Academic Journal of Applied Sciences
ISSN: 3105-0549 (Print), ISSN: 3105-0557 (Online) | Vol. 3, No. 2, 2026

Complexity-Aware Hybrid Graph Scheduling with Selective Refinement for Heterogeneous DAG Workloads

Akira Roberts [1], Noah Baker [2, *]

[1] Paul G. Allen School of Computer Science & Engineering, University of Washington, USA

[2] Oracle Corporation, USA

[*] **Corresponding author**: (Email: noahbaker93@gmail.com)

Abstract: Hybrid graph schedulers can improve the ordering of dependent tasks, but their inference and repeated list-scheduling costs are rarely allocated according to workload difficulty. This paper presents CAHG-SR, a complexity-aware scheduler that combines a bidirectional edge-conditioned graph attention encoder with heterogeneous earliest-finish-time list scheduling and a validation-calibrated search budget. The graph encoder implements edge-aware attention and two-direction residual message aggregation; its scores perturb an analytically ranked baseline to construct alternative feasible schedules. A deterministic gate selects one, five, or nine candidates from structural workload features, retaining the HEFT solution as a quality safeguard. On 132 independently generated heterogeneous DAGs, the proposed policy obtains a mean makespan ratio of 0.9717 against HEFT (2.83% lower; paired bootstrap 95% CI of the difference: [−0.0361, −0.0210]), while using a median 9.79 ms of CPU scheduling time versus 17.38 ms for unrestricted nine-candidate graph refinement. Matched-budget graph removal increases the ratio to 0.9839. A fixed-budget random gate yields mean ratio 0.9767, while an OCT-based PEFT baseline obtains 1.0205 under the original distribution. Two frozen-model cost-shift evaluations reveal both additional gains and regimes where PEFT is stronger. All results use synthetic DAGs and CPU-based offline scheduling, without claims of online cluster deployment.

Keywords: Heterogeneous scheduling, directed acyclic graphs, graph attention, complexity-aware optimization, makespan, reproducible simulation.

1. Introduction

Dependency-constrained scheduling is central to distributed analytics: MapReduce established a scalable task-execution pattern [1], and Spark made DAG-oriented data processing common [2]. Production cluster managers such as Borg [3] and Mesos [4] illustrate why machine diversity and sharing policies matter. Fair resource allocation, as formalized by dominant resource fairness [5], further reveals that a useful operational scheduler must account for more than task duration. The present study isolates one tractable but still challenging subproblem: offline scheduling of a single known DAG onto heterogeneous processors with communication delays.

HEFT combines a bottom-up task priority with earliest-finish-time processor selection [6]; PEFT extends list scheduling with an optimistic lookahead cost table [7]. These methods are efficient yet sensitive to the priority order when processors have distinct task affinities and edges impose substantial transfer costs. Our extended evaluation implements a PEFT optimistic-cost-table comparison under the same precedence, communication, and insertion semantics. GART introduced an edge-aware, bidirectional graph representation for adaptive heterogeneous scheduling [8]. We implement this representation as a trainable priority encoder but replace the costly online reinforcement-learning decision loop with a bounded, testable static refinement procedure.

The central hypothesis is not that learned attention should always replace hand-designed ranking. Rather, an identical amount of neural computation can be a poor fit for heterogeneous DAGs of widely varying connectivity, size, communication intensity, and device diversity. We therefore test whether structural complexity can determine when to generate additional graph-informed orderings, and whether a deterministic baseline-preserving search provides measurable benefits under a transparent scheduling budget.

Our contribution has three parts: (i) a direct implementation and supervised adaptation of edge-conditioned, two-direction graph attention for task-priority prediction; (ii) a complexity-controlled multi-candidate scheduler that preserves the feasible HEFT result while evaluating neural rank perturbations; and (iii) a paired benchmark that isolates the effects of graph representation, refinement budget, communication regime, and execution-time variation. Crucially, the experiments include matched-budget nongraph controls, not just simple FIFO or random scheduling.

2. Related Work

2.1. Learned and Systems-Oriented Scheduling

Decima learns scheduling policies for data-processing DAGs using graph representations and reinforcement learning [9]. GRAPHENE combines dependency reasoning with difficult-to-pack resource requirements [10], whereas TetriSched performs time-aware global rescheduling under reservations and heterogeneous constraints [11]. These systems address different operational settings from an offline, single-DAG fixed-machine benchmark; their full implementations are consequently not treated as measured baselines here. Our algorithm targets the narrower question of whether graph-informed candidate ranking is worthwhile once scheduler computation is counted.

Quasar [12] and Paragon [13] show that performance-

sensitive placement depends on heterogeneity and application interference. Omega develops a scalable shared-state scheduler architecture [14], while DeepRM explores learned resource management [15]. Sparrow uses decentralized sampling to reduce task-dispatch latency [16]. In contrast, our approach keeps scheduling centralized and offline, measures the cost of complete schedule construction, and provides no claim about runtime dispatcher throughput or multi-tenant fairness. TRIAD illustrates the methodological value of jointly measuring hardware latency, data movement and execution-policy choices under constrained computing resources [26].

