SCHEMA: HIERARCHICAL REINFORCEMENT LEARNING FOR SCHEDULING UNDER COMMUNICATION CONTENTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Scheduling a large graph of dependent tasks on a small pool of machines is a foundational problem in distributed and high-performance computing, with applications in cloud job scheduling, compiler operator placement, and computation-graph execution on accelerators, among others. It is hardest when tasks vastly outnumber machines, where communication, not computation, dominates makespan. Most schedulers treat the cost of moving data between machines as fixed and known in advance. Under contention this fails. When data transfers compete for shared resources, the cost of communication is no longer fixed but depends on the scheduler's own earlier placements. A value function trained under static costs therefore mispredicts the makespan actually achieved, and the error grows as contention increases. SCHEMA is a reinforcement-learning framework built around this effect. It casts scheduling as a hierarchical Markov decision process whose two levels, selecting a task, then its machine, are pointer networks that attend over the current frontier, keeping inference tractable as graphs reach tens of thousands of nodes. Rather than a fixed cost table, communication is modeled as an environment of capacity-limited link tiers whose occupancy grows as tasks are placed; SCHEMA predicts the tier each data transfer would use under current load and exposes this to the policy alongside structural graph features. Trained with PPO and evaluated on four workloads from 90 to 16,208 nodes against eight classical and learned baselines, SCHEMA achieves shorter schedules, and its margin over the strongest baseline improves with graph size, reaching 29% on the largest workload. Ablation studies attribute independent gains to the congestion-aware environment, the tier-estimation features, and the hierarchical policy, and removing any one lengthens the largest schedule by more than 75%.

1 INTRODUCTION

Scheduling a directed acyclic graph (DAG) of dependent tasks onto a set of machines is a classical problem in computing (Kwok & Ahmad, 1999; Topcuoglu et al., 2002), underlying applications that range from executing computation graphs on clusters of accelerators (Mirhoseini et al., 2017; Addanki et al., 2019a), to placing operations in a compiler (Liu et al., 2024), to dispatching jobs in the cloud (Mao et al., 2019a). Its difficulty depends on the number of machines available relative to the amount of work. When machines are plentiful, tasks can be spread across them and dependencies rarely compete for the same resources. The setting that matters in practice is the opposite one, in which tens of thousands of dependent tasks must run on a few dozen machines. This setting is hard for several reasons at once. The scheduler must order a large number of dependency-constrained decisions toward a single makespan objective; tasks with widely varying execution times must be packed onto machines that are rarely idle; and because those machines are joined by an interconnect of limited capacity, the movement of data between them, rather than the computation itself, becomes the dominant factor in performance.

A scheduler's effectiveness rests on how accurately it accounts for the cost of moving data between machines. Nearly all existing methods make the same assumption, that the cost of a transfer is fixed and known in advance, determined by the volume of data and the bandwidth and latency of the link that carries it. Classical list-scheduling heuristics such as HEFT and CPOP (Topcuoglu et al., 2002)

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

were built on this assumption, and the learning-based schedulers that improved on them inherited it, encoding communication as a static cost on each edge of the graph. The assumption holds only when machines are abundant and transfers rarely coincide. When many tasks compete for few machines, it fails. Transfers collide on the same links, and the cost of moving data no longer depends on the data alone — it depends on what else is moving at the same time.

This dependence makes communication cost *endogenous*, determined not by the workload alone but by the scheduler's own prior placements. A scheduler that assumes static costs therefore optimizes against an inaccurate model of the system it is producing, and underestimates the makespan its schedule will achieve. For learned schedulers the effect is more than a modeling error. A reinforcement-learning scheduler relies on a value function that estimates the makespan-to-go from each state. When this value function is learned under a static cost model, its estimates are systematically lower than the makespan actually realized, and the gap grows as contention increases. The policy is therefore trained toward a target that is least accurate exactly when machines are scarce and communication is heavy.

This work introduces SCHEMA, a scheduler that makes communication contention an explicit, dynamic part of the learning problem. SCHEMA models the interconnect as a set of capacity-limited link tiers. Each transfer occupies a tier for the duration of its data movement and is escalated to a costlier tier when its preferred link is saturated, so that the cost the scheduler observes reflects the traffic its own decisions have created. Scheduling is cast as a hierarchical policy that first selects which ready task to place and then selects its machine, each level parameterized as a Transformer over the current set of ready tasks and available machines. The policy is trained end-to-end with PPO under a dense, potential-based reward for which maximizing return is equivalent to minimizing the terminal makespan.

This study makes the following contributions:

- It identifies that, when machines are scarce, communication cost is endogenous — determined by the scheduler's own prior placements — and shows that scheduling under a static cost model yields makespan estimates that are systematically biased, with the bias growing as contention increases.
- It introduces SCHEMA, a hierarchical reinforcement-learning scheduler that couples a congestion-aware model of the interconnect, based on capacity-limited link tiers, with a two-level Transformer policy that selects tasks and machines over the current ready set.
- It demonstrates that SCHEMA scales to DAGs of tens of thousands of tasks on a limited pool of machines, the setting in which communication contention is the binding constraint.
- It shows across four workloads that SCHEMA's advantage over classical and learned schedulers grows with contention and is largest on the biggest, most contended graph, and attributes this advantage through ablations to the congestion-aware environment, the tier-estimation features, and the hierarchical policy.

2 Related Work

Work on heterogeneous DAG scheduling has followed two directions, improving how scheduling decisions are made and improving how computation and communication costs are modeled. SCHEMA brings these together, learning the scheduling decisions while making communication contention part of the environment the policy sees.

Most classical schedulers are heuristics that build a schedule one decision at a time. The largest class is list scheduling, where tasks are given priorities and then mapped to machines by a fixed rule. HEFT and CPOP (Topcuoglu et al., 2002) are the standard examples. HEFT orders tasks by upward rank, an estimate of the remaining critical path, and places each on the machine that gives it the earliest finish, while CPOP schedules along a critical path found from upward and downward ranks. Related methods such as DLS, HCPT, and SDBATS (Sih & Lee, 1993; Hagras & Janecek, 2003; Munir et al., 2013) differ mainly in how they set priorities. Later heuristics such as PEFT (Arabnejad & Barbosa, 2014) and IPEFT (Zhou et al., 2017) add stronger look-ahead, but the principle is unchanged. Decisions follow hand-crafted priorities and rules fixed in advance, limiting their ability to adapt beyond the assumptions built into them.

