Mao et al. (2016)).

But early DRL 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., 2019b). 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. This confirms that exploiting graph structure helps the policy generalize within the training distribution.

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. The first step is to identify what the shift looks like in our setting. Define and measure how deployment DAGs differ from training, then decide how to handle 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. (2019b), 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. 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 scheduler on these environments. We use separate agents specialized to Wide workflows and to LongCP workflows, plus one mixed topology agent trained on both. 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. Prior work has either used topology as a constraint or as a signal for heuristic scheduling, or has embedded DAGs to improve performance in fixed environments. In contrast, we focus on how changes in topology and host regime alter what the policy has learned and where its generalization breaks down.

Research Questions This setup lets us ask four main questions:

- **Q1:** How does DAG structure (wide parallel versus long critical path) affect what a learned policy prioritizes when trained with a mixed energy and makespan objective? Does structure create implicit biases even when the objective stays the same?
- **Q2:** How do different speed and power setups change the scheduling problem? When one dimension is homogeneous (HS or HP), when speed and power are aligned (AL), or when they conflict (NA), what strategies emerge?
- **Q3:** How well do policies trained on one DAG structure generalize to another? When does a Wide specialist beat a LongCP specialist and the other way around, and what explains these gaps?
- **Q4:** Given what we observe about generalization, what practical approach can handle diverse regimes? When are simple heuristics enough, when do learned specialists help, and how should we route between them based on what we see in the environment?

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 RL schedulers specialized to Wide workflows, to LongCP workflows, and to a mixed topology distribution. 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 2.1), 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. Unlike 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) 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.

This reward formulation preserves credit assignment across scheduling decisions while accounting for the benefits of concurrent task execution. When an action schedules a task that can run concurrently with already-scheduled tasks (exploiting available VM cores and memory), both $\widehat{T}(s)$ and $\widehat{E}(s)$ typically decrease, yielding positive reward signals that guide the agent toward efficient resource utilization.

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 $W = \sum_{i \in V} L_i$,

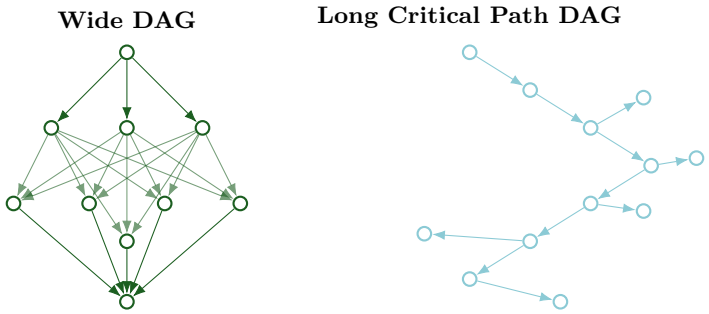


Figure 2: Schematic comparison of a **Wide** DAG (left, shallow with many parallel branches) and a **Long-CP** DAG (right, deep dependency chain with limited side-branch concurrency).

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

A useful scalar summary of intrinsic parallelism is

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

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 (Long-CP).** 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 assignment order can cause energy consumption to jump around unpredictably. The problem compounds in parallel layers where load imbalance across VMs adds up quickly.

Long-CP 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 this distinction in mind, we can better address both questions Q1 and Q2. Figure **??** summarizes how workflow structure and execution patterns differ between "wide" and "long critical-path" DAGs; in the following we place these behaviors in the context of four host regimes.

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.