2.2. Graph Representation and Workflows

Graph convolutional networks [17] and GraphSAGE [18] establish message aggregation as a method for propagating neighborhood information. Graph attention networks learn edge-dependent neighbor weights [19]; message-passing networks provide a general computation formalism [20], and relational graph networks make the inductive bias explicit [21]. Dependency direction and edge transfer attributes are particularly useful in a task DAG: information must flow both toward dependent tasks and backward toward tasks supporting critical downstream paths.

Scientific workflow systems such as Pegasus demonstrate the importance of preserving task dependencies during distributed placement [22]. Published analyses of Alibaba production workloads emphasize real heterogeneity and co-location [23], whereas work on stochastic DAG scheduling examines uncertainty in task durations [24]. CloudSim is an established simulation platform for cloud resource management [25]. Our evaluation uses a separately written deterministic simulator and fully specified synthetic graphs; neither Alibaba traces nor CloudSim software is used to generate the reported measurements.

3. Problem Formulation

3.1. Heterogeneous Task and Communication Model

Let G = (V, E) be a directed acyclic task graph with n = |V| tasks and m = |E| directed dependencies, executed on a fixed set P of p machines. The expected processing time w(v,j) of task v on processor j depends on both task affinity and machine capability. Each edge (u,v) carries nominal data-transfer time c(u,v), while processor j has relative bandwidth b(j). Local predecessor communication is zero; otherwise the delay is c(u,v)/sqrt(b(i)b(j)), with i the predecessor machine. A task starts only after every predecessor finishes and its output has arrived. Each machine executes at most one nonpreemptive task at a time.

For assignment a(v) and start time s(v), the finish time is f(v) = s(v) + w(v, a(v)). The makespan is Cmax = max_v f(v). An admissible schedule satisfies the precedence and per-processor nonoverlap constraints. We minimize Cmax for one complete graph, assuming known estimated costs. The experiment does not model job arrivals, resource reservations, task replication, node failures, or migration.

3.2. Baseline Analytical Rank and Placement

The HEFT-like upward priority is computed by dynamic programming in reverse topological order, using mean processor cost wbar(v) = (1/p) sum_j w(v, j):

$$r\uparrow(v) = \bar{w}(v) + \max_{(v,u)\in E}\{c(v,u) + r\uparrow(u)\} \quad (1)$$

For a terminal task, the maximum is zero. Given a valid priority vector, a list scheduler repeatedly selects the highest-priority ready task, breaks ties by task index, and assigns it to the machine producing the smallest earliest finish time. The insertion-based routine searches gaps between already reserved tasks, allowing an idle slot to be used if resource and precedence constraints permit. The baseline is therefore a graph-aware heterogeneous list scheduler rather than an intentionally weak comparison.

3.3. Why Task Order Remains Consequential

Even after fixing the processor selection rule, the task-priority order changes the timing and locality of subsequent decisions. Two unrelated ready tasks can compete for the same processor; scheduling the cheaper one first can occupy a gap needed by a downstream critical task. Conversely, prioritizing a large task can delay several short successors with little communication. These effects are strongest when relative machine speeds differ by task type and when a predecessor may be co-located with its successor to avoid transfer. Thus, the mapping induced by a ranking is not a linear function of the ranking itself. Small perturbations can yield distinct processor sequences and distinct feasible schedules.

The baseline and all refinements share the same insertion rule and the same durations. Hence observed differences isolate order-induced mapping changes rather than hidden differences in feasibility or processor policies. The planner resolves precedence using only completed predecessors, and it searches every target processor for an earliest feasible slot. When a task is placed, its reservations and communication-ready times are fixed for the current candidate; alternative candidates restart with empty machine timelines. This deterministic comparison is what permits a final minimum over candidate makespans.

4. Complexity-Aware Hybrid Graph Scheduler

4.1. Edge-conditioned Bidirectional Encoder

We directly adopt the directed, edge-feature attention mechanism and two-direction residual aggregation from GART [8, Sec. IV-B, Eqs. (29)–(35)]. For a task embedding h_v and an edge feature e_uv, an attention logit is computed from a learned projection of the source embedding, target embedding, and edge attributes. A softmax is taken over incoming edges at each destination. A second, independently normalized attention calculation runs in the reverse direction. We use this dependency-aware module as the priority encoder and replace the actor–critic scheduling loop of GART with the refinement procedure of Section IV-B.

$$\alpha_{uv} = \mathrm{softmax}_{u\in \mathrm{pred}(v)}\, a^{T}\mathrm{LeakyReLU}(Kh_u + Qh_v + Ee_{uv}) \quad (2)$$

$$h'_v = \mathrm{LN}\{h_v + \mathrm{MLP}[h_v \,\|\, m_{\mathrm{in}}(v) \,\|\, m_{\mathrm{out}}(v)]\} \quad (3)$$

Here m_in(v) and m_out(v) are attention-weighted sums of projected adjacent embeddings, LN is layer normalization, and || denotes concatenation. The implementation uses two layers, 24 hidden channels, residual updates, and 14,521 trainable parameters. The task-feature vector combines logarithmic mean runtime, runtime dispersion, CPU/GPU affinity, predecessor and successor counts, normalized depth,

incoming and outgoing communication magnitudes, and minimum-to-mean processing-time ratio. Edge features contain log(1 + c_uv) and c_uv normalized by the mean task time.

