

An ML-Based Hybrid Task Scheduler for Classical–Quantum Computing Environments Using Real Graph-Derived Workloads

Peter Nimbe

`peter.nimbe@uenr.edu.gh`

UENR

Nicodemus Songose Awarayi

UENR

Vivian Akoto-Adjepong

UENR

Faiza Umar Bawah

UENR

Patrick Kwabena Mensah

UENR

Obed Appiah

UENR

Christopher Bombie Ninfaakang

UENR

Mighty Abra Ayidzoe

UENR

Augustine Takyi

UENR

Bernard Andoh

UENR

Daniel Asamoah Apraku

UENR

Frank Opoku Kwarteng

UENR

Farida Osman

UENR

Keywords: Hybrid classical–quantum computing, task scheduling, resource allocation, machine learning, Max-Cut, Qiskit, queue-aware scheduling, heterogeneous computing

Posted Date: September 15th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-9774486/v1>

License: © ⓘ This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

An ML-Based Hybrid Task Scheduler for Classical–Quantum Computing Environments Using Real Graph-Derived Workloads

Peter Nimbe, Nicodemus Songose Awarayi, Vivian Akoto-Adjepong, Faiza Umar Bawah, Patrick Kwabena Mensah, Obed Appiah, Christopher Bombie Ninfaakang, Mighty Abra Ayidzoe, Augustine Takyi, Bernard Andoh, Daniel Asamoah Apraku, Frank Opoku Kwarteng, Farida Osman

Department of Computer Science and Informatics, UENR, P. O. Box 214, Sunyani, Ghana

peter.nimbe@uenr.edu.gh, nicodemus.awarayi@uenr.edu.gh, vivian.akoto-
adjepong@uenr.edu.gh, faiza.bawah@uenr.edu.gh, patrick.mensah@uenr.edu.gh,
obed.appiah@uenr.edu.gh, chrsitopher.ninfaakang@uenr.edu.gh,
mighty.ayidzoe@uenr.edu.gh, augustine.takyi@uenr.edu.gh, bernard.andoh@uenr.edu.gh,
danielasamoahapraku@gmail.com, proffnarrow838@gmail.com,
akosua49@gmail.com

Corresponding author: Peter Nimbe

Abstract

Hybrid classical-quantum computing creates a scheduling challenge. Tasks need assignment over time and across different resources, each with unique execution, setup, and queue traits. This paper introduces a machine-learning-based scheduler for placing tasks between classical CPU and quantum processing resources in a simulated environment. We use the Wiki-Vote network from the Stanford Network Analysis Project as a workload source. We build a benchmark of connected Max-Cut task instances by sampling subgraphs of different sizes. Each task is profiled under both classical and quantum execution modes. The resulting dataset combines task features, runtime behaviour, and queue-state conditions. We evaluate multiple learning models for predicting task placement. Their scheduling decisions are compared to classical-only, quantum-only, and rule-based hybrid baselines. Results show that our ML-based scheduler outperforms all others. It achieves the lowest average completion time, smallest makespan, highest throughput, and best decision accuracy. Feature analysis reveals that queue pressure, system regime, and task priority are key factors in placement decisions. Graph and circuit complexity also enhance predictive power. These findings suggest that data-driven scheduling can boost resource efficiency in hybrid classical-quantum environments. This approach offers a practical path for intelligent task placement in emerging heterogeneous systems.

Keywords

Hybrid classical–quantum computing; task scheduling; resource allocation; machine learning; Max-Cut; Qiskit; queue-aware scheduling; heterogeneous computing

1. Introduction

The rapid rise of quantum computing has led to new computing environments where classical and quantum resources work together. In these setups, scheduling goes beyond just assigning processors and ordering tasks. A scheduler now needs to decide if a task should stay on a classical resource or switch to a quantum one. This decision must consider execution overhead, queue behavior, and task suitability. Thus, scheduling becomes a task-placement and resource-allocation challenge.

Classical scheduling methods were designed for homogeneous or slightly heterogeneous systems, like multicore processors and clusters. They aid in load balancing and resource management. However, they struggle with quantum processors. Quantum resources may involve higher setup costs, limited availability, and delays that far exceed conventional CPU execution. Thus, static allocation rules may not be effective. The benefits of classical or quantum execution depend on workload structure and system conditions (Shehata et al., 2026).

It's important to note that hybrid classical-quantum systems shouldn't assume that quantum execution is always better for optimization tasks. While quantum optimization frameworks can frame problems like Max-Cut for quantum solutions, practical execution depends on qubit needs, circuit depth, simulator or device latency, and queue state (Nimbe et al., 2022). Qiskit Optimization supports Max-Cut as a graph-based optimization and helps build quantum workflows from NetworkX graphs, making it useful for studying task placement in a hybrid setting (Esposito & Haus, 2025).

These insights encourage the use of machine learning in hybrid scheduling. Unlike fixed rules, a learned scheduler can evaluate workload features and system states. It can then decide when to use classical execution versus quantum offloading. This is crucial in environments with variable queue pressure, diverse setup costs, and different task structures. A learning-based scheduler can adapt to the interplay between graph complexity, execution behavior, and changing system conditions (Chen et al., 2019).

In this work, we define hybrid classical-quantum scheduling as a supervised task-placement problem. We use real workloads from the Wiki-Vote network, a directed social voting graph that can be turned into an undirected graph for optimization experiments. From this graph, we sample connected subgraphs of different sizes and treat them as Max-Cut task instances. Each task is profiled under classical execution and a Qiskit-based quantum workflow. Then, we introduce heterogeneous queue conditions to create a scheduler dataset with CPU- and QPU-friendly decisions. We learn the task-placement problem using various machine learning models and evaluate it at both prediction and system-policy levels (Singh & Saha, 2025) results show that our ML-based hybrid scheduler performs better than classical-only, quantum-only, and rule-based hybrids. Specifically, the learned scheduler has the lowest average completion time, smallest makespan, highest throughput, and best decision accuracy among the compared methods. Feature analysis reveals that queue-aware variables are key in decision-making. This suggests that practical hybrid scheduling should focus not only on task type but also on system states. These findings support viewing hybrid classical-quantum resource management as an adaptive scheduling problem rather than a static offloading rule.