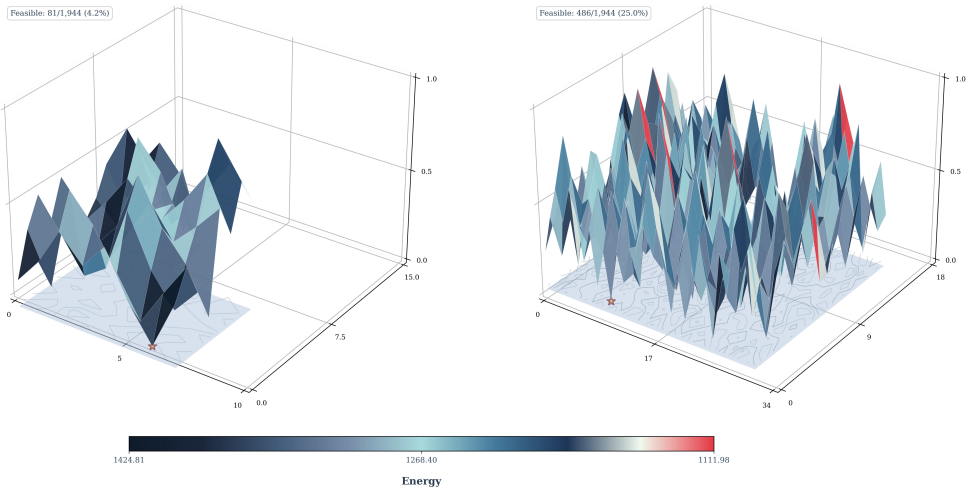


Figure 3: **Fitness landscape of the scheduling search space.** We exhaustively enumerate all feasible action sequences for a small scenario and project them onto a 2D plane using a Hilbert curve. The Z-axis shows energy consumption (lower is better). Left: Long-CP DAG. Right: Wide DAG .

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,

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

Thus, whenever a task becomes ready, there exists a feasible placement that does not violate capacity. Any waiting time is therefore induced by the policy, not by hard resource constraints. The four regimes below differ only in how VM *speed* and *power* are parameterized.

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 $T^\star = L_{\text{CP}}/s$, 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. Under the division_off energy model, active energy is computed by integrating instantaneous power over time: $E \approx \int P_v(u(t))\, dt$. 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.

Research question: Does training on wide DAGs (which expose larger ready sets and more VM assignment choices) produce policies that generalize differently than training on long-CP DAGs (which have limited concurrency and fewer choices)? Section 4.2 tests this.

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 $E \approx \int P^{\text{act}}\, dt$. 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.

Research question: When the problem simplifies to time minimization, does training on long-CP DAGs (which emphasize critical path optimization) produce policies that behave differently than training on wide DAGs (which emphasize parallel resource allocation)? Section 4.2 examines this.

Queue-Free, Heterogeneous Aligned Regime

The heterogeneous aligned regime introduces both speed and power heterogeneity with a monotone relationship: faster machines consume more power, i.e., $s(v_1) < s(v_2) \Rightarrow P^{\text{act}}(v_1) < P^{\text{act}}(v_2)$. Under division_off, the active energy increment for executing work on VM v scales with its power: $E^{\text{act}}(j, v) \approx \int P^{\text{act}}(v)\, dt$. Moving a task to a faster and higher-power VM decreases completion time but increases energy due to the larger $P^{\text{act}}(v)$.

This creates a natural coupling between objectives: decisions that reduce makespan tend to increase energy, and vice versa. However, the coupling is monotone, so the trade-off surface is relatively smooth.

Research question: When objectives are coupled but monotonically aligned, does DAG structure during training (wide vs. long-CP) affect which part of the Pareto frontier the policy learns to target? Section 4.2 investigates this.

Queue-Free, Heterogeneous Non-Aligned Regime

In the heterogeneous non-aligned regime, we break the monotone speed–power relationship. Some VMs are fast and energy-cheap, others slow and power-hungry. With division_off, energy depends primarily on P^{act} and the time integral, not on $1/s(v)$.

This creates conflicting local preferences: faster VMs may reduce time but not necessarily energy, and higher power increases energy even if time improves. The trade-off surface becomes non-monotone, with multiple local optima depending on the specific speed–power pairings available.

Research question: When speed and power are non-monotonically related, does the state distribution induced by DAG structure (wide vs. long-CP) create different inductive biases during training? Specifically, do policies trained on different topologies learn to prioritize different objectives even when trained with the same reward function? Section 4.2 explore this through empirical evaluation and state-space analysis.

3 Model: GIN Scheduler

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.

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.