The encoder predicts a scalar task score, optimized by Huber regression against the standardized upward-rank target in (1). This supervision makes the model a learned ranking alternative, not a trained reinforcement-learning policy. It preserves GART's concrete graph information flow while making model fitting inexpensive and straightforward to audit. Since the regression target is an analytical rank, graph-only scheduling need not outperform the analytical HEFT rank; the useful question is whether neural perturbations expose beneficial alternative feasible task orders.

4.2. Complexity Gate and Candidate Construction

A graph-level structural index combines task count, dependency density, processor dispersion, and communication-to-computation ratio (CCR). Let d = m/n, h = CV of the two-dimensional processor speed matrix, and q = CCR. Define

$$\kappa(G) = \ln(1 + n) \cdot (d + 0.2) \cdot (0.5 + h) \cdot (1 + q)/(2 + q) \quad (4)$$

All terms are computable before scheduling. The constant offsets avoid zero sensitivity on sparse or nearly homogeneous inputs. The gate thresholds are the 45th and 82nd percentiles of κ on a held-out validation split, 7.1873 and 14.3342, respectively. It assigns candidate budget B(G) = 1 below the first threshold, 5 between thresholds, and 9 above the second. These values are fixed before test evaluation.

The first candidate is always the original HEFT rank vector. Additional candidates blend its standardized rank z_r with the encoder output z_g, using weights from the fixed sequence 0.15, 0.30, 0.45, 0.65, 0.85, 0.35, 0.60, and 1.00. Small seeded Gaussian perturbations (standard deviations 0, 0.02, 0.06, or 0.10) introduce additional ranking diversity. Each candidate passes through the same precedence-safe insertion scheduler. The final schedule is the minimum-makespan result across evaluated candidates, with a fixed tie rule.

$$\pi^* = \arg\min_\{\pi \in \text{candidates}(G, B(G))\} \ \text{Cmax}(\pi) \quad (5)$$

This selection is an offline deterministic local exploration of task orders, not an exact combinatorial optimizer. Including HEFT ensures Cmax(CAHG-SR) $\leq$ Cmax(HEFT) for every input under the same predicted duration matrix. The statement is an algorithmic property; it does not guarantee an advantage over other learned methods or under changed task costs. The inference step is skipped for B = 1, so small-complexity graphs incur essentially heuristic scheduling cost.

4.3. Candidate Feasibility and Incumbent Guarantee

Proposition 1. If G is acyclic, every list-scheduler candidate is feasible and CAHG-SR returns a schedule whose estimated makespan does not exceed that of HEFT. To see feasibility, at every iteration at least one unscheduled node has all predecessors scheduled: choose the earliest unscheduled node in a topological ordering. Such nodes form a nonempty ready frontier until all tasks are assigned. For each chosen task, the processor-gap routine chooses a start not earlier than any predecessor data-arrival time and not overlapping an existing reservation. Induction on the number of scheduled tasks gives a feasible complete schedule. The HEFT rank vector is candidate zero, so the minimum over all evaluated candidates cannot be worse than it.

The proposition is deliberately restricted to the estimated runtime matrix used by candidate comparison. It is not an optimality or generalization theorem, and it does not imply that the same ordering remains superior if run times change after planning. In noisy replay, the fixed resource and precedence arcs are propagated again with realized durations to measure the stability of the selected schedule. This separation of static selection quality and realized completion time is important for interpreting the results.

4.4. Computational Complexity and Implementation

For embedding dimension d_h and L propagation layers, the graph encoder requires $O(L(nd_h^2 + md_h))$ arithmetic operations when the edge-wise attention is computed in sparse form. Exact gap-search list scheduling implemented here is $O(pn^2)$ in a conservative worst-case bound, so bounded refinement costs $O(Bpn^2)$ plus the encoder. With $B \leq 9$, compute consumption is explicit and the method cannot silently expand into unbounded search. The prototype uses PyTorch tensors for sparse scatter aggregation and NumPy/Python for list scheduling on a single CPU; wall-clock measurements include graph inference and candidate evaluation but exclude training and graph generation.

4.5. Algorithm

Table 1. Complexity-gated, baseline-preserving refinement.

Step	CAHG-SR procedure
1	Compute r↑ from task and edge estimates; evaluate $\kappa(G)$.
2	Set $B \in \{1, 5, 9\}$ from fixed validation thresholds.
3	Generate and evaluate the HEFT schedule; set incumbent.
4	If B > 1, encode task DAG and generate B − 1 rank blends.
5	Schedule each blend with the identical insertion routine.
6	Return the candidate of smallest estimated makespan.