Because each placement is committed once made, a good local choice can still lead to a poor overall schedule (Topcuoglu et al., 2002; Xu et al., 2014). Global-search methods address this by exploring many candidate schedules; genetic algorithms, particle-swarm, and ant-colony optimization have all been applied to heterogeneous scheduling (Xu et al., 2014; Izakian et al., 2009; Xiang et al., 2017), recovering from poor local choices at the cost of evaluating many candidates. A separate line handles uncertainty in task durations, communication times, or resource availability (Canon & Jeannot, 2010; Lombardi et al., 2013). For the problem studied here, however, communication cost is not merely an exogenous quantity. It is shaped by earlier decisions, since previous placements determine which transfers later compete for the same resources. These approaches still rely primarily on hand-designed search procedures or scheduling rules.

Learning-based schedulers replace or augment hand-crafted rules with policies learned by interacting with the scheduling environment. Decima (Mao et al., 2019b), DeepSoCS (Sung et al., 2020), and READYS (Grinsztajn et al., 2021) pair reinforcement learning with a graph representation of the DAG. What such a policy can learn, however, is limited by the environment it trains in. An effect the environment leaves out cannot be captured by the policy, and contention is one such effect. For instance, READYS assumes communication overlaps with computation and ignores its cost (Grinsztajn et al., 2021), and SCALE (Xu & He, 2026) learns to select tasks and machines but on small graphs with a static communication cost. Either way, contention never reaches the policy.

Recent work has also applied large language models (LLMs) to heterogeneous workflow scheduling. GLS (Wu et al., 2026) uses an LLM for network-aware task placement, and LLMCS (de Carvalho & Sichman, 2026) for DAG-aware cloud workflow scheduling. GLS fixes task selection to first-come, first-served, while LLMCS combines an LLM with heuristics and optimization. Neither, though, learns task and machine selection jointly, and both treat communication as a fixed input.

Communication has long been central to DAG scheduling. HEFT and similar methods model it through edge costs that depend on the data volume and the two processors (Topcuoglu et al., 2002), but assume contention-free links, where one transfer does not affect another. Contention-aware scheduling drops this assumption by modeling communication resources explicitly. Early work mapped communication edges onto network links, so that link availability limits when a transfer can happen (Sinnen & Sousa, 2004), and later work extended this to heterogeneous interconnects and shared buses, where computation and communication are scheduled together and messages compete for limited resources (Xie et al., 2015; Roy et al., 2020). There, a transfer's delay depends on the rest of the schedule, not on its edge alone. These methods make contention part of the model but still schedule with hand-designed heuristics. A few recent works combine the two. RL-TMC (Rahim et al., 2026), for example, learns task priorities for contention-aware co-scheduling on a shared bus, then passes them to a heuristic that assigns processors and schedules messages under bus-contention constraints.

SCHEMA builds on the intersection of these two lines of work by learning both task selection and machine assignment while allowing communication costs to evolve with the contention induced by the schedule.

3 METHOD

In this section, the scheduling problem and its contention-aware environment are formalized, the hierarchical policy that acts in it is defined, and the procedure that trains it with reinforcement learning is described.

3.1 PROBLEM SETUP

A workload is a directed acyclic graph $G = (V, E)$ with $N = |V|$ tasks. Each edge $(u, v) \in E$ is a dependency, meaning task v cannot start until task u has finished and the data it produces, of size d_{uv}, has reached v's machine. The tasks run on a set of M machines, with $N \gg M$. A task v takes time $w_{v,m}$ on machine m, and each machine runs one task at a time.

Machines are joined by an interconnect with several tiers, ordered from fastest and cheapest to slowest and most costly. The tiers open to a transfer depend on where its two machines sit relative to one another — the same machine, two machines joined by a direct link, two machines in the same

group, or two not-connected machines in different groups. Each tier holds only a limited number of transfers at once. A transfer takes the cheapest open tier that still has availability, and moves up to the next when that one is full, increasing the latency for an available path. It holds its place from the moment its data is produced until the data is read. This cost can be stated precisely. In the standard model, a transfer between machines m and m' has a fixed cost,

$$c_{uv}^{\text{static}}(m, m') = \rho_0(m, m')\, d_{uv}, \tag{1}$$

where $\rho_0(m, m')$ is the per-unit cost set by their relationship, and the cost does not depend on any other transfer. SCHEMA replaces this fixed cost with the tiers above. Each machine pair (m, m') has an ordered ladder of tiers $\mathcal{T}(m, m')$, where tier t has capacity κ_t and per-unit cost ρ_t. When the transfer on edge (u, v) begins, at the finish time f_u of u, it takes the cheapest tier that still has spare capacity,

$$\tau_{uv} = \min\{\, t \in \mathcal{T}(m, m') \;:\; o_t(f_u) < \kappa_t \,\}, \tag{2}$$

where $o_t(f_u)$ counts the transfers already on tier t at time f_u, and the transfer holds that tier over $[\, f_u,\ s_v \,)$, from when u finishes to when v starts at s_v. Its realized cost is

$$c_{uv} = \rho_{\tau_{uv}}\, d_{uv}, \tag{3}$$

with d_{uv} the size of that data. Because o_t is set by every transfer scheduled before it, a transfer's cost depends on the placements already made, which makes its cost *endogenous*. The fixed-cost model (equation 1) is the special case $\kappa_t \to \infty$ of equation 2, where every transfer stays on the cheapest tier regardless of load. A schedule is an assignment of tasks to machines. Once it is fixed, the start and finish times follow from the dependencies, the machine each task runs on, and the transfer costs above. The goal is to find the schedule that minimizes the makespan ($C_{\max}$), the time at which the last task finishes,

$$C_{\max} = \max_{v \in V} f_v. \tag{4}$$

Because $N \gg M$, machines are rarely idle, transfers often overlap, and communication becomes the main factor in makespan. Building the schedule one task at a time makes this a sequential decision problem. At each step, a task whose dependencies are all met is placed on a machine, its start and finish times are set, and the tasks that depend on it become available. This continues until all N tasks are placed. Figure 1 gives an overview of the method on a small instance. At each step a hierarchical policy selects a task and then a machine for it, the environment places the pair and updates the tier occupancy that sets each transfer's cost, and a dense reward on the makespan trains both policies; the policy and its training are described in the subsections that follow.