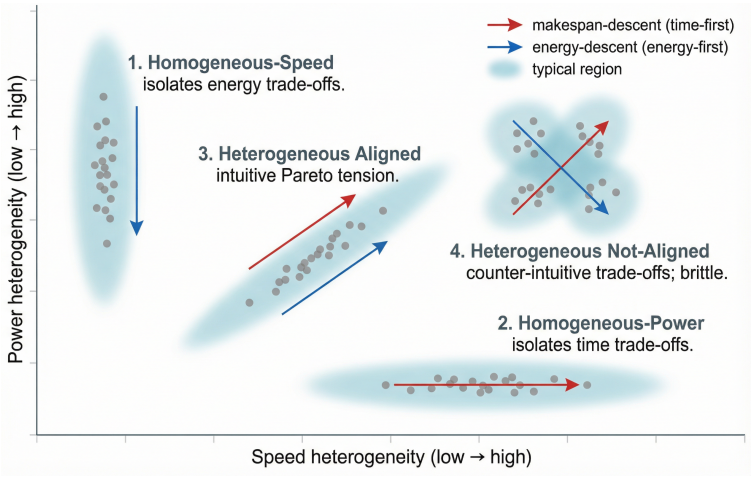


Figure 4: **Speed–power host regimes.** Illustration of the four queue-free host regimes studied in this paper. Homogeneous-Speed: all VMs share the same processing speed but differ in power, isolating energy trade-offs at fixed makespan. Homogeneous-Power: all VMs share the same active power but differ 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.

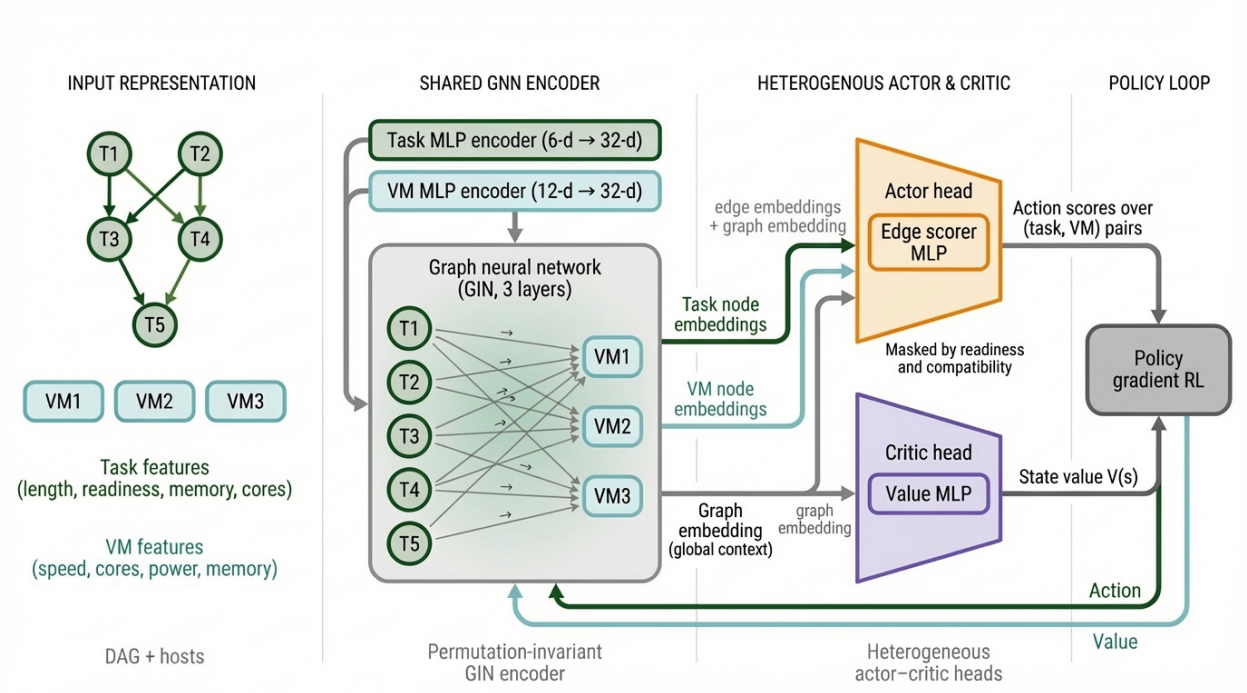


Figure 5: Overview of the proposed GIN-based actor–critic scheduler architecture.