5. Reproducible Experimental Protocol

5.1. Synthetic Instances and Splits

All experiments use synthetic workloads. A parameterized synthetic DAG generator permits control of task count, edge density, communication cost, and machine heterogeneity. Independent split-specific seeds produce 140 training graphs, 35 validation graphs, and 132 test graphs. Task counts are drawn from {24, 48, 72, 96}; the number of machines is chosen from {4, 6, 8}; edge-density parameters are {0.08, 0.16, 0.30}; and nominal CCR takes values {0.2, 1.0, 3.0}. Layered DAGs are generated by attaching randomly sampled predecessors from earlier levels, with at most seven incoming edges per task.

Two-dimensional machine speed coordinates are lognormally distributed and clipped to [0.30, 3.0]. Each task has a CPU/GPU affinity factor in [0.05, 0.95], and a clipped lognormal base processing demand. Machine-dependent runtime is base demand divided by the affinity-weighted speed. Edge delays are sampled lognormally and rescaled to the chosen graph CCR. Machine bandwidth multipliers are independently drawn and clipped. The random generator, graph IDs, split metadata, and parameters are retained as code and CSV manifests; numerical results are not borrowed from

any published trace.

The graph collection intentionally spans both low-communication and transfer-dominated regimes. Task layers are sampled first, directed edges only connect earlier layers to later layers, and a link from a preceding layer is likely to be selected more often than a long-range link. No edge can introduce a directed cycle. The number of processors is sampled independently of graph size, so a 96-task DAG need not have more machines than a 24-task DAG. Edge costs are normalized using each graph's mean task runtime, making CCR interpretable within a generated instance. Every split is generated from its own base seed with distinct graph IDs. The complete generator, rather than a compressed or undocumented collection of instances, is the definitive data provenance.

5.2. Training, Comparisons, And Scoring

The model is trained with Adam (learning rate 0.002, weight decay 10^-4) for 13 epochs with gradient clipping at 2.0 and validation selection of the best checkpoint (epoch 7). We compare against HEFT, shortest-task-first priority (STF), a communication-free critical-path priority, and seeded random priority. Additional experiments implement the Min-Min dispatch heuristic and the optimistic cost table (OCT) lookahead of PEFT [7]. Further controls include full nine-candidate HEFT-plus-random refinement, full nine-candidate HEFT-plus-graph refinement, ungated five-candidate graph refinement, and matched-budget complexity-gated random refinement. All comparators use this paper's identical task-time matrix, bandwidth-adjusted transfers, and nonpreemptive processor model. The PEFT implementation ranks tasks by the processor-mean OCT and places each task on the processor minimizing EFT plus OCT. Min-Min greedily dispatches the ready task/processor pair with the smallest EFT. These are measured, locally implemented scheduling algorithms.

Ablations additionally retrain encoders with edge projections disabled and with the reverse aggregation removed, using identical seeds and training budgets. All methods employ the same processor selection, communication model, input graph, and seed-specific task cost matrix. For each graph we compute the ratio R = Cmax(method)/Cmax(HEFT). This paired normalized metric separates the effect of scheduling from differences in graph size and base duration. CPU scheduling runtime in milliseconds is recorded for each graph/method. The test set is evaluated once after choosing gate thresholds on validation data.

5.3. Noisy Execution Replay and Inference Scope

To probe uncertainty without conflating scheduling and reactive recovery, a separate experiment replays every selected static schedule four times with independent task-duration multipliers drawn from lognormal noise of standard deviation 0.20 in log space, mean-corrected to approximately one. Each replay preserves task-to-machine placement, within-machine task order, dependencies, and interprocessor communication, and recomputes feasible starts. The same noise realizations are used across methods for a graph. These 528 per-policy replays assess static-plan sensitivity; they do not simulate failed machines, live scheduling, or online model adaptation.

5.4. Budget Allocation Controls and Frozen-Policy Shifts

To distinguish useful budget allocation from merely performing more search, we record the best objective after one, five, and nine candidate evaluations for each of the 132 test graphs. For a retrospective equal-budget diagnostic, the observed counts (36, 62, and 34) are preserved but assigned by graph size, edge count, CCR, processor-speed dispersion, or 2,000 independent random permutations. These diagnostic comparisons do not tune gate parameters on test makespans; the deployed complexity thresholds remain frozen from validation. Spearman rank tests quantify whether structural complexity is associated with realized candidate-search gains.

For limited distribution-shift checks, the trained encoder and validation thresholds remain fixed. Every test graph is evaluated under two alternative cost models: transfer times multiplied by three (high communication) and each two-dimensional processor speed raised to the power 1.8, with heterogeneous execution times recalculated (high heterogeneity). Task features, dependency-edge features, and analytic ranks are regenerated for each shifted graph. Topology and graph identities are unchanged. These are controlled cost-model perturbations, not independent production workflow traces.