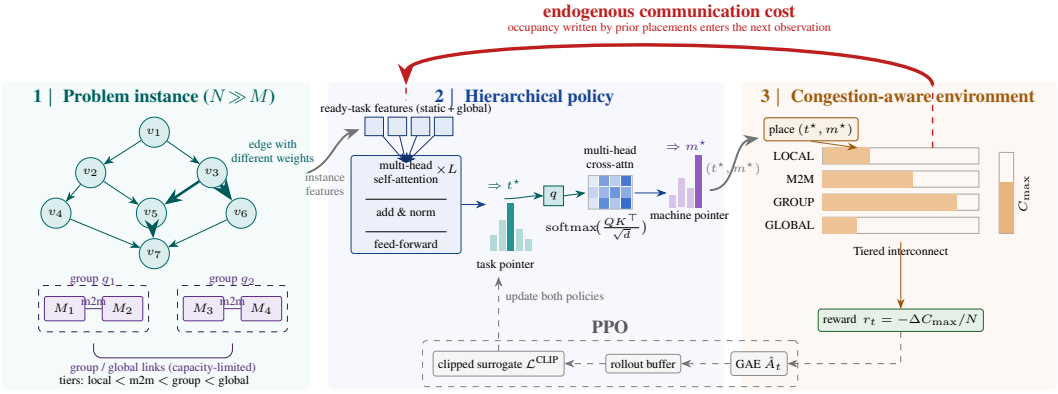


Figure 1: Overview of SCHEMA. A hierarchical policy selects a task and then a machine for it; the environment places the pair and fills link occupancy, which re-enters the next observation (red) and makes communication cost endogenous. Both policies are trained with PPO on the shared reward.

3.2 Scheduling as a Markov Decision Process

SCHEMA builds the schedule with reinforcement learning, treating the sequence of placements as a Markov decision process. At each step the *state* s_t holds the part of the schedule built so far, the

set of ready tasks (all predecessors complete), the time each machine becomes free, and how full each tier is. Exposing the tier occupancy here is what lets the policy see the congestion its earlier placements have created, rather than a fixed cost.

An *action* a_t places one ready task on one machine. This choice is made in two steps, first selecting which ready task to place and then selecting its machine (Section 3.3). After the placement, the task's start and finish times are set, its transfers are committed to their tiers, and the tasks that depend on it may become ready.

The *reward* is dense and tied directly to the makespan. Each placement is penalized by the increase in makespan it causes, divided by the number of tasks,

$$r_t = -\frac{C_{\max}(t) - C_{\max}(t-1)}{N}. \tag{5}$$

Added over an episode, these rewards sum to $-C_{\max}/N$, so maximizing the total reward is the same as minimizing the final makespan. Dividing by N keeps the reward on a similar scale across graphs of very different size, and giving a signal at every step, rather than once at the end, makes the long horizon easier to learn.

3.3 Hierarchical Policy

Each action places one ready task on one machine. SCHEMA makes this choice in two stages. A task policy π_{task} first selects a task from the ready set, and a machine policy π_{mach} then selects a machine for it. Both are pointer networks (Vinyals et al., 2015) that use Transformer self-attention (Vaswani et al., 2017) over their candidates, so each score is computed in the context of the full set rather than in isolation.

The task policy operates on the ready set $\mathcal{R}$, with a feature vector x_i for each task $i \in \mathcal{R}$. A Transformer encoder maps these to contextual embeddings,

$$\{h_i\}_{i\in\mathcal{R}} = \text{Encoder}_{\text{task}}\big(\{x_i\}_{i\in\mathcal{R}}\big), \tag{6}$$

and a pointer head scores each task and forms a distribution over the ready set,

$$\pi_{\text{task}}(i) = \frac{\exp(\mathbf{w}^\top h_i)}{\sum_{i'\in\mathcal{R}} \exp(\mathbf{w}^\top h_{i'})}, \tag{7}$$

from which a task $t^\star$ is sampled.

The machine policy then chooses where to place $t^\star$. Each machine j has a feature vector y_j, encoded by a second Transformer,

$$\{g_j\}_{j=1}^{M} = \text{Encoder}_{\text{mach}}\big(\{y_j\}_{j=1}^{M}\big). \tag{8}$$

A query $q = W_q\,\phi(t^\star)$, where $\phi(t^\star)$ is the selected task's embedding, formed from the selected task cross-attends over the machine embeddings,

$$c = \text{CrossAttn}\big(q,\, \{g_j\}_{j=1}^{M}\big), \tag{9}$$

and a pointer scores each machine by its compatibility with this query,

$$\pi_{\text{mach}}(j) = \frac{\exp(g_j^\top c)}{\sum_{j'=1}^{M} \exp(g_{j'}^\top c)}, \tag{10}$$

from which a machine $m^\star$ is sampled.

What makes these policies congestion-aware is one feature in particular. Alongside static properties of each task and its earliest start and finish times on a machine, SCHEMA adds the cost of moving the task's incoming data right now, read directly from the current tier occupancy. This tier estimate turns contention from a penalty paid at execution into a signal available at the moment of choice. A placement that would push a transfer onto a slower tier is marked expensive before it is ever made. The task policy sees this signal summarized across machines and the machine policy sees it for each machine, so both levels act on the congestion that earlier placements have created. The full list of features is given in Appendix A.

3.4 TRAINING

Both policies are trained together with proximal policy optimization (PPO) (Schulman et al., 2017). A single rollout builds the schedule for one graph, and every placement contributes one transition to each policy, the task policy and the machine policy, both carrying the same reward r_t from Section 3.2. The two policies share one objective, the final makespan, but keep separate parameters, value estimates, and updates.

For each policy, advantages $\hat{A}_t$ are computed with generalized advantage estimation (Schulman et al., 2015) and normalized within each batch. The parameters are updated by maximizing the clipped surrogate objective

$$L(\theta) = \mathbb{E}_t\Big[\min\big(w_t(\theta)\,\hat{A}_t,\ \mathrm{clip}(w_t(\theta),\,1-\epsilon,\,1+\epsilon)\,\hat{A}_t\big)\Big], \qquad w_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}. \tag{11}$$

The full loss adds a value loss and an entropy bonus to this objective, the entropy bonus encouraging exploration. The task policy masks any slot that is not a ready task before its softmax, while the machine policy chooses among all machines, since every machine can run the selected task. Architecture and training hyperparameters, and the hardware used, are listed in Appendix B.

4 EXPERIMENTS

This section evaluates SCHEMA on four task graphs, showing how it compares with classical and learned schedulers, how its advantage depends on contention, and how much each part of the model contributes.

4.1 SETUP

The experiments use four task graphs that span more than two orders of magnitude in size, listed in Table 1. This range exercises each method from small graphs, where machines are not scarce, up to one where tens of thousands of tasks share a small pool of machines. Two of the graphs come from the Standard Task Graph Set (Tobita & Kasahara, 2002). The first is the Robot control graph, derived from the Newton–Euler control computation of Kasahara & Narita (1985). The second is the SPEC fpppp graph (Standard Performance Evaluation Corporation, 1995), taken from a quantum-chemistry workload. The other two are a public GEMM graph for general matrix multiplication and a proprietary matrix-inversion workload (MATINV), the largest of the four. Because MATINV cannot be released, Appendix C reports its scale and topology so that equivalent data can be reconstructed, along with construction details for the GEMM graph.