This paper makes four main contributions. We start with a hybrid scheduling framework. This framework sees task placement on classical and quantum resources as a machine-learning problem. Next, we create a workload benchmark based on the Wiki-Vote dataset. We generate connected Max-Cut task instances in different sizes. Then, we develop and evaluate an ML-based scheduler. This scheduler predicts if tasks should run on CPU or QPU resources under various conditions. Finally, we compare this scheduler to classical-only, quantum-only, and rule-based hybrid baselines. We use metrics like completion time, makespan, throughput, and decision accuracy for our comparison (Ge et al., 2022)

2. Related Work

The emergence of hybrid classical-quantum computing necessitates a fundamental shift in how computational tasks are managed (Vasile et al., 2015). Traditional scheduling, designed for

homogeneous or mildly heterogeneous classical environments, is increasingly inadequate for systems where resource characteristics such as setup costs, execution latencies, and queue behaviors differ by orders of magnitude (Cao et al., 2025). This chapter reviews the existing literature across four key dimensions: classical and heterogeneous scheduling, machine learning (ML) applications in resource management, quantum optimization frameworks, and the necessity of realistic workload benchmarking.

2.1 Classical and Heterogeneous Task Scheduling

Task scheduling has long been a cornerstone of high-performance and distributed computing. In classical contexts, research has focused on optimizing resource usage and minimizing delays across multicore processors, clusters, and cloud environments. These systems rely on scheduling quality that accounts for workload properties and dynamic system states, such as queue length and processor availability (Mack et al., 2022).

However, as systems have become more heterogeneous incorporating diverse accelerators static heuristic models have begun to struggle. While heuristics perform adequately under stable conditions, they often fail when workloads or system states vary significantly (Bahrani et al., 2024). In hybrid classical-quantum environments, these disparities are even more pronounced. Quantum resources involve unique overheads, including circuit preparation and significant queue delays, which make static allocation rules or fixed routing thresholds ineffective for maximizing system flexibility (Shim, 2025).

2.2 Machine Learning for Adaptive Resource Allocation

To address the limitations of fixed rules, there is a growing body of research focused on learning-based scheduling. Unlike manual heuristics, ML models can capture complex interactions between task-specific features (e.g., graph complexity) and system-level variables (e.g., queue pressure) (Li et al., n.d.).

Recent studies demonstrate that when workload features are observable prior to execution, predictive models can achieve superior resource allocation (Cao et al., 2025). In the context of hybrid systems, ML allows for adaptive decision-making. Rather than assuming quantum execution is always superior, a learned scheduler can evaluate whether a task should remain on a classical CPU or be offloaded to a quantum processing unit (QPU) based on the current system "regime" and expected completion costs (Liu et al., 2022).

2.3 Quantum Optimization and Hybrid Execution Frameworks

The practical study of hybrid scheduling has been facilitated by advancements in quantum programming frameworks (Nimbe et al., 2022). Qiskit Optimization, for instance, has standardized the translation of graph-based optimization problems specifically Max-Cut into forms suitable for variational quantum workflows like QAOA (Javadi-Abhari et al., 2024).

Despite these algorithmic advancements, a significant gap remains in the literature: most research focuses on problem formulation or algorithm execution rather than system-level placement across classical and quantum targets (Herrman et al., 123 C.E.). There is limited exploration of how these quantum workflows behave when integrated into a broader resource-allocation framework that must balance execution quality with resource congestion (Weidenfeller et al., n.d.).

2.4 Workload Realism and Benchmarking

A final critical area of concern in the current literature is workload realism. Many scheduling studies rely on synthetic benchmarks or abstract tasks that may not reflect the patterns found

in real-world applications. This lacks the structural complexity inherent in real network data (Zhou et al., 2020).

To bridge this gap, researchers are increasingly turning to datasets from the Stanford Network Analysis Project (SNAP), such as the Wiki-Vote network (Shehata et al., 2026). By deriving Max-Cut task instances from real interaction graphs, researchers can ground hybrid scheduling experiments in more realistic patterns, ensuring that the resulting models are robust to the structural variations found in actual graph-structured optimization tasks (Vesselinova et al., 2020). Table 1 below shows the summary of literature for this research.

Table 1: Summary of Literature

Author	System & Resources	Role of ML/Qunatum	Workload Type & Graph Use	Key Metrics & Outcomes
(Huang et al., 2022)	CPU–GPU heterogeneous task graph runtime	ML mentioned peripherally; mainly graph-based scheduling runtime	Expressive task graph model with in-graph control flow	Up to 29% faster, $1.5\times$ less memory, $1.9\times$ higher throughput vs oneTBB on ML workloads
(Yu et al., 2022)	Large-scale cloud workflows (Alibaba trace)	Attention-based deep model for performance prediction, not direct scheduling	$\sim 1\text{M}$ real workflows as DAGs; DAG matrices used as input graph structure	85–92% CPU and 94–98% memory prediction accuracy; proposed as basis for better scheduling
(Yeung et al., 2022.)	GPU clusters for DL training	Graph-feature–based prediction of GPU utilization for co-location	Uses DL computation graphs to derive features for GPU utilization prediction	Up to 61.5% $\uparrow$ GPU utilization, 23.7–30.7% $\downarrow$ makespan, 68.3% $\downarrow$ wait time vs other managers
(Śmierzchalski et al., 123 C.E.)	D-Wave hybrid CPU–QPU solver vs classical ILP	Black-box hybrid solver sometimes offloads to QPU; no ML in scheduler	AGV scheduling mapped to QUBO-like form; constraint graph analyzed	Hybrid solver comparable to or better than ILP on practical AGV scenarios (~ 21 AGVs)
(Zhang et al., 2024)	Heterogeneous cloud (virtual/physical devices)	GNN-encoded DAG + Elite PSO for collaborative scheduling	DAG captures complex tasks; GNN encodes task set and resource info	Improves completion time, energy, and load balance vs state-of-the-art schedulers

(Du et al., 2025)	PC clusters for DAG workloads	Heuristic WOA with ML-related baselines in survey; optimizer itself heuristic	Real Alibaba DAG traces; SPD-FTG correlation exploited	Up to 93% ↓ completion time vs seven baseline algorithms
(Luo et al., 2025)	Distributed QPUs + classical channels	Compares multiple schedulers including RL-based noise-aware quantum job scheduling	Large circuits decomposed; not graph-derived workloads per se	Evaluates runtime, fidelity, communication costs; shows trade-offs and benefits of parallel, noise-aware RL scheduling
(Mali et al., 2025)	Edge for IoT (edge servers, UEs)	BiLSTM-GRU-Attention predictor + D4PG federated RL for offloading & resource allocation	Google cluster traces; time-series resource traces, not explicit graph workloads	Better energy, performance, and power stability than best-fit / first-fit / worst-fit; strong FL accuracy/F1
(Shen et al., 2025)	End-edge-cloud; CPUs, GPUs, NPUs	Q-learning-based scheduler, state encodes task and environment matrices	Real EEC testbed with bursty queues; not explicitly graph-derived workloads	Up to 271% performance improvement in burst scenarios vs existing schedulers
(Shankar & Kumari, 2025)	MEC networks; edge/cloud + simulated QPUs	Hybrid quantum-classical neural network for offloading policy, Lyapunov-stabilized	Network offloading, not explicit graph workloads	Up to 30% ↑ throughput, >20% ↓ power vs classical methods

3. Method and Materials

3.1 Research Design

The study adopts an experimental systems design. The central idea is to construct a hybrid scheduling benchmark from a real graph source, profile each task under both classical and quantum execution workflows, and then train a classifier to predict the preferred execution target under heterogeneous queue conditions. The research proceeds in five main stages. First, the Wiki-Vote dataset is loaded and preprocessed into an undirected working graph. Second, connected subgraphs are sampled to produce Max-Cut task instances. Third, each task is profiled under classical and quantum execution paths. Fourth, queue-state conditions are

introduced to generate CPU- and QPU-favorable cases and build the scheduler dataset. Fifth, machine-learning models are trained and evaluated against baseline policies.

3.3 System Model

The system considered in this study consists of two execution targets: a classical CPU resource and a quantum processing resource represented by a Qiskit-based simulated execution path. Each incoming task must be assigned to exactly one of these resources. The CPU pathway has conventional execution cost and relatively low setup overhead, while the QPU pathway includes circuit construction, simulator execution, and post-processing cost. For each task i , the total completion cost on a resource r is defined as the sum of queue delay, setup overhead, execution time, and post-processing time:

$$C_i^{(r)} = T_{i,queue}^{(r)} + T_{i,setup}^{(r)} + T_{i,exec}^{(r)} + T_{i,post}^{(r)}$$

The preferred execution target is defined by comparing the total cost on the CPU and QPU pathways. If the classical cost is lower, the task is labeled CPU; otherwise, it is labeled QPU.

3.4 Architecture of the Proposed Hybrid Scheduling Framework.

The architecture as shown in Figure 1 below shows five stages:

- Task Input
- Feature Extraction
- Scheduler Decision
- Execution on CPU or QPU, and
- Feedback Logging for Evaluation.

The proposed architecture is a hybrid classical–quantum scheduling framework designed to intelligently allocate graph-based computational jobs between classical CPUs and quantum processing units (QPUs). The system begins with a Task Arrival Pool where incoming graph-oriented tasks are parsed, validated, and transformed into structured task objects during preprocessing. A Feature Extraction Layer then derives task-level, graph-level, quantum circuit, and system-state features, which are passed into a Hybrid Scheduling Core consisting of a rule-based baseline scheduler and an ML-based decision engine that determines the optimal execution resource. Based on the scheduler’s decision, tasks are executed either on a classical CPU using exact or heuristic methods or on a quantum engine such as Qiskit/Aer running QAOA workflows. The Execution Monitor and Result Logger continuously track runtime, queue state, waiting time, cut quality, and selected resources, while the Evaluation and Feedback Layer analyze system-wide metrics such as makespan, throughput, scheduler accuracy, and feature importance to support adaptive retraining and continuous optimization of the scheduling model.

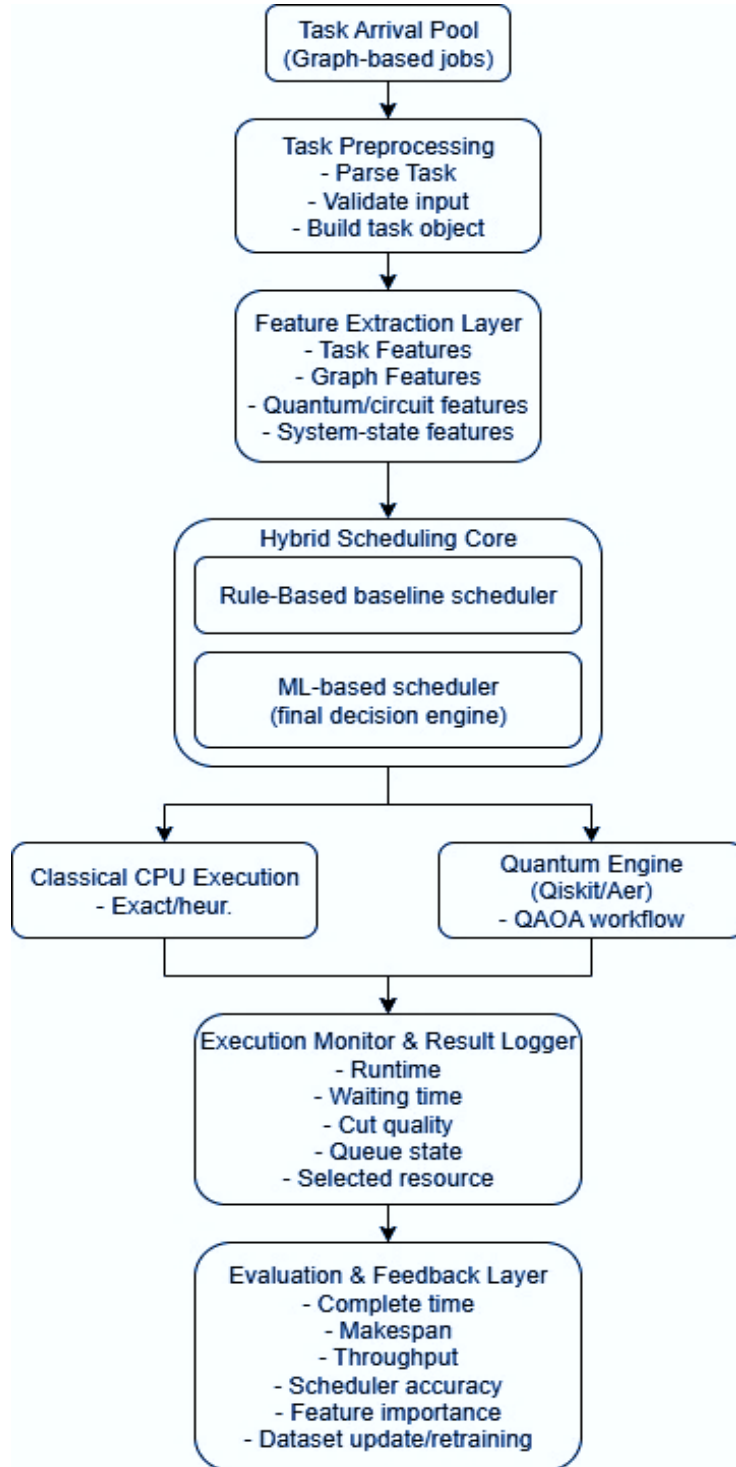


Figure 1. System Architecture

3.5 Mathematical Modelling and Verification of System Architecture

Below is a formalization that models the system as a hybrid classical–quantum scheduling optimization problem:

$$J = \{j_1, j_2, \dots, j_n\}$$

where each job has features such as graph size, density, circuit depth, queue state, expected runtime, and hardware availability. The scheduler learns a decision function:

$$f_{\theta}(x_j) \rightarrow r_j, r_j \in \{CPU, QPU\}$$

where x_j is the extracted feature vector and r_j is the selected execution resource.

The optimization objective may be written as:

$$\min_{\pi} \alpha T_{completion} + \beta W_{queue} + \gamma C_{cost} - \delta Q_{solution}$$

where the scheduler seeks to minimize completion time, queue waiting time, and computational cost while maximizing solution quality.

Verification can then be done through:

1. Correctness verification

Ensure that every submitted task is mapped to exactly one valid execution resource:

$$\forall j_i \in J, \exists! r_i \in \{CPU, QPU\}$$

where:

- J = set of submitted tasks
- r_i = assigned execution resource

2. Performance verification

Verify that the hybrid scheduler improves execution efficiency relative to a baseline scheduler:

$$M_{hybrid} < M_{baseline}$$

$$W_{hybrid} < W_{baseline}$$

$$Th_{hybrid} < Th_{baseline}$$

where:

- M = makespan
- W = average waiting time
- Th = throughput

The optimization objective may be expressed as:

$$\min_{f_{\theta}} \sum_{i=1}^n (\alpha T_i + \beta W_i + \gamma C_i - \delta Q_i)$$

3. Feasibility verification — ensure quantum tasks satisfy QPU constraints such as qubit number, circuit depth, gate compatibility, and queue availability.

Ensure that quantum tasks satisfy QPU hardware constraints:

$$r_i = QPU \Rightarrow \left\{ \begin{array}{l} q_i \leq Q_{max} \\ d_i \leq D_{max} \\ g_i \in G_{supported} \\ a_i = 1 \end{array} \right\}$$

where:

- q_i = required qubits
- d_i = circuit depth
- g_i = gate set
- a_i = queue availability state

4. Stability verification

Ensure that small perturbations in task features do not produce unstable scheduling decisions:

$$\|x_i - x'_i\| < \epsilon \Rightarrow f_\theta(x_i) = f_\theta(x'_i)$$

or more generally:

$$P(f_\theta(x_i) \neq f_\theta(x'_i)) < \tau$$

where:

- x_i and x'_i are nearby feature vectors
- ϵ = perturbation threshold
- τ = acceptable instability probability

5. Learning verification

Validate the ML scheduler using predictive and generalization metrics.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1\ Score = \frac{2TP}{2TP + FP + FN}$$

where:

- TP, TN, FP, FN are classification statistics

The proposed hybrid classical–quantum scheduling architecture can be formally modelled as a constrained multi-objective optimization system, where task-resource assignment is governed by feature-driven decision functions and verified against correctness, feasibility, efficiency, and learning-based performance criteria.

3.6 Dataset Description and Preprocessing

The source dataset used in the study is the Wiki-Vote graph. From the file header and dataset description, the original graph contains 7,115 nodes and 103,689 directed edges. Because Max-Cut is defined on undirected graphs, the dataset was converted into an undirected working representation. Self-loops were removed, and the largest connected component was retained in order to support stable connected-subgraph sampling. After preprocessing, the working graph contained 7,066 nodes and 100,736 edges. These values formed the basis of the task-generation process.

3.7 Graph-Derived Task Construction

To create a scheduling workload from the source graph, connected subgraphs were sampled from the preprocessed working graph. A breadth-first-neighborhood expansion strategy was used to build connected candidate sets, from which induced subgraphs of fixed size were extracted. Only connected subgraphs with at least one edge were retained.

Task sizes of 6, 8, 10, 12, and 14 nodes were used, and 40 tasks were generated for each size, yielding a total of 200 Max-Cut task instances. For each task, graph features were computed, including number of nodes, number of edges, graph density, average degree, maximum degree, minimum degree, average clustering coefficient, and number of connected components. Derived execution-oriented variables such as estimated depth, estimated gate count, and qubit requirement were also included. Figure 2 below shows a workflow of the Graph-Derived Task Construction and Figure 3 shows a Sampled Subgraph.

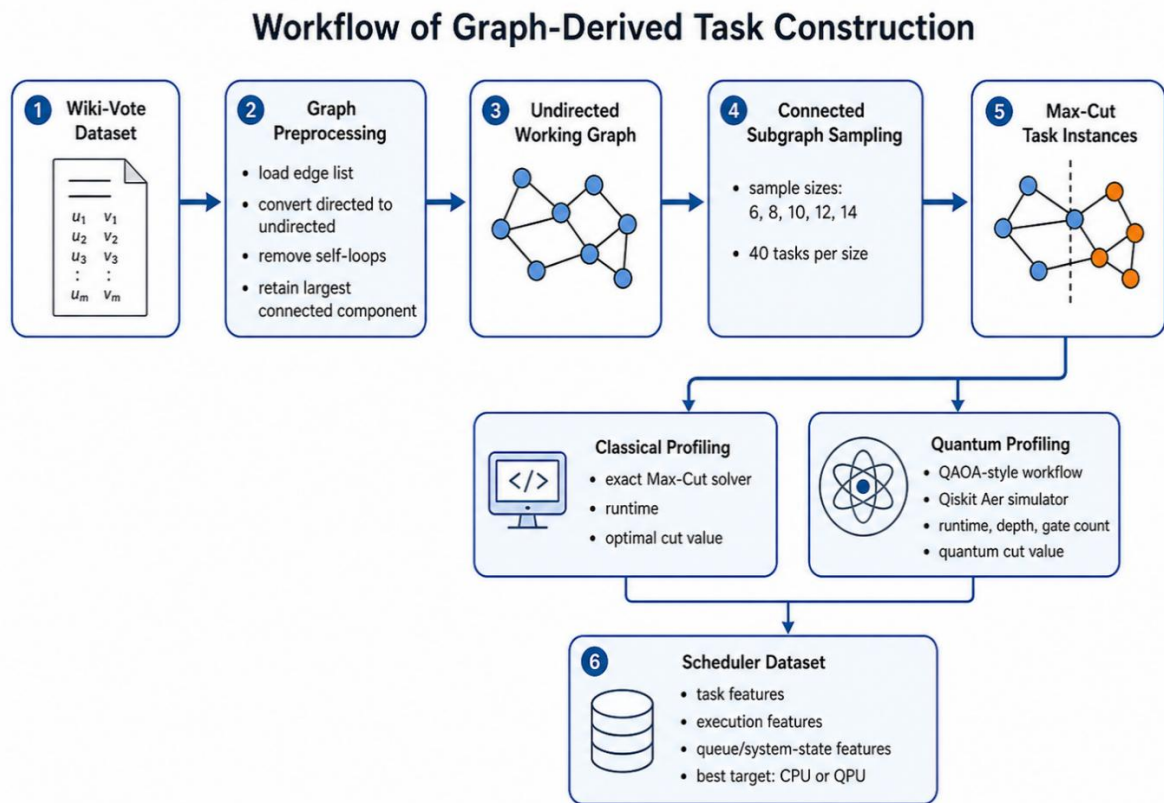


Figure 2. Workflow Diagram

Example sampled subgraph used as a Max-Cut task.

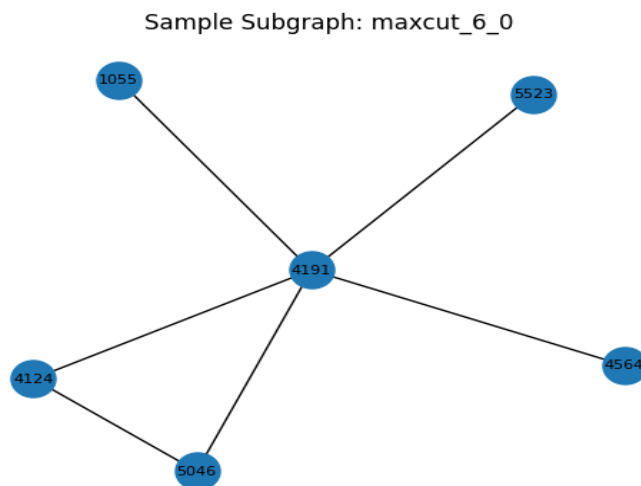


Figure 3. Sampled Subgraph

3.8 Mathematical Modelling and Verification of the Graph-Derived Task Construction

It can be formalized as:

$$G = (V, E) \rightarrow G' = (V', E') \rightarrow \{G1, G2, \dots, Gn\}$$

where the original Wiki-Vote graph is preprocessed into an undirected connected graph, then sampled into connected subgraphs of sizes 6,8,10,12,146,8,10,12,146,8,10,12,14. Each sampled subgraph becomes a Max-Cut task instance:

$$\max_{S \subseteq V_i} |\{(u, v) \in E_i : u \in S, v \notin S\}|$$

The classical profiler produces exact runtime and optimal cut values, while the quantum profiler produces QAOA-based runtime, circuit depth, gate count, and approximate cut values. Verification can then check that preprocessing preserves graph validity, each sampled subgraph is connected, each Max-Cut instance is well-defined, classical optimal values are correct, QAOA outputs are feasible cuts, and the final scheduler dataset correctly assigns the best execution target, CPU or QPU, using measurable criteria such as runtime, solution quality, and queue state.

3.9 Definition of task and system features.

Table 2 below defines the variables used by the scheduler, including graph features, circuit features, queue features, and regime indicators. Table 3 shows the workload construction summary.

Table 2: Task and system features

Feature Group	Variables	Description
Task metadata	task_id, task_type	Identifies the task instance and workload type.
Graph structure	num_nodes, num_edges, graph_density, avg_degree, max_degree, min_degree, avg_clustering, num_components	Describes the structural properties of the sampled subgraph.
Optimization and circuit descriptors	num_variables, num_qubits_required, estimated_depth, estimated_gate_count, quantum_depth, quantum_gate_count, gates_per_edge, depth_per_qubit	Captures problem size and quantum circuit complexity.
Execution behavior	classical_runtime_sec, quantum_runtime_sec, runtime_ratio	Represents measured execution cost on CPU and QPU paths.
Solution quality	classical_cut_value, quantum_cut_value, quantum_approx_ratio, cut_gap, quantum_matches_optimum	Measures the quality of classical and quantum solutions.
Queue and overhead state	queue_cpu_sec, queue_qpu_sec, queue_ratio, cpu_setup_sec, cpu_post_sec, qpu_setup_sec, qpu_post_sec	Captures resource congestion and execution overhead.

Scheduling context	priority, system_regime	Encodes task urgency and global operating condition.
Labeling variables	cpu_total_cost, qpu_total_cost	Used to determine the lower-cost execution path; excluded from model training.
Prediction target	best_target	Indicates whether CPU or QPU is the preferred resource.

Table 3: Workload construction summary.

Item	Value
Original Wiki-Vote nodes	7,115
Original Wiki-Vote edges	103,689
Working undirected graph nodes	7,066
Working undirected graph edges	100,736
Number of sampled tasks	200
Task sizes used	6, 8, 10, 12, 14
Tasks per size	40
Quantum shots	256
QAOA depth p	1

3.9 Classical Profiling

Each task instance was solved classically using an exact brute-force Max-Cut solver. Because the task sizes were intentionally kept small, exhaustive enumeration remained feasible and produced an exact cut value for every graph. The solver also recorded runtime and the corresponding partition assignment. The classical profiling stage established the reference solution quality for each task and served as the baseline against which the quantum workflow was later compared.

3.10 Quantum Profiling

The quantum execution path was implemented using a Qiskit-based QAOA-style circuit on a simulator. For each task instance, a shallow circuit with QAOA depth $p = 1$ was constructed, and a small grid search was performed over the variational parameters. The best observed cut value from the sampled measurement outcomes was recorded for each task together with runtime, circuit depth, gate count, and best parameter values. Qiskit Aer was used for simulation. The experiments used 256 shots per circuit evaluation. Although this workflow was not intended to establish quantum superiority on small graphs, it provided a realistic quantum-style execution path with nontrivial overhead and high-quality approximate solutions.

3.11 Scheduler Dataset Construction

After classical and quantum profiling, the results were merged into a unified scheduler dataset. Each row represented a task and included structural graph features, execution-profile variables, and queue-state descriptors. To avoid a trivial CPU-only dataset, heterogeneous queue-state regimes were simulated. Three regimes were introduced:

1. CPU-favorable regime
2. Balanced regime
3. QPU-favorable regime

These regimes altered queue delays and setup conditions in order to create realistic placement diversity. Under the resulting dataset, 160 tasks were labeled CPU and 40 tasks were labeled QPU, giving an 80:20 class distribution.

3.12 Machine-Learning Scheduler Design

The hybrid scheduler was formulated as a binary classification problem. The input to the model was a feature vector containing task descriptors and system-state variables, while the output was the preferred execution target, CPU or QPU.

Four machine-learning models were evaluated:

- Logistic Regression
- Decision Tree
- Random Forest
- Gradient Boosting

Numerical variables were imputed and standardized, while categorical variables were one-hot encoded. The data were split into training and test sets using stratified sampling so that the class distribution was preserved.

3.13 Baseline Policies

To assess the value of the learned scheduler, three baseline scheduling policies were also implemented:

1. Classical-only scheduling: all tasks are sent to the CPU.
2. Quantum-only scheduling: all tasks are sent to the QPU.
3. Rule-based hybrid scheduling: task placement follows manually defined queue- and size-based rules.

These policies were compared against the proposed ML-based scheduler at the system level.

3.14 Evaluation Metrics

Two categories of evaluation metrics were used.

At the prediction level, the study used:

- Accuracy
- Precision
- Recall
- F1-score
- Confusion matrix

At the policy level, the study used:

- Average completion time
- Average waiting time
- Makespan
- Throughput
- Decision accuracy
- CPU selection ratio
- QPU selection ratio

3.15 Experimental Configuration

The experiments were implemented in Python using NetworkX, Qiskit, Qiskit Aer, NumPy, pandas, matplotlib, and scikit-learn. The final task pool consisted of 200 graph-derived Max-Cut instances. The train/test split produced 160 training instances and 40 test instances. The details of the experimental configuration are shown in Table 4 below.

Table 4: Experimental configuration

Item	Setting
Dataset	Wiki-Vote
Working graph size	7,066 nodes, 100,736 edges
Task sizes	6, 8, 10, 12, 14
Tasks per size	40
Total number of tasks	200
Classical solver	Exact brute-force Max-Cut solver
Quantum simulator	Qiskit AerSimulator
Shots	256
QAOA depth	1
Train/test split	160 / 40 (80% / 20%)
Scheduler policies compared	Classical-only; Quantum-only; Rule-based Hybrid

4 Results and Discussion

4.1 Classical and Quantum Profiling Results

The classical profiling stage showed that exact classical solving remained very fast on the sampled task instances. Across the 200 tasks, the mean classical runtime was approximately 0.0371 seconds, with very small runtimes on the smallest subgraphs and higher, but still modest, runtime on the larger ones.

By contrast, the quantum profiling stage produced a mean runtime of approximately 3.5902 seconds. This gap reflects the additional overhead associated with circuit construction, parameter exploration, simulator execution, and repeated shot-based measurement. On small graph instances, such overhead naturally dominates the overall quantum execution cost.

Despite this difference in runtime, the quality of the quantum solutions was very strong. The mean quantum approximation ratio relative to the classical optimum was approximately 0.9981, and all 200 quantum executions completed successfully. This means that the quantum workflow almost always produced solutions at or near the exact classical optimum. From a

scheduling perspective, this is an important finding because it shows that QPU execution was competitive in solution quality even when it was not competitive in raw latency.

4.2 Classical and quantum profiling summary

Table 5: Classical profiling summary

Metric	Classical	Quantum
Number of tasks	200.000000	200.000000
Mean runtime (sec)	0.023315	3.790335
Std. runtime (sec)	0.029918	0.770929
Minimum runtime (sec)	0.000199	3.221956
Maximum runtime (sec)	0.103951	10.516959
Mean cut value	12.755000	12.725000
Std. cut value	5.803739	5.746694
Minimum cut value	5.000000	5.000000
Maximum cut value	27.000000	27.000000

The results shown in Table 5 indicate that raw execution time alone is not sufficient to define the hybrid scheduling problem. If the study had relied only on measured classical and quantum runtimes, the scheduler would have collapsed into a trivial CPU-only policy. The high solution quality of the quantum workflow, however, indicates that the QPU path remained meaningful and could become advantageous under suitable queue conditions. This justified the use of heterogeneous system regimes in the later task-placement dataset.

4.3 Scheduler Learning Performance

Four machine-learning models were evaluated on the scheduler dataset: Logistic Regression, Decision Tree, Random Forest, and Gradient Boosting. Logistic Regression produced the best results, achieving an accuracy of 0.975, a QPU precision of 0.8889, a QPU recall of 1.0000, and a QPU F1-score of 0.9412 on the held-out test set.

Decision Tree, Random Forest, and Gradient Boosting each achieved an accuracy of 0.9500 and a QPU F1-score of 0.8889. Although these results were still strong, Logistic Regression provided the best balance of overall accuracy and minority-class recognition.

4.4 Machine learning model comparison

The study used two classes of evaluation metrics. At the prediction level, the scheduler models were evaluated using accuracy, precision, recall, F1-score, and confusion matrices. The comparison of models is shown in Table 6 below.

Table 6: Model comparison

Model	Accuracy	Precision QPU	Recall QPU	F1 QPU
Logistic Regression	0.985	0.888889	1.0	0.941176
Decision Tree	0.975	0.888889	1.0	0.941176
Random Forest	0.975	0.888889	1.0	0.941176
Gradient Boosting	0.950	0.800000	1.0	0.888889

4.5 Confusion matrix of the best ML scheduler

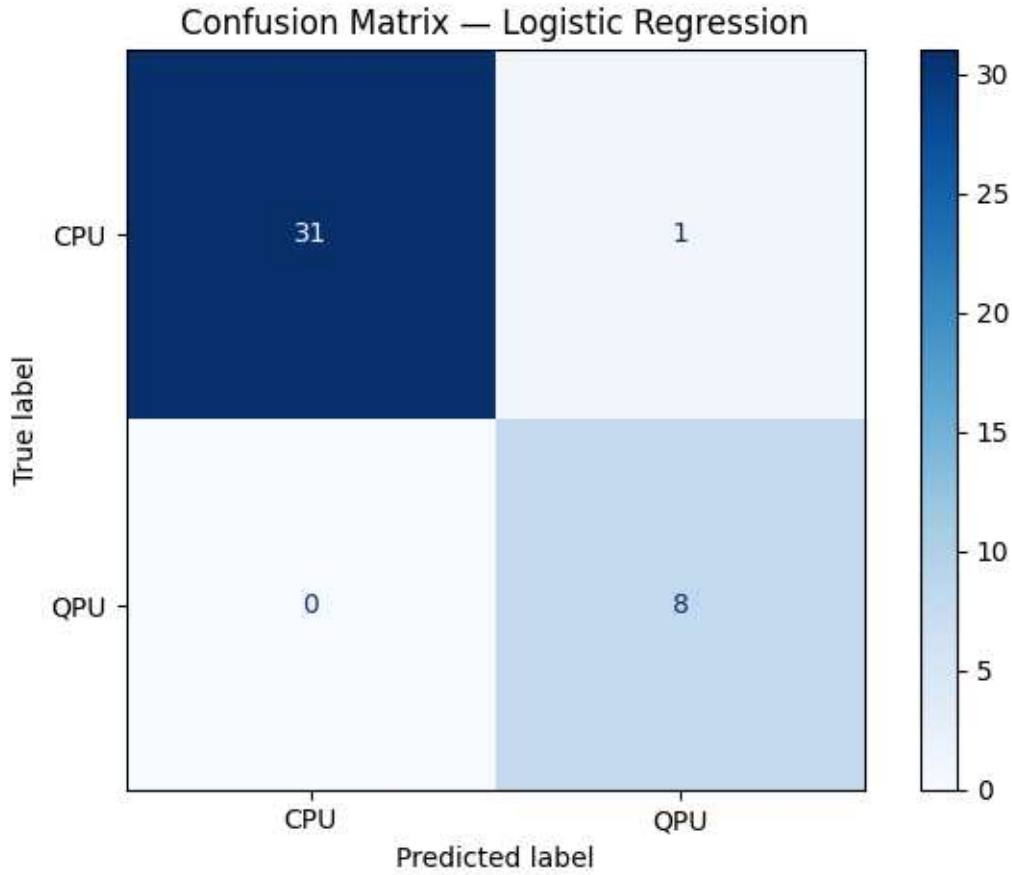


Figure 4. Confusion matrix

The strong performance of Logistic Regression suggests that the CPU/QPU decision boundary in the feature space was relatively well structured. This is a useful result because it shows that hybrid task placement in the constructed workload does not necessarily require overly complex models. When task-structure features and system-state descriptors are chosen carefully, even a relatively simple classifier can perform very well.

The confusion matrix as highlighted in Figure 4 above showed that the best model correctly classified all QPU-favorable tasks in the test set and made only one CPU-to-QPU error. This behavior is desirable in a hybrid scheduler because missing QPU-favorable cases would reduce the value of intelligent offloading, while too many false QPU assignments would increase total completion cost.

4.6 Policy-Level Scheduling Comparison

The most important evaluation in the study compared four scheduling policies on the held-out test set:

- Classical-only
- Quantum-only
- Rule-based hybrid

- ML-based hybrid

The ML-based hybrid scheduler achieved the best overall performance. It obtained a decision accuracy of 0.975, an average completion time of 1.7029 seconds, an average waiting time of 0.8481 seconds, a makespan of 68.1144 seconds, and a throughput of 0.5872 tasks per second. It also selected the CPU for 77.5% of tasks and the QPU for 22.5% of tasks.

The classical-only policy performed second best, with a decision accuracy of 0.800 and an average completion time of 1.9979 seconds. The rule-based hybrid policy performed worse than both the ML-based hybrid and classical-only strategies, while the quantum-only policy performed worst overall, with a decision accuracy of 0.200 and an average completion time of 5.3820 seconds.

4.7 System-level comparison of scheduling policies

At the system-policy level, four scheduling policies were compared using average completion time, average waiting time, makespan, throughput, and decision accuracy and is shown in Table 7 below.

Table 7: System-level comparison of scheduling policies

Policy	Decision Accuracy	Average Completion Time	Average Waiting Time	Makespan	Throughput	CPU Selection Ratio	QPU Selection Ratio
ML-based Hybrid	0.975	1.730227	0.856562	69.209088	0.577959	0.775	0.225
Classical-only	0.800	1.971557	1.904385	78.862287	0.507213	1.000	0.000
Rule-based Hybrid	0.775	2.198565	0.478968	87.942600	0.454842	0.575	0.425
Quantum-only	0.200	5.393637	1.622656	215.745486	0.185404	0.000	1.000

Figures 6, 7 and 8 below shows Policy comparison in average completion time, Policy comparison in makespan and Policy comparison in throughput respectively.

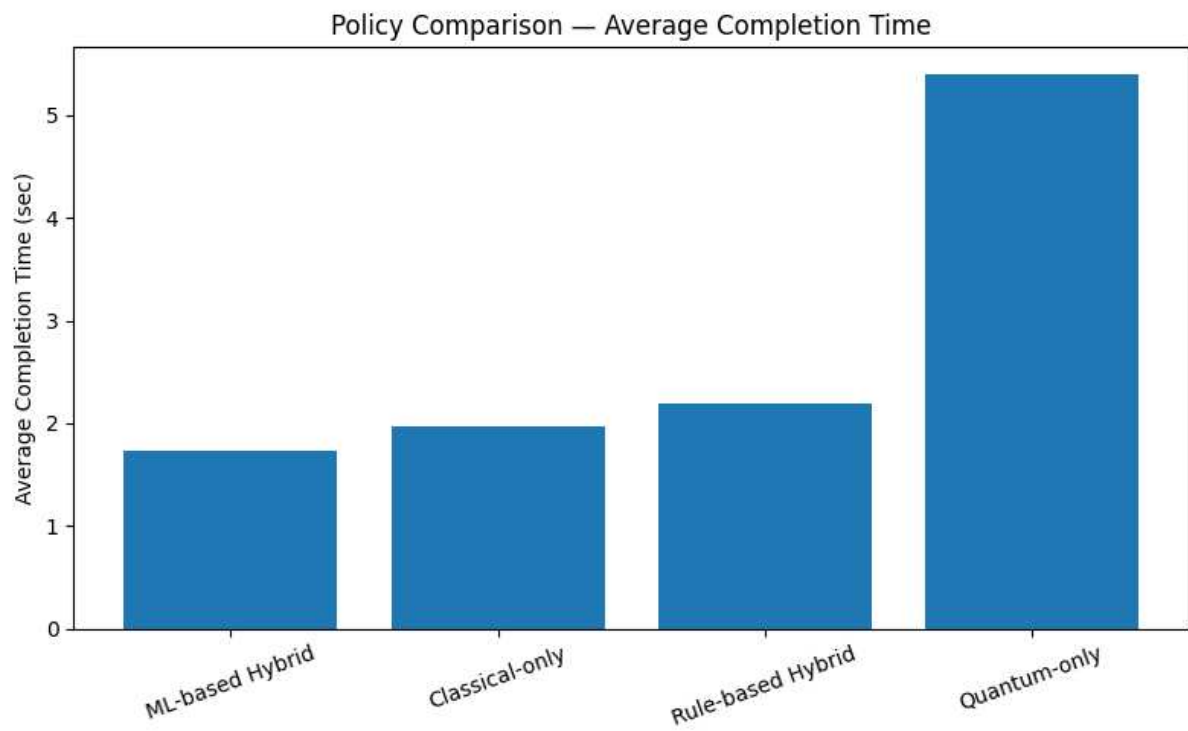


Figure 5. Policy comparison in average completion time.

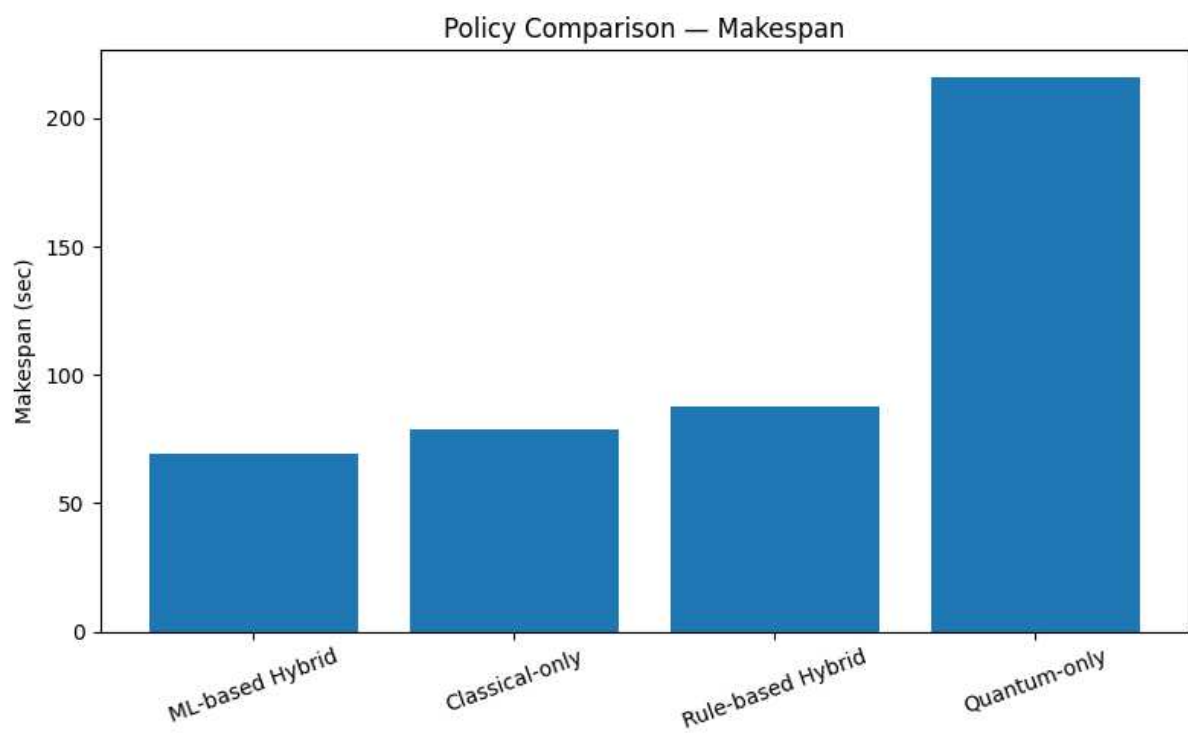


Figure 6. Policy comparison in makespan.