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) 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—corresponding 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. Sharing the backbone encourages the value function to use the same structural features as 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 describes the experimental setup used to study how workflow structure and host heterogeneity interact with our GNN-based reinforcement learning scheduler. We specify the training protocol and evaluation metrics, building on the problem formulation (Section 2.1) and structural decomposition (Section 2.2) presented earlier.

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.
3. **Mixed-topology:** each episode samples Wide or LongCP with equal probability.

All agents train for a fixed number of environment steps with workflows drawn from training seed sets.

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. This ensures performance differences reflect learned policies rather than hyperparameter tuning. We provide the training script (`ablation_gnn_traj_main.py`) and configurations to reproduce all results.

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

This yields 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 long-cp 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 long-cp, 0.44 on wide), but the wide specialist achieves substantially lower active energy (8.38 vs 11.24 on long-cp, 13.03 vs 14.70 on wide). This means the wide specialist wins on energy while maintaining the same makespan, even when tested on long-cp 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 long-cp 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 long-cp specialist has a consistent advantage under HP. On both evaluation domains, it attains lower makespan (2.57 vs 2.75 on long-cp, 0.84 vs 0.87 on wide) and lower active energy (9.68 vs 10.62 on long-cp, 14.70 vs 15.71 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. HP reduces the energy-time trade-off largely to a single time-dominated axis, and in this setting a straightforward heuristic is both simpler and more reliable than the learned policies.

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	Train Domain	Eval Domain	Makespan	Active Energy
HS host configuration				
HS	long_cp	long_cp	1.51	11.24
HS	wide	long_cp	1.51	**8.38**
HS	long_cp	wide	0.44	14.70
HS	wide	wide	0.44	**13.03**
HP host configuration				
HP	long_cp	long_cp	**2.57**	**9.68**
HP	wide	long_cp	2.75	10.62
HP	long_cp	wide	**0.84**	**14.70**
HP	wide	wide	0.87	15.71
AL host configuration				
AL	long_cp	long_cp	**2.07**	**16.46**
AL	wide	long_cp	2.24	19.94
AL	long_cp	wide	**0.68**	**24.15**
AL	wide	wide	0.73	27.71
NA host configuration				
NA	long_cp	long_cp	**2.36**	10.55
NA	wide	long_cp	3.26	**9.02**
NA	long_cp	wide	**0.82**	15.94
NA	wide	wide	0.99	**13.81**

Heterogeneous Aligned (AL). As shown in Table 1,the long-cp specialist achieves lower makespan on both evaluation domains (2.07 vs 2.24 on long-cp, 0.68 vs 0.73 on wide) and lower active energy consumption (16.5 vs 19.9 on long-cp, 24.2 vs 27.7 on wide). This means the long-cp specialist wins on both objectives, even when tested on wide configurations where it was never trained. To understand why this happens, we