Table 1: Task-graph workloads and their structure

Workload	#Tasks	#Edges	Levels	Mean width	Max width	Domain
ROBOT	90	135	26	3.5	10	Robot control
SPEC fpppp	336	1,211	17	19.8	136	Quantum chemistry
GEMM	3,392	5,632	24	141.3	1,280	Matrix multiplication
MATINV	16,208	45,840	167	97.1	1,024	Matrix inversion

Every graph is scheduled on 32 machines, connected through the tiered interconnect of Section 3.1. Each machine runs one task at a time, and task durations differ across machines. The tier capacities and costs, and the way the machines are grouped and linked, are given in Appendix D.

SCHEMA is compared against eight baselines, covering both classical and learned scheduling. The classical heuristics are HEFT (Topcuoglu et al., 2002), PEFT (Arabnejad & Barbosa, 2014), and IPEFT (Zhou et al., 2017); the learning-based schedulers are READYS (Grinsztajn et al., 2021), Placeto (Addanki et al., 2019b), DeepSoCS (Sung et al., 2020), Doppler (Yao et al., 2026), and GDP (Zhou et al., 2019). All methods are run on the same graphs, and the learned baselines are trained under the same conditions as SCHEMA (Appendix E). Each method is measured by the makespan of the schedule it produces, the time at which the last task finishes, where lower is better. The makespan of every schedule is computed by the same evaluation function, which applies the

computation and communication model of Section 3.1 to the placements, so the comparison does not depend on the timing any scheduler used while building its schedule. Because the four graphs differ widely in size, makespans are compared across methods on the same graph, not across graphs. The details of the evaluation function are given in Appendix F.

4.2 MAIN RESULTS

Table 2 reports the best makespan each scheduler reaches on the four graphs. On the two smallest, where machines are not scarce and few transfers overlap, the strongest schedulers sit within a few percent of one another. On ROBOT, SCHEMA (768) is within a few percent of the best baseline, DeepSoCS (738), and on SPEC fpppp, it moves ahead, 1,473 against 1,484. In this setting, modeling contention brings little benefit, and SCHEMA performs on par with the strongest heuristics and learned schedulers.

The baselines fall into two families, and the split explains the ordering in Table 2. The classical heuristics, HEFT, PEFT, and IPEFT, encode the core signals of scheduling, such as computation cost, communication cost, and task priority, in fixed logic that is not tuned to any workload, so they stay competitive across graph shapes and sizes. The learned schedulers, READYS, Placeto, Doppler, and GDP, are tuned to the setting they were built for and transfer poorly beyond it, which may explain why they trail on every graph. DeepSoCS is the strongest baseline because it combines the two. It places tasks with the same heuristic rule that HEFT uses and learns only the task ordering with a graph network (Sung et al., 2020), so it keeps the generality of the heuristic while improving on it on most graphs. The exception is GEMM. Its wide, shallow structure (Table 1) leaves so many near-independent tasks ready at once that a fixed rank is already near-optimal, so DeepSoCS's learned ordering adds nothing and it slips behind the heuristics. The same structure exposes how much the learned baselines depend on graph shape, and the rest fall further still, Doppler most of all, which collapses to more than an order of magnitude above the best. What none of the baselines do, heuristic or learned, is account for contention. Each treats the cost of a transfer as fixed and cannot see the contention its own placements create, which is the part SCHEMA adds while using the same core signals in a learned form.

The value of modeling contention grows with the size of the graph. SCHEMA's margin over the best baseline climbs from one graph to the next, from slightly behind on ROBOT to 0.7% on SPEC fpppp, then 3.6% on GEMM and 29.4% on MATINV. On MATINV, where tens of thousands of tasks share the same machines and communication becomes the main cost, SCHEMA reaches 19,403, cutting the best baseline by 8,079 units and coming in more than 40% below HEFT. The classical heuristics and the other learned schedulers, all within reach on the small graphs, are left far behind, and the gap only widens as machines grow scarce.

Table 2: Best makespan of each scheduler on the four graphs (lower is better). The last two rows give the strongest baseline per graph and SCHEMA's improvement over it.

Scheduler	ROBOT	SPEC fpppp	GEMM	MATINV
HEFT	800	1,600	2,751	33,253
PEFT	795	1,594	2,751	31,727
IPEFT	800	1,581	2,751	30,680
DeepSoCS	**738**	1,484	4,257	27,482
Doppler	1,061	1,831	37,210	44,415
READYS	1,179	1,801	6,582	46,465
GDP	1,206	1,843	6,930	47,318
Placeto	839	1,883	8,414	62,158
SCHEMA	768	**1,473**	**2,653**	**19,403**
Best baseline	738	1,484	2,751	27,482
Impr. (%)	-4.1%	0.7%	3.6%	29.4%

4.3 WHY MODELING CONTENTION MATTERS

Table 2 shows that SCHEMA's advantage over schedulers that treat communication as a fixed cost grows with contention, and is largest on MATINV. This section explains the reason. Treating transfer costs as fixed is accurate only when transfers rarely contend for the same links. On the scarce machines of MATINV that breaks down as communication grows, and the fixed-cost estimate of makespan falls further and further below the truth. To measure this, a schedule is produced by a static-cost scheduler (HEFT) on MATINV and then evaluated two ways, without changing any of its placements. The *believed* makespan is the value HEFT computes assuming transfer costs are fixed. The *realized* makespan is the value of the same schedule under the full congestion model, where transfers compete for limited link capacity and are pushed to a higher-cost tier when a link is full. The gap between the two reflects only whether contention is counted, since the schedule itself never changes. This is repeated across a range of communication-to-computation ratios (CCR), the average communication cost divided by the average computation cost (Jiang et al., 2014), which measures how large communication is relative to computation. CCR is increased by scaling the data volumes on the graph's edges while the structure and task durations stay fixed, so that only communication changes.

Figure 2 illustrates the situation. At low CCR, the believed and realized makespans stay close, because few transfers overlap, so contention rarely happens and the fixed-cost estimate holds. As CCR grows the two pull apart, and the static model underestimates the true makespan by an increasing margin. This error is not a fixed offset but grows steadily as contention increases. The effect carries into a reinforcement-learning scheduler. Its value function is trained to predict makespan, and under fixed costs it learns from makespans that underestimate the truth, so its predictions are biased. The resulting bias is largest where SCHEMA operates, on graphs with many tasks, few machines, and heavy communication. By making contention part of the environment, SCHEMA removes this bias at its source, so the estimates that guide it reflect the cost that truly determines makespan.