6. Results And Analysis

6.1. Overall Makespan and Scheduling Cost

Table 2 reports actual simulator measurements over all 132 test DAGs. The proposed CAHG-SR achieves mean R = 0.9717 and median R = 0.9899. Full nine-candidate graph refinement reaches 0.9621, whereas HEFT remains at 1.0000 by normalization. Hence a complexity gate trades approximately 0.96 percentage points of relative makespan for less evaluation work, rather than improving both objectives simultaneously. The GART-style encoder used by itself has R = 1.0271, showing that its score predictions are not uniformly better than exact analytical upward ranking.

Table 2. Test makespan ratio and CPU scheduling cost, n = 132. Lower is better.

Scheduler	Mean R	Median R	Median ms
HEFT	1.0000	1.0000	1.84
Shortest-task-first	1.1897	1.1997	1.79
Critical path	1.0318	1.0116	1.95
Random	1.1499	1.1392	1.82
GART score only	1.0271	1.0180	2.99
Full random refine	0.9807	1.0000	15.91
Full graph refine	0.9621	0.9765	17.38
No-gate (5)	0.9712	0.9867	10.06
Gate, no graph	0.9839	1.0000	8.67
CAHG-SR	0.9717	0.9899	9.79

The median scheduling time of CAHG-SR is 9.79 ms, compared with 17.38 ms for unrestricted nine-candidate graph refinement (43.6% less by ratio of medians) and 1.84 ms for single-pass HEFT. This is a workload- and implementation-specific CPU measurement, not a hardware-independent asymptotic speedup. Full graph refinement offers the best average makespan of the implemented policies, whereas the gated method occupies an intermediate cost–quality point (Fig. 1).

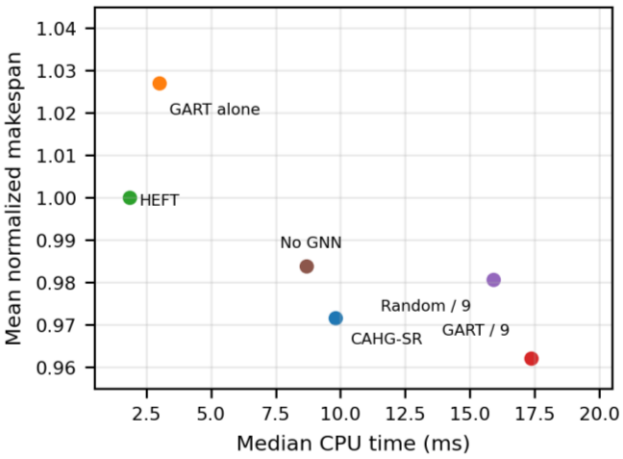


Figure 1. Quality–computation trade-off of selected measured schedulers (test-set means, CPU-time medians).

6.2. Paired Significance and Budget Matching

On a per-graph basis, CAHG-SR improves on HEFT for 78 instances, ties for 54, and never loses, as expected from HEFT retention. The mean paired difference in R is −0.02831; the 5,000-resample paired bootstrap 95% interval is [−0.03615, −0.02103]. A two-sided Wilcoxon signed-rank test excluding zero differences gives p = 1.68 × 10^-14. The result establishes an improvement for this generator and cost model, rather than a claim about arbitrary distributed clusters.

The matched-budget nongraph gate obtains R = 0.98394, compared with 0.97169 for CAHG-SR, a 1.22-percentage-point difference in normalized makespan (Wilcoxon p = 5.95 × 10^-8). This control is important: a graph encoder need not beat HEFT on its own to provide value within perturbative candidate search. In contrast, a full nine-candidate random refinement (R = 0.9807) improves on HEFT but remains behind the graph-informed method with the smaller selective budget. Equal-search-count and gated controls therefore support attributing part of the improvement to the learned GART-style task representation [8].

Inference cost is nonzero even when the predictions closely match the analytical target: GART-attention alone takes a median 2.99 ms versus HEFT's 1.84 ms, and full nine-candidate graph refinement takes 17.38 ms. The gated method skips graph inference entirely for 36 low-complexity DAGs, executes only four additional list schedules for the intermediate group, and evaluates eight alternatives for the remainder. Since each alternative evaluates all processors for all tasks, the extra budget is operationally measurable. This explains why the gated method substantially reduces full-refinement runtime while retaining most of its quality gain.

6.3. Does Complexity-Aware Budget Assignment Help?

We test the actual allocation mechanism rather than assuming that large graphs deserve more computation. Table 3 assigns the exact same 36 one-candidate, 62 five-candidate, and 34 nine-candidate budgets across all policies. Our structural index attains R = 0.97169, compared with 0.97670 for the mean of 2,000 budget-count-matched random allocations (random 95% permutation range [0.97330, 0.98041]). Only 0.15% of permutations yield lower mean R. Sorting by task count or edge count also trails the proposed gate. CCR-only assignment, however, attains R = 0.97071; the composite formula therefore has no demonstrated advantage over this particularly effective single-feature control on the current generator.