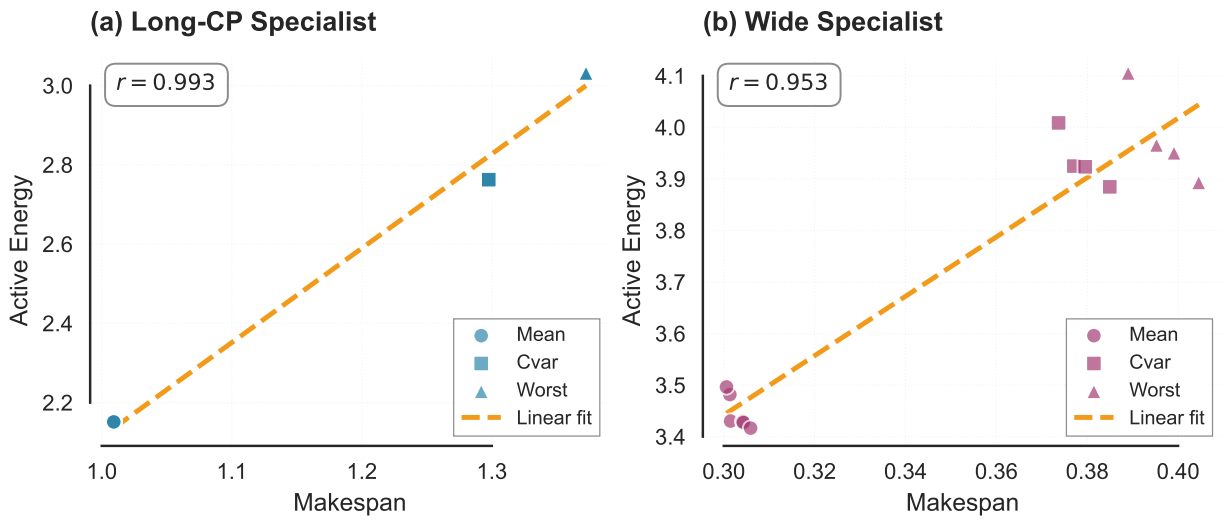


Figure 6: Objective correlation during training in the heterogeneous aligned setting. Each point is a Pareto-optimal checkpoint, with markers indicating mean, CVaR, and worst-case objectives. The long-cp specialist (left) shows near-perfect coupling between makespan and active energy ($r = 0.993$), while the wide specialist (right) shows weaker coupling ($r = 0.953$).

analyze the objective correlation during training (Figure 6). The long-cp specialist exhibits near-perfect correlation between makespan and energy ($r = 0.993$), while the wide specialist shows weaker correlation ($r = 0.953$). This difference shows that training on long dependency chains creates a tight coupling between objectives: when the agent learns to reduce makespan by optimizing the critical path, it simultaneously reduces energy consumption. The wide specialist, trained on more parallel structures, learns a policy where objectives can vary somewhat independently. This tighter coupling in long-cp training acts as an inductive bias that transfers well to other configurations, because the agent learns that structured resource allocation is necessary, not optional.

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 long-cp domain, the long-cp specialist achieves lower makespan (2.36 vs 3.26), but the wide specialist achieves lower active energy (9.02 vs 10.55). The same pattern appears on the wide domain: the long-cp specialist is faster (0.82 vs 0.99), but the wide specialist is more energy-efficient (13.81 vs 15.94).

To confirm these results beyond simple averages, Figure 7 shows empirical attainment functions (EAFs) over 100 test jobs. On both configurations, the wide specialist dominates the low-energy corner, while the long-cp specialist dominates the low-makespan corner. This confirms that each specialist has learned a different bias: one prioritizes time, the other prioritizes energy.

Why this happens: Structure shapes what the agent learns. DAG structure determines which states the agent visits during training, and this shapes what it learns to prioritize. Let π denote a policy over observations s (task requirements and readiness, VM utilization, energy-rate features), and let $d_\pi(s)$ be 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 8 shows that long-cp states fall largely within the outer envelope of wide states. So the performance gaps are not because agents see completely out-of-distribution observations. The states themselves overlap, but what differs is the action affordances and reward geometry in each topology.