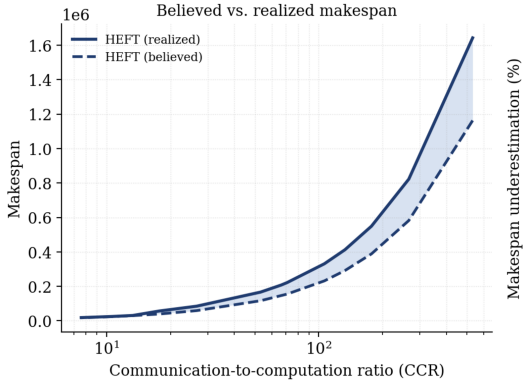


Figure 2: Believed and realized makespan of a fixed HEFT schedule on MATINV as CCR increases.

4.4 ABLATION STUDIES

This section measures how much each part of SCHEMA contributes by removing one part at a time and retraining. The three parts are the congestion in the environment, the tier-estimation features that make it visible to the policy, and the two-level structure of the policy. All ablations use the MATINV graph on 32 machines, and makespan is reported in Table 3.

The first ablation removes congestion from the environment, so the policy trains with every transfer priced as a fixed cost. Section 4.3 shows, on a fixed schedule, that ignoring contention produces an error that grows as contention rises. This ablation measures what that costs once a policy is trained without it. Evaluated under the true dynamics, SCHEMA-STATIC reaches a makespan 75.7% higher than the full model because it has learned to plan around costs that no longer hold once transfers compete for links. Without the congestion model, SCHEMA-STATIC falls behind even the classical heuristics of Table 2, showing that the environment, not the architecture alone, is what makes the learned scheduler effective.

The second ablation, SCHEMA-NoTier, keeps the congestion in the environment but removes the tier-estimation features from the policy's input, so the agent can no longer see which tier a transfer would use or what it would cost. Transfers still compete at execution; the policy simply cannot see it coming. Makespan rises by 84.0%, because the policy places tasks without knowing which choices will overload a link, and only finds out once the cost has been paid.

The third ablation, SCHEMA-Flat, replaces the two-level policy with a single flat policy that scores task and machine together, removing the task-then-machine split. Retrained under the same conditions, it reaches a makespan 91.5% higher than the full model. The joint choice over all ready tasks and machines is a much larger space and is harder to learn on a graph of this size. Each part therefore contributes on its own, and all three are needed to schedule the large, contended workload SCHEMA is built for.

Table 3: Ablation on MatInv. Each row removes one part and retrains; lower makespan is better.

Variant	Makespan	Δ vs. SCHEMA (Full)
SCHEMA (full)	**19,403**	—
SCHEMA-Static (no congestion)	34,097	$+75.7\%$
SCHEMA-NoTier (no tier-estimation features)	35,701	$+84.0\%$
SCHEMA-Flat (no hierarchical policy)	37,155	$+91.5\%$

5 Conclusion

Scheduling large task graphs on a small pool of machines is dominated by communication, and when machines are scarce the cost of communication is endogenous, set by the scheduler's own earlier placements. This work introduced SCHEMA, a reinforcement-learning scheduler that makes this contention part of the environment and learns a two-level policy over it, first selecting a task and then a machine, with a tier estimate that lets each decision see the congestion it would create. On small graphs, where machines are not scarce and contention is mild, SCHEMA is competitive with the strongest classical and learned schedulers. As graphs grow and machines become scarce, its advantage widens, and on MatInv, the largest and most contended graph with 16,208 nodes, it reduces makespan by 29.4% over the best baseline. Ablations show that the congestion-aware environment, the tier-estimation features, and the two-level policy each contribute on their own.

These results come from simulation, on a platform model built to mirror a realistic hardware setup (Appendix D, Appendix F), with a single large proprietary graph alongside the public ones and a fixed family of interconnects. Evaluating SCHEMA on real hardware, across a wider range of interconnect structures, and at larger scale is left to future work.

AI Use Statement

LLM assistants were used selectively to refine the wording and improve the clarity of certain sentences and paragraphs. The manuscript remains predominantly human-authored, and all content, including text revised with LLMs, was carefully reviewed and verified by the authors. LLMs were not involved in the conception of the research, experimental design, data analysis, interpretation of results, or the formulation of scientific conclusions.

Ethics Statement

This work does not involve human participants, personally identifiable information, or sensitive personal data. The proprietary MatInv dataset is used in accordance with applicable confidentiality and organizational requirements. The study focuses on reinforcement learning for computational kernel optimization, and the authors are not aware of specific ethical risks arising from the methodology or experimental setting beyond those generally associated with the responsible development and deployment of machine-learning systems.

Reproducibility Statement

Substantial care has been taken to support the reproducibility and independent verification of the findings. The manuscript and supplementary material describe SCHEMA in detail, including its model architecture, data selection and preprocessing, experimental setup, and evaluation methodology, together with the details needed to re-implement the approach. Due to confidentiality restrictions, the proprietary MatInv dataset and the SCHEMA implementation cannot be publicly released. MatInv, however, is based on standard mathematical operations for which public benchmarks and implementations exist. To further support independent evaluation, SCHEMA is assessed on several publicly available datasets under the same methodological and evaluation framework. Together, these experiments provide a reproducible basis for evaluating the method and show that the central findings do not depend on the restricted MatInv dataset.

Author Contributions

All authors contributed to the conceptualization and development of the proposed methodology. The authors jointly designed the experiments, analyzed the results, and contributed to writing and revising the manuscript.

References

Ravichandra Addanki, Shaileshh Bojja Venkatakrishnan, Shreyan Gupta, Hongzi Mao, and Mohammad Alizadeh. Learning generalizable device placement algorithms for distributed machine learning. In *Advances in Neural Information Processing Systems 32*, pp. 3983–3993, 2019a.

Ravichandra Addanki, Shaileshh Bojja Venkatakrishnan, Shreyan Gupta, Hongzi Mao, and Mohammad Alizadeh. Learning generalizable device placement algorithms for distributed machine learning. In *Advances in Neural Information Processing Systems*, volume 32, pp. 3983–3993, 2019b.

Hamid Arabnejad and Jorge G. Barbosa. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Transactions on Parallel and Distributed Systems*, 25(3):682–694, 2014.

Guy E. Blelloch, Phillip B. Gibbons, and Yossi Matias. Provably efficient scheduling for languages with fine-grained parallelism. *Journal of the ACM*, 46(2):281–321, 1999.

Louis-Claude Canon and Emmanuel Jeannot. Evaluation and optimization of the robustness of DAG schedules in heterogeneous environments. *IEEE Transactions on Parallel and Distributed Systems*, 21(4):532–546, 2010. doi: 10.1109/TPDS.2009.84.

Vinicius Renan de Carvalho and Jaime Simão Sichman. Integrating heuristics and large language models for multi-objective carbon-aware cloud workflow scheduling. *Applied Soft Computing*, 203:115992, 2026. doi: 10.1016/j.asoc.2026.115992.