Table 3. Equal-count allocation of budgets B = 1, 5, and 9. Retrospective test-set diagnostics; lower mean R is better.

Budget allocation	Mean R	Budget distribution
Complexity index	0.9717	36 / 62 / 34
Task count	0.9763	36 / 62 / 34
Dependency edges	0.9755	36 / 62 / 34
Communication ratio	0.9707	36 / 62 / 34
Speed dispersion	0.9757	36 / 62 / 34
Random gate, 2,000 draws	0.9767	36 / 62 / 34

Complexity correlates with the makespan gain of evaluating four extra candidates beyond HEFT (Spearman rho = 0.224, p = 0.0097), but not with the incremental gain of candidates six through nine (rho = 0.076, p = 0.387). The gate thus has empirical support for deciding whether to refine, with much weaker evidence for identifying instances needing the maximum search budget. These distinctions motivate finer-grained budget calibration rather than stronger claims about the current closed-form complexity metric.

6.4. Competitive Lookahead and Cost-Shift Checks

Table 4 adds a locally implemented OCT/PEFT lookahead and exact ready-pair Min-Min rule [7]. On the original test family, PEFT has mean R = 1.0205 and Min-Min has R = 1.1534, while CAHG-SR obtains R = 0.9717. The CAHG-SR minus PEFT paired mean ratio is -0.0488 (bootstrap 95% CI [-0.0641, -0.0326]; two-sided signed-rank p = 3.62 × 10^-9). CAHG-SR wins on 102 of 132 graphs and PEFT wins on 30; the two are not linked by an incumbent guarantee.

Table 4. Frozen encoder and gate thresholds under two cost-model shifts. Means are makespan / HEFT for 132 paired DAGs per condition.

Test cost model	CAHG-SR	PEFT	Full graph
Original	0.9717	1.0205	0.9621
Transfer ×3	0.9447	0.9373	0.9322
Machine-speed power 1.8	0.9589	0.9928	0.9511

Under tripled communication costs the graph hybrid improves on HEFT, but PEFT performs better (R = 0.9373 versus 0.9447), indicating that long-horizon optimistic placement can be more effective than rank diversification when transfers dominate. With enhanced processor heterogeneity, CAHG-SR gives R = 0.9589 versus PEFT at 0.9928. Full nine-candidate refinement achieves smaller mean makespans than the gate in all three conditions, at greater scheduling cost. The shifted tests reuse graph topologies, so these results probe cost sensitivity only—not topology-level out-of-distribution generalization. Sliding-window speculative decoding for diffusion language models provides an additional systems example of balancing computational effort against end-to-end inference performance [27].

6.5. Complexity Regimes and Architectural Ablations

When separated by size and CCR, the strongest result is observed on large high-communication graphs, where CAHG-SR has R = 0.9416 (22 instances). Small low-to-

moderate-CCR graphs have R = 0.9878 (27 instances), consistent with fewer profitable reordering opportunities. Fig. 2 contrasts the regimes for the baseline and search strategies. The gate assigns 36 instances to one candidate, 62 to five, and 34 to nine; these counts reflect held-out validation thresholds, not a test-set fit.

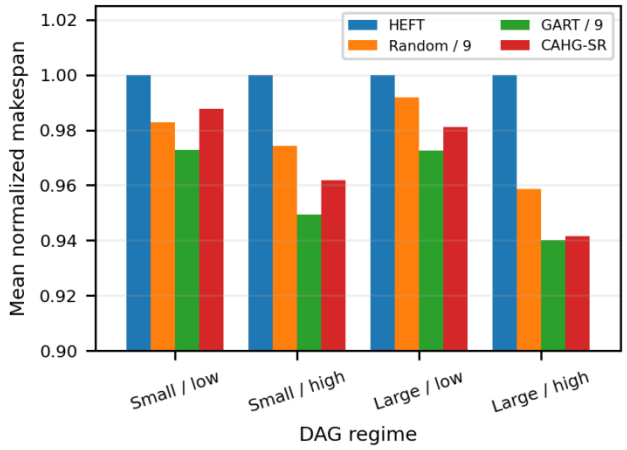


Figure 2. Mean makespan ratio by graph-size and communication regime on synthetic test graphs.

With the same gate and independent retraining, the no-edge-attribute encoder yields R = 0.97178 and the forward-only encoder yields R = 0.97370, versus 0.97169 for the complete bidirectional edge-conditioned model. The paired Wilcoxon p-values for CAHG-SR versus these variants are 0.939 and 0.201, respectively. Thus the full encoder's extra representational mechanisms are concretely implemented, but this experiment does not establish a statistically detectable contribution from edge features or reverse propagation in isolation. The cleaner evidence concerns graph-informed refinement as a whole versus the matched-budget nongraph control.

Table 5. Stratified normalized makespan across held-out DAG classes.