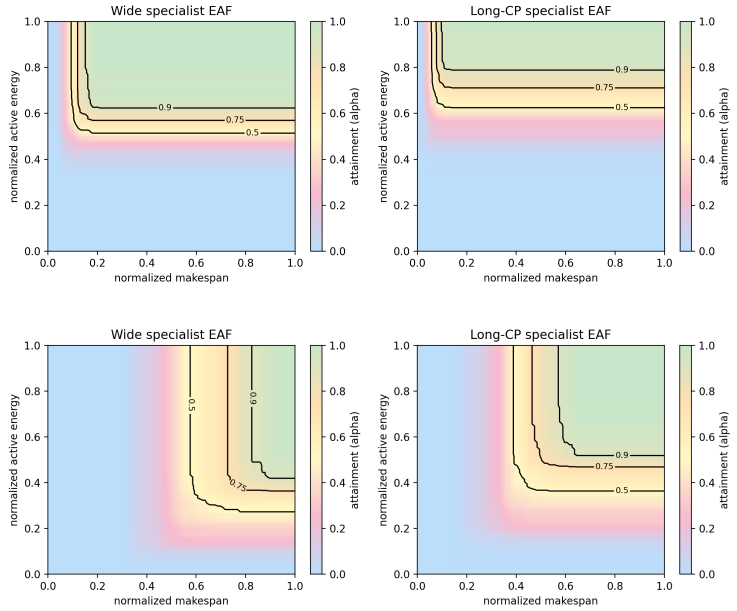


Figure 7: Empirical attainment functions (EAFs) over 100 test jobs for the wide and long-CP specialists on the wide configuration (top) and the long-CP configuration (bottom). Each panel shows the distribution of the trade-off between normalized makespan (x-axis) and normalized active energy (y-axis),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

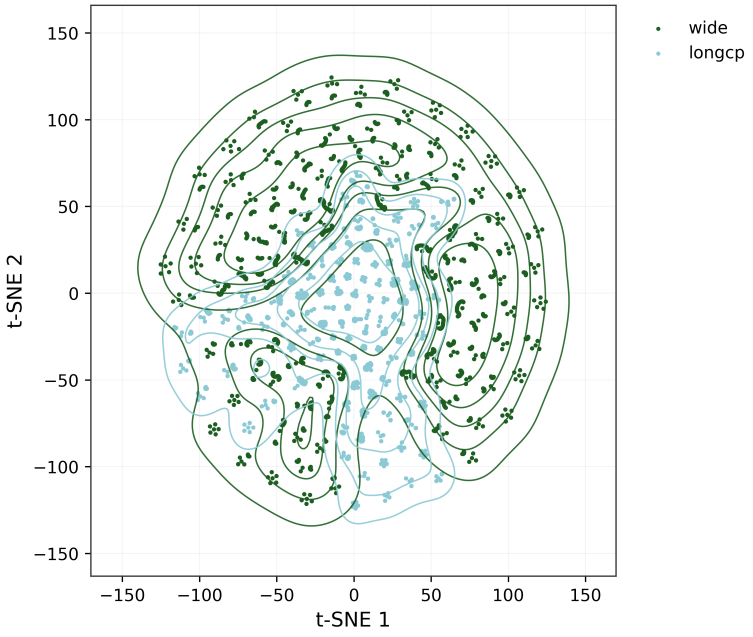


Figure 8: State space coverage under random agents on wide (green) and long-cp (teal) topologies. Points are 2D t-SNE embeddings of collected observations with thin outlines showing local density.

Given that the raw state distributions overlap, the question becomes: how do trained policies actually use this shared state space? Figure 9 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).