Nathan Grinsztajn, Olivier Beaumont, Emmanuel Jeannot, and Philippe Preux. READYS: A reinforcement learning based strategy for heterogeneous dynamic scheduling. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 70–81. IEEE, 2021.

Tarek Hagras and Jan Janecek. A simple scheduling heuristic for heterogeneous computing environments. In *Proceedings of the Second International Symposium on Parallel and Distributed Computing*, pp. 104–110. IEEE, 2003. doi: 10.1109/ISPDC.2003.1267650.

Yuxiong He, Charles E. Leiserson, and William M. Leiserson. The cilkview scalability analyzer. In *Proceedings of the 22nd Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pp. 145–156, 2010. doi: 10.1145/1810479.1810509.

Hesam Izakian, Ajith Abraham, and Václav Snášel. Metaheuristic based scheduling meta-tasks in distributed heterogeneous computing systems. *Sensors*, 9(7):5339–5350, 2009.

Yuyi Jiang, Zhiqing Shao, and Yi Guo. A dag scheduling scheme on heterogeneous computing systems using tuple-based chemical reaction optimization. *The Scientific World Journal*, 2014: 404375, 2014. doi: 10.1155/2014/404375.

Hironori Kasahara and Seinosuke Narita. Parallel processing of robot-arm control computation on a multimicroprocessor system. *IEEE Journal of Robotics and Automation*, 1(2):104–113, 1985. doi: 10.1109/JRA.1985.1087004.

Yu-Kwong Kwok and Ishfaq Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys*, 31(4):406–471, 1999. doi: 10.1145/344588.344618.

Dajiang Liu, Yuxin Xia, Jiaxing Shang, Jiang Zhong, Peng Ouyang, and Shouyi Yin. E2EMap: End-to-end reinforcement learning for CGRA compilation via reverse mapping. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 46–60. IEEE, 2024. doi: 10.1109/HPCA57654.2024.00015.

LLVM Project. LLVM 12.0.0 Release Notes. LLVM Project Documentation, 2021. URL `https://releases.llvm.org/12.0.0/docs/ReleaseNotes.html`. Accessed 2026-09-21.

Michele Lombardi, Michela Milano, and Luca Benini. Robust scheduling of task graphs under execution time uncertainty. *IEEE Transactions on Computers*, 62(1):98–111, 2013. doi: 10.1109/TC.2011.203.

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. ACM, 2019a. doi: 10.1145/3341302.3342080.

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 (SIGCOMM)*, pp. 270–288. ACM, 2019b. doi: 10.1145/3341302.3342080.

Azalia Mirhoseini, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. Device placement optimization with reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 2430–2439. PMLR, 2017.

Ehsan Ullah Munir, Sajjad Mohsin, Altaf Hussain, Muhammad Wasif Nisar, and Shoukat Ali. SDBATS: A novel algorithm for task scheduling in heterogeneous computing systems. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, pp. 43–53. IEEE, 2013. doi: 10.1109/IPDPSW.2013.259.

Shaima Rahim, Navaneeth V Sankar, Piyoosh Purushothaman Nair, and Rajesh Devaraj. A reinforcement learning approach to contention-aware co-scheduling in heterogeneous cyber-physical systems. *Computer Communications*, pp. 108616, 2026.

Sanjit Kumar Roy, Rajesh Devaraj, Arnab Sarkar, Kankana Maji, and Sayani Sinha. Contention-aware optimal scheduling of real-time precedence-constrained task graphs on heterogeneous distributed systems. *Journal of Systems Architecture*, 105:101706, 2020.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

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

Gilbert C. Sih and Edward A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Transactions on Parallel and Distributed Systems*, 4(2):175–187, 1993. doi: 10.1109/71.207593.

Oliver Sinnen and Leonel Sousa. List scheduling: extension for contention awareness and evaluation of node priorities for heterogeneous cluster architectures. *Parallel Computing*, 30(1):81–101, 2004.

Standard Performance Evaluation Corporation. SPEC CFP95: 145.fpppp. SPEC CPU95 Benchmark Suite, 1995. URL https://www.spec.org/osg/cpu95/CFP95/145.fpppp/. 145.fpppp, quantum chemistry benchmark.

Tegg Taekyong Sung, Jeongsoo Ha, Jeewoo Kim, Alex Yahja, Chae-Bong Sohn, and Bo Ryu. DeepSoCS: A neural scheduler for heterogeneous system-on-chip (SoC) resource scheduling. *Electronics*, 9(6):936, 2020.

Cheng Tan, Chenhao Xie, Ang Li, Kevin J. Barker, and Antonino Tumeo. OpenCGRA: An open-source unified framework for modeling, testing, and evaluating CGRAs. In *2020 IEEE 38th International Conference on Computer Design (ICCD)*, pp. 381–388. IEEE, 2020. doi: 10.1109/ICCD50377.2020.00070. URL https://doi.org/10.1109/ICCD50377.2020.00070.

Cheng Tan et al. CGRA-Bench: Applications from different domains for CGRA acceleration. GitHub repository. URL https://github.com/tancheng/CGRA-Bench. Commit 6729aaf225d0320e4e0d3b419e20483069a5a69b; accessed 2026-09-21.

Takao Tobita and Hironori Kasahara. A standard task graph set for fair evaluation of multiprocessor scheduling algorithms. *Journal of Scheduling*, 5(5):379–394, 2002. doi: 10.1002/jos.116.

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.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

Ruizhi Wu, Miao Hu, Xianzhi Zhang, Bin Wang, Di Wu, Xuezheng Liu, Chuxuan Zeng, and Liang Du. Gls: Empowering hpc/ai workflow scheduling with gnn-encoded large language models. *Journal of Systems Architecture*, 180:103941, 2026. doi: 10.1016/j.sysarc.2026.103941.

Bin Xiang, Bibo Zhang, and Lin Zhang. Greedy-ant: ant colony system-inspired workflow scheduling for heterogeneous computing. *IEEE Access*, 5:11404–11412, 2017.

Guoqi Xie, Renfa Li, and Keqin Li. Heterogeneity-driven end-to-end synchronized scheduling for precedence constrained tasks and messages on networked embedded systems. *Journal of Parallel and Distributed Computing*, 83:1–12, 2015.

Jierui Lan Zixuan Liang Aiji Liang Xu, Zhifei and Jinxi He. Scale: Scalable cross-attention learning with extrapolation for agentic workflow scheduling. *arXiv preprint arXiv:2606.06820*, 2026.

Yuming Xu, Kenli Li, Jingtong Hu, and Keqin Li. A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. *Information Sciences*, 270: 255–287, 2014.