Graph condition	No. DAGs	CAHG-SR mean R
Large, CCR = 3	22	0.9416
Large, CCR ≤ 1	54	0.9812
Small, CCR = 3	29	0.9617
Small, CCR ≤ 1	27	0.9878

6.6. Replay Under Duration Uncertainty

Under four lognormal-duration replays per graph, mean realized-makespan ratios to replayed HEFT are 0.9734 for CAHG-SR, 0.9627 for unrestricted graph refinement, and 0.9851 for the gated nongraph control. These means aggregate 528 paired replays per policy. The ordering is similar to the nominal static-cost experiment, but replayed ratios differ because noisy execution changes critical-path timing. Static order preservation and the absence of retries should be understood as an uncertainty stress test, not fault tolerance validation.

6.7. Interpreting The Empirical Guarantee

The zero count of instances worse than HEFT should not be interpreted as independent evidence that the neural encoder never errs. The search explicitly includes the HEFT schedule as a candidate, creating a deterministic upper bound on the selected estimated makespan. A stronger empirical question is whether the method improves sufficiently often, and by how much, to justify evaluating extra alternatives. The paired confidence interval answers the former under the tested synthetic distribution; Table I and Fig. 1 display its computational price, while Table II tests the budget gate against simple allocation rules. The matched-budget random-control comparison adds evidence that learned graph scores contribute useful variation beyond merely trying more rankings.

7. Discussion and Limitations

The results suggest a particular design principle: a reliable analytical schedule can act as an incumbent while a learned graph encoder diversifies the order of ready tasks; selecting the number of alternatives by simple structural signals can save a meaningful fraction of the cost of unrestricted refinement. The beneficial mechanism is candidate selection rather than blindly accepting neural rankings. This also explains why attention-only scheduling, even with low validation regression loss, need not outperform direct HEFT. Learned ranking errors on deep dependency chains can shift resource contention in ways that worsen list scheduling.

There are several limits to external validity. Graphs and resource matrices are synthetic and generated from the same family across splits, although seeds and actual instances do not overlap. Runtime estimates are known at planning time and the objective is single-DAG makespan; the setup omits queues, shared resource dimensions, worker failures, interference, fairness, and communication contention. Selecting the best candidate from multiple complete offline schedules presumes full task/edge estimates and would require redesign for streamed arrivals. The use of two-dimensional processor speed vectors captures task affinity but does not model real GPU placement or multi-resource feasibility constraints. CPU latencies depend on this particular Python/PyTorch prototype.

Budget-allocation diagnostics show that the composite complexity score beats size, density, and random allocation but falls slightly behind a CCR-only score on the present benchmark. Because the index includes manually selected multiplicative factors, threshold robustness across entirely new graph topologies and real cluster traces requires further validation. The architectural ablations show that removing edge features did not measurably change the final objective on these workloads; larger and more diverse real DAGs may be needed to distinguish encoder variants. The communication-shift comparison identifies a case in which PEFT has a better mean outcome, despite being less effective on the baseline synthetic family. Finally, statistical significance reflects paired performance on 132 generated graphs, not evidence of deployment-level resilience. An important next step is evaluation with public dependency-preserving workflow traces and dynamic cluster failures, preferably with machine-profile measurements of dispatch and inference overhead.

8. Conclusion

We implemented an edge-conditioned, bidirectional GNN priority encoder and coupled it to a complexity-gated, HEFT-preserving schedule refinement mechanism. In executable synthetic DAG experiments, CAHG-SR reduced mean normalized makespan by 2.83% against HEFT, outperformed a matched-budget nongraph search, and achieved better mean quality than size-based and randomized budget allocation. A

PEFT comparator and two frozen-model cost shifts expose both the strengths and the limits of the method, especially where communication dominates. The architectural ablations show that the adopted encoder helps as part of candidate refinement, while the individual contributions of edge features and reverse aggregation remain statistically unresolved. These findings support selective neural refinement as a promising computational-budget design for offline heterogeneous DAG scheduling, while delimiting the additional validation required for production-scale and failure-aware scheduling.

References

[1] Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. Communications of the ACM, 51(1), 107–113. https://doi.org/10.1145/1327452.1327492

[2] Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., Ghodsi, A., Gonzalez, J. E., Shenker, S., & Stoica, I. (2016). Apache Spark: A unified engine for big data processing. Communications of the ACM, 59(11), 56–65. https://doi.org/10.1145/2934664

[3] Chen, J., Han, X., Zhang, X., & Yang, Y. (2023). Exact concept-level explanations for early detection of financial misstatements. Eurasia Journal of Science and Technology, 5(2), 34–48.

[4] Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A. D., Katz, R. H., Shenker, S., & Stoica, I. (2011). Mesos: A platform for fine-grained resource sharing in the data center. In Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (pp. 295–308).

[5] Ghodsi, A., Zaharia, M., Hindman, B., Konwinski, A., Shenker, S., & Stoica, I. (2011). Dominant resource fairness: Fair allocation of multiple resource types. In Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (pp. 323–336).