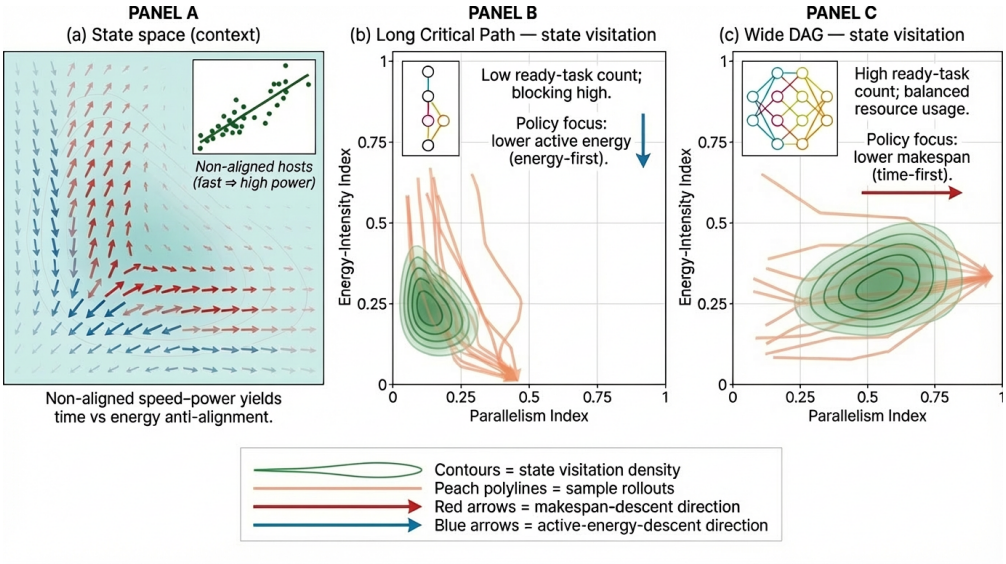


Figure 9: **State visitation under a fixed mixed objective and non-aligned speed–power.** (a) Conceptual state-space diagram with anti-aligned red (makespan) and blue (energy) descent directions, plus an inset showing non-aligned host speed–power. (b) Long Critical Path (LongCP) training: KDE contours 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.

Both agents are trained with the same mixed objective:

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

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

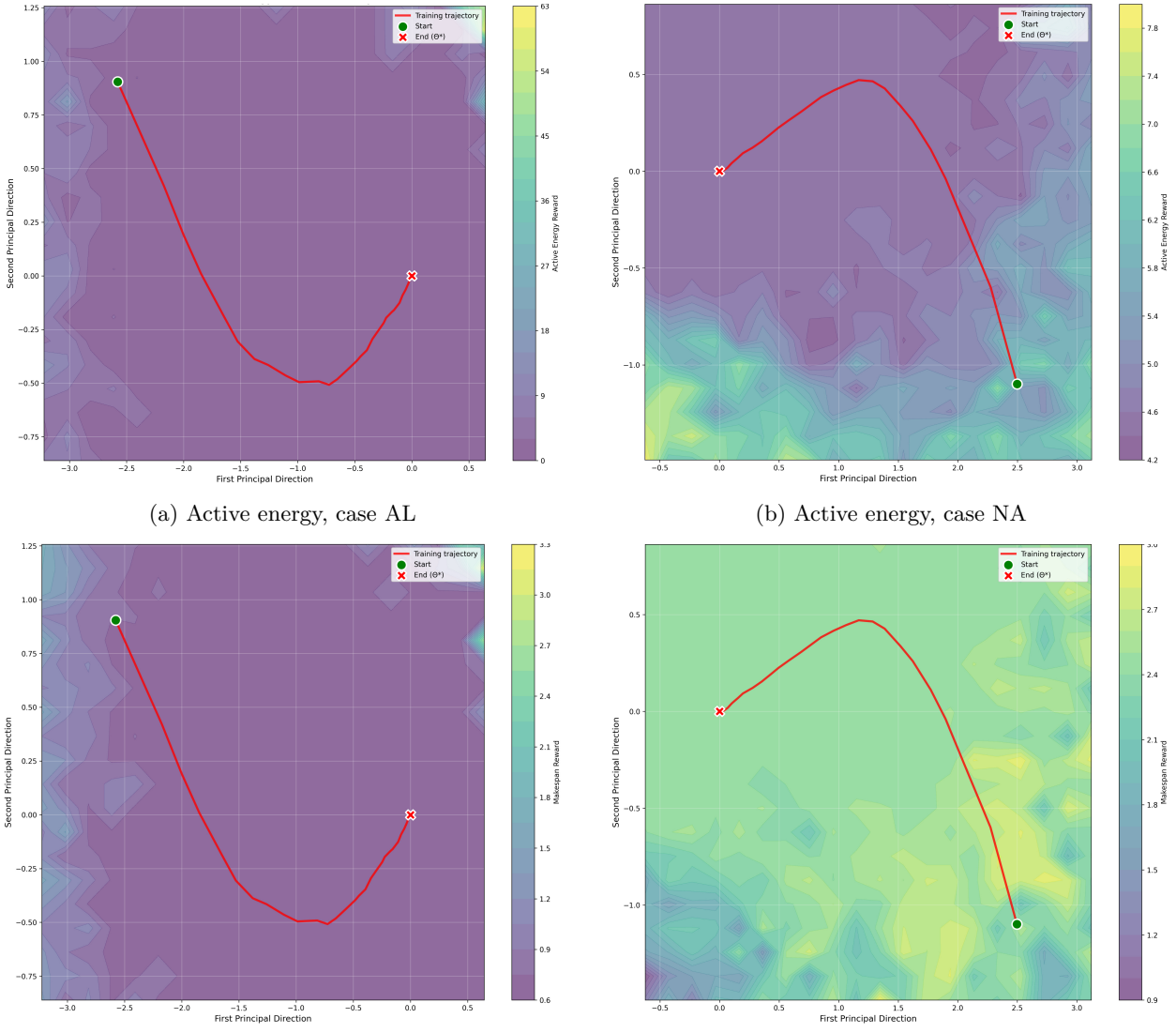
When training on long-cp DAGs (Panel 9b), the agent rarely encounters states with many ready tasks because of the sequential bottlenecks. Most states have low parallelism. In the non-aligned regime, these low-parallelism states often allow multiple VM choices with similar makespan but different power consumption. So the agent learns to focus on reducing energy when it has limited concurrency. The density concentrates in the low-parallelism, moderate-energy region, and the policy develops an energy-first bias.