Xinyu Yao, Daniel Bourgeois, Abhinav Jain, Yuxin Tang, Jiawen Yao, Zhimin Ding, Arlei Silva, and Chris Jermaine. DOPPLER: Dual-policy learning for device assignment in asynchronous dataflow graphs. In *International Conference on Learning Representations*, 2026.

Naqin Zhou, Deyu Qi, Xinyang Wang, Zhishuo Zheng, and Weiwei Lin. A list scheduling algorithm for heterogeneous systems based on a critical node cost table and pessimistic cost table. *Concurrency and Computation: Practice and Experience*, 29(5):e3944, 2017.

Yanqi Zhou, Sudip Roy, Amirali Abdolrashidi, Daniel Wong, Peter C. Ma, Qiumin Xu, Ming Zhong, Hanxiao Liu, Anna Goldie, Azalia Mirhoseini, and James Laudon. GDP: Generalized device placement for dataflow graphs. *arXiv preprint arXiv:1910.01578*, 2019.

A Policy Features

For each pairing of a ready task and a machine, a feature vector of nine values is formed. Five are static properties of the task, computed once from the graph and reused throughout:

- The task's duration, normalized by the largest task duration;
- Its topological level, normalized by the depth of the graph;
- Its number of successors, normalized by the number of tasks;
- Its number of predecessors, normalized by the number of tasks;
- The number of paths passing through it, normalized by the largest such count over all tasks.

The other four depend on the current partial schedule:

- The earliest time the task could start on that machine;
- The earliest time it could finish there, its start time plus its duration on that machine;
- The largest transfer cost its incoming data would incur under the current tier occupancy, the tier estimate of Section 3.3;
- The fraction of the task's predecessors already placed on that machine.

The first three of these are normalized by the graph's maximum upward rank.

The task policy acts on a summary of these features across machines, taking their minimum and mean, giving eighteen values per ready task, together with three global features of the partial schedule:

- The fraction of tasks currently ready;
- The fraction not yet placed;
- The current makespan normalized by the graph's maximum upward rank.

The machine policy acts on the same nine features evaluated for the selected task at each machine, of which the five static features are shared across machines and the other four vary by machine, together with that task's eighteen-value summary used as the query. The availability of each machine is carried by the earliest-start feature rather than as a separate input.

B Training Details

Both policies use the same encoder backbone and training settings, listed in Table 4. Each is trained with its own optimizer on the shared reward of Section 3.2. All experiments ran on a single NVIDIA GeForce RTX 4080 SUPER; scheduling graphs of tens of thousands of tasks did not require large-scale hardware.

C Workload Statistics

C.1 GEMM

The General Matrix Multiplication (GEMM) workload is derived from the integer GEMM kernel provided in CGRA-Bench (Tan et al.), itself based on the PolyBench/C GEMM benchmark. For the experimental workload considered here, the kernel is computing a standard matrix product $C_{8\times 8} = A_{8\times 20} \cdot B_{20\times 8}$, resulting in 1,280 Multiply-Accumulate (MAC) operations. The kernel is compiled using LLVM 12 (LLVM Project, 2021) with vectorization disabled, and the corresponding data-flow graph is generated using CGRA-Mapper (Tan et al., 2020). The resulting graph contains 3,392 operation nodes and 5,632 dependency edges.

Table 4: Architecture and training hyperparameters.

Setting	Value
Embedding dimension	128
Attention heads	4
Transformer layers	4
Feedforward multiplier	2
Discount γ	0.99
GAE λ	0.95
Clip ϵ	0.2
Value coefficient	0.5
Entropy coefficient	0.01
Gradient clip	0.5
Learning rate	3e-4
PPO epochs	4
Batch size	64
Training episodes	1000

C.2 MATINV

MATINV is a matrix-inversion kernel for a 32×32 matrix, compiled into a weighted directed acyclic graph, where nodes are computational tasks and edges are precedence constraints from data dependencies. The graph has 16,208 tasks and 45,840 edges, with a single entry task and 992 terminal tasks. Its longest dependency chain is 167 tasks, while the widest level holds 1,024 tasks. The graph is therefore deep but also wide, with large parts of the computation free to run in parallel.

The graph is moderately connected. The mean in-degree and out-degree are both 2.83, and the maximum in-degree is only 3. The out-degree is highly right-skewed. The median task has one successor, 90% of tasks have at most two, and a few have many more, up to 94. A small number of high-fan-out tasks therefore feed data to many others.

With task durations included, the computation-only critical path (CP) is 1,752 time units and the total computational work W is 68,152 units. The ratio of work to critical path measures the average parallelism of the computation (He et al., 2010),

$$\frac{W}{CP} = \frac{68{,}152}{1{,}752} \approx 38.9,$$

so, ignoring communication, scheduling overhead, and the limited number of machines, the structure permits a speedup of at most about $38.9\times$.

Communication is also a large part of the workload. The CCR is 7.28, which marks a communication-heavy workload.

In the table, w_v is the computation duration of task v, d_{uv} the data size sent along dependency (u, v), and $\bar{c}_{\mathrm{tr}}$ the mean cross-machine transfer cost. The CCR compares the mean communication cost of a dependency with the mean computation cost of a task, and does not depend on any schedule.

The longest-path level of a task v follows the standard notion of DAG depth (Blelloch et al., 1999),

$$\ell(v) = \begin{cases} 1, & \text{if } \mathrm{pred}(v) = \emptyset, \\ 1 + \max\limits_{u \in \mathrm{pred}(v)} \ell(u), & \text{otherwise.} \end{cases}$$

D PLATFORM CONFIGURATION

The machines are divided into groups of equal size. Which tier a transfer can use is decided by the relationship between the machine that produces the data and the machine that reads it. There are four tiers, from cheapest to most costly:

- LOCAL — predecessor and successor tasks are on the same machine, so the data stays on it and is the cheapest to move.

Table 5: Structural and weighted characteristics of the MATINV task graph.

Characteristic	**Value**	**Definition**
Tasks, $\lvert V\rvert$	16,208	Number of vertices
Dependency edges, $\lvert E\rvert$	45,840	Number of directed precedence edges
Entry tasks	1	$\lvert\{v : \deg^-(v) = 0\}\rvert$
Terminal tasks	992	$\lvert\{v : \deg^+(v) = 0\}\rvert$
Longest dependency chain	167	$\max_{p\in\mathcal{P}} \lvert p\rvert$
Widest level	1,024	Most tasks at any longest-path level
Mean in-/out-degree	2.83	$\lvert E\rvert/\lvert V\rvert$
Maximum in-degree	3	$\max_v \deg^-(v)$
Maximum out-degree	94	$\max_v \deg^+(v)$
Out-degree $p_{50}/p_{90}/p_{95}/p_{99}$	1 / 2 / 9 / 32	Percentiles of $\{\deg^+(v)\}$
Critical path, CP	1,752	$\max_{p\in\mathcal{P}} \sum_{v\in p} w_v$
Total work, W	68,152	$\sum_{v\in V} w_v$
Average parallelism	38.9	W/CP
CCR	7.2774	$\frac{\frac{1}{\lvert E\rvert}\sum_{(u,v)\in E} d_{uv}\bar{c}_{\mathrm{tr}}}{\frac{1}{\lvert V\rvert}\sum_{v\in V} w_v}$

- M2M — the two machines are joined by a direct link. The cost of this tier is set per linked pair. Since a link's cost reflects the physical distance between the two machines, a direct link between distant machines can be more expensive than the shared GROUP tier, in which case the cheaper GROUP tier is used instead.
- GROUP — the two machines are in the same group but not directly linked, so the data moves through the shared area of that group.
- GLOBAL — a tier of unlimited capacity and highest cost. It carries data between machines in different groups, and also takes any transfer whose cheaper tiers are all full.

Each relationship determines which tiers a transfer may use. Among those tiers, a transfer takes the cheapest one that currently has room, and moves to the next cheapest when that tier is full. Because the cost of the M2M tier is set per linked pair, a directly linked pair uses M2M only when it is cheaper than its other options; otherwise the cheaper tier is used. The tiers available to each relationship are:

- same machine: LOCAL, GROUP, GLOBAL
- same group, directly linked: M2M, GROUP, GLOBAL
- same group, not linked: GROUP, GLOBAL
- different group, directly linked: M2M, GLOBAL
- different group, not linked: GLOBAL

GLOBAL has unlimited capacity, so a transfer always has an available tier. Table 6 lists the capacity κ_t and per-unit cost ρ_t of each tier on a machine. Every tier except GLOBAL holds a limited number of transfers at once.

Table 6: Tier capacities and per-unit costs.

Tier	Capacity κ_t	Per-unit cost ρ_t
LOCAL	64	2
M2M	8	per linked pair
GROUP	32	10
GLOBAL	∞	40

The machine configuration used in the experiments follows this tier structure. It places 32 machines in 4 groups of 8, joined by 64 direct machine-to-machine links, as shown in Figure 3 with the links drawn and the group boundaries marked. Machines within a group are linked in a one-way loop, and each machine also connects to a machine in the next group. Arranged as a two-dimensional grid, every machine has two ingress and two egress links.

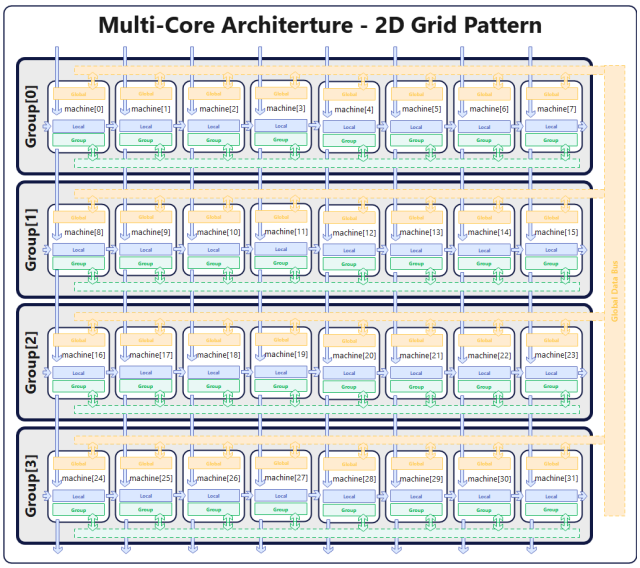


Figure 3: The 32-machine configuration used in the experiments.

E Baseline Details

All baselines are run on the same task graphs and machine configurations as SCHEMA, and their resulting schedules are scored by the same evaluation function (Appendix F). For learned baselines, quantities required by a method that have a direct counterpart in the environment are supplied from the same underlying computation and communication model; proxy features are not introduced for quantities that have no corresponding notion in this study's setting.

Classical heuristics. HEFT, PEFT, and IPEFT are implemented following their original descriptions (Topcuoglu et al., 2002; Arabnejad & Barbosa, 2014; Zhou et al., 2017). These methods assume contention-free communication, so each is given the static per-edge transfer cost of equation 1, the cost of a transfer on its cheapest tier under no load, as its communication input. Each heuristic then builds a schedule under this static cost, and that schedule is replayed through the evaluation function, where its true makespan is measured under contention. This matches how the methods are intended to be used and mirrors the diagnostic of Section 4.3.

Learning-based schedulers. READYS, Placeto, DeepSoCS, Doppler, and GDP are implemented from their published methods and released implementations where available (Grinsztajn et al., 2021; Addanki et al., 2019b; Sung et al., 2020; Yao et al., 2026; Zhou et al., 2019). Their policy architectures, learning objectives, and published hyperparameters are preserved wherever applicable, with only the interface required to operate on the workload graphs and machine configurations. For each method, the selected schedule or placement produced by training is replayed through the evaluation function of Appendix F, and this replayed makespan is the value reported in the comparisons. Thus, differences in the methods' internal communication or execution models do not alter the evaluation model used for the final comparison. For Doppler, Stage-I imitation learning and Stage-II simulation-based RL are used, and Stage III is omitted. Stage III fine-tunes the policy on feedback from real GPU execution through the Einsummable engine, a hardware-in-the-loop step outside the scope of this simulation study; including it would tune Doppler on real-hardware feedback that no other baseline receives. The schedule Doppler produces after Stages I and II is scored by the evaluation function of Appendix F, the same simulation-based measure applied to every method. In Placeto, a single placement sweep rather than two has been used. All other architectural and training hyperparameters follow the corresponding paper or released implementation unless stated otherwise.

F EVALUATION FUNCTION

Given a workload, a task-to-machine schedule, and a platform configuration describing the available machines and resource tiers, the performance model evaluates the schedule against a stateful representation of the multi-machine platform. The platform model traverses the tasks in schedule order while maintaining the evolving platform context. Before evaluating a task, the model restores the context of the assigned machine, estimates the task's execution latency subject to task dependencies, and propagates the resulting state changes to the machine and its associated resource tiers. This evaluation process is repeated for each task in the schedule and, upon completion, returns the cumulative makespan for the workload execution. The updated platform context is also returned, allowing subsequent evaluations to resume from the resulting platform state when the model is invoked iteratively. The evaluation can therefore be expressed as

$$(C_{\max_t},\ \psi_t) \ = \ f\big(\sigma_t,\ G,\ P,\ \psi_{t-1}\big),$$

where $C_{\max_t}$ is the cumulative makespan and ψ_t the platform context (the current tier occupancy) after step t, σ_t is the schedule built so far, G the workload graph, and P the fixed platform configuration.