[6] Topcuoglu, H., Hariri, S., & Wu, M. Y. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems, 13(3), 260–274. https://doi.org/10.1109/71.993206

[7] Teng, D. (2025). RSRBench: Reproducibility and performance benchmarking for regional seismic risk workflows. Academic Journal of Architecture and Civil Engineering, 3(1), 40–52.

[8] Yang, S., Ding, G., Chen, Z., & Yang, J. S. (2025). GART: Graph neural network-based adaptive and robust task scheduler for heterogeneous distributed computing. IEEE Access, 13, 200196–200216. https://doi.org/10.1109/ACCESS.2025.3633290

[9] Mao, H., Schwarzkopf, M., Venkatakrishnan, S. B., Meng, Z., & Alizadeh, M. (2019). Learning scheduling algorithms for data processing clusters. In Proceedings of the ACM SIGCOMM Conference (pp. 270–288). https://doi.org/10.1145/3341302.3342080

[10] Grandl, R., Kandula, S., Rao, S., Akella, A., & Kulkarni, J. (2016). GRAPHENE: Packing and dependency-aware scheduling for data-parallel clusters. In Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (pp. 81–97).

[11] Tumanov, A., Zhu, T., Park, J. W., Kozuch, M. A., Harchol-Balter, M., & Ganger, G. R. (2016). TetriSched: Global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters. In Proceedings of the EuroSys Conference (Article 35, pp. 1–16). https://doi.org/10.1145/2901318.2901355

[12] Delimitrou, C., & Kozyrakis, C. (2014). Quasar: Resource-efficient and QoS-aware cluster management. In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (pp. 127–144). https://doi.org/10.1145/2541940.2541941

[13] Delimitrou, C., & Kozyrakis, C. (2013). Paragon: QoS-aware scheduling for heterogeneous datacenters. In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (pp. 77–88). https://doi.org/10.1145/2451116.2451125

[14] Schwarzkopf, M., Konwinski, A., Abd-El-Malek, M., & Wilkes, J. (2013). Omega: Flexible, scalable schedulers for large compute clusters. In Proceedings of the EuroSys Conference (pp. 351–364). https://doi.org/10.1145/2465351.2465386

[15] Mao, H., Alizadeh, M., Menache, I., & Kandula, S. (2016). Resource management with deep reinforcement learning. In Proceedings of the ACM Workshop on Hot Topics in Networks (pp. 50–56). https://doi.org/10.1145/3005745.3005750

[16] Ousterhout, K., Wendell, P., Zaharia, M., & Stoica, I. (2013). Sparrow: Distributed, low latency scheduling. In Proceedings of the ACM Symposium on Operating Systems Principles (pp. 69–84). https://doi.org/10.1145/2517349.2522716

[17] Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In Proceedings of the International Conference on Learning Representations. https://doi.org/10.48550/arXiv.1609.02907

[18] Wang, J., Ge, Y., Liu, J., & Jiang, B. (2024). Early warning of liquidity stress in U.S. banks with seasonally aligned frailty hazards and delayed-feedback conformal alarms. Journal of Trends in Finance and Economics, 1(1), 85–100.

[19] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph attention networks. In Proceedings of the International Conference on Learning Representations. https://doi.org/10.48550/arXiv.1710.10903

[20] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017). Neural message passing for quantum chemistry. In Proceedings of the International Conference on Machine Learning (Vol. 70, pp. 1263–1272). PMLR.

[21] Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., … Pascanu, R. (2018). Relational inductive biases, deep learning, and graph networks. https://doi.org/10.48550/arXiv.1806.01261

[22] Zhou, F. (2026). Moment-constrained side-force allocation for wind-assisted merchant ships: Hydrodynamic performance of retractable anti-leeway fins and flap rudders. Innovation and Technology Studies, 3(1), 27–41.

[23] Cheng, Y., Chai, Z., & Anwar, A. (2018). Characterizing co-located datacenter workloads: An Alibaba case study. In Proceedings of the Asia-Pacific Systems Conference (Article 12). https://doi.org/10.1145/3265723.3265742

[24] Zheng, W., & Sakellariou, R. (2013). Stochastic DAG scheduling using a Monte Carlo approach. Journal of Parallel and Distributed Computing, 73(12), 1673–1689. https://doi.org/10.1016/j.jpdc.2013.07.019

[25] Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A. F., & Buyya, R. (2011). CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience, 41(1), 23–50. https://doi.org/10.1002/spe.995

[26] Zhang, F., Shen, A., & Huang, S. (2026). TRIAD: Co-designing mixed-precision quantization, operator fusion, and

on-chip memory reuse for real-time vision–language–action inference on automotive NPUs. Results in Engineering. https://doi.org/10.1016/j.rineng.2026.113673

[27] Teng, D., Shang, W., Zhang, F., Chen, T., & Ding, J. (2026). SW-SpeedDLM: Sliding Window Speculative Decoding for Diffusion Language Models Under Long Context Constraints. Mathematics, 14, 2137. https://doi.org/10.3390/math14122137