When training on wide DAGs (Panel 9c), the agent frequently encounters states with large ready sets because of the parallel structure. These high-parallelism states enable many task-VM assignments. In the non-aligned regime, this creates opportunities to reduce makespan by using faster (but higher-power) VMs without drastically increasing energy. So the agent learns to focus on reducing makespan when it has high concurrency. The density shifts toward higher-parallelism, higher-energy regions, and the policy develops 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 compress the feasible manifold toward low parallelism, so d_π places more mass in low-parallelism regions where energy-reducing choices matter. Wide DAGs 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 10 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 long-cp specialist's landscape shows stronger gradients toward low makespan, while the wide specialist's landscape shows stronger gradients toward low energy. This reflects the different biases learned from different $d_\pi(s)$ patterns.

This explains the cross-evaluation results in Table 1. When the long-cp specialist is tested on wide configurations, it still tries to minimize makespan aggressively because that's what it learned to prioritize in low-parallelism states. When the wide specialist is tested on long-cp configurations, it still tries to minimize energy 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.



(a) Active energy, case AL

(b) Active energy, case NA

(c) Makespan, case AL

(d) Makespan, case NA

Figure 10: Actor landscape comparison for active energy (top row) and makespan (bottom row), for AL and NA cases.

5 Limitations and Conclusion

Toward regime-robust scheduling. The detailed cross-regime evaluation we conducted was necessary to understand exactly what works and what does not in each extreme setting. We deliberately tested corner cases: perfectly homogeneous speed (HS), perfectly homogeneous power (HP), perfectly aligned speed and power (AL), and anti-aligned speed and power (NA). Without this analysis, we would not know that simple heuristics beat learned policies when only one resource dimension varies, that the long-cp specialist dominates when objectives are tightly coupled, or that each specialist learns a different bias when objectives conflict. Real cloud environments will likely fall somewhere between these extremes, but understanding the boundary cases points to a practical hybrid approach: detect where you are in this space using simple statistics (variance of speeds, variance of powers, correlation between them), then route to the appropriate expert. In HS use a power heuristic, in HP use a time heuristic, in AL use the long-cp specialist, and in NA use a mixture of specialists weighted by the current parallelism level. This regime-aware routing matches the strategy to the environment and should be more robust than any single policy. Future work can explore learned gating mechanisms and adaptive objective weighting to handle mixed or intermediate regimes that lie between these extremes.

References

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*, 2012a.

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*, 2012b.

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.

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.

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. 2025. Available at: https://www.iea.org/reports/data-centres-and-data-transmission-networks.

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, 2019a.

Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM special interest group on data communication*, pp. 270–288. 2019b.

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

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.

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.

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, 2002a.

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, 2002b.

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.

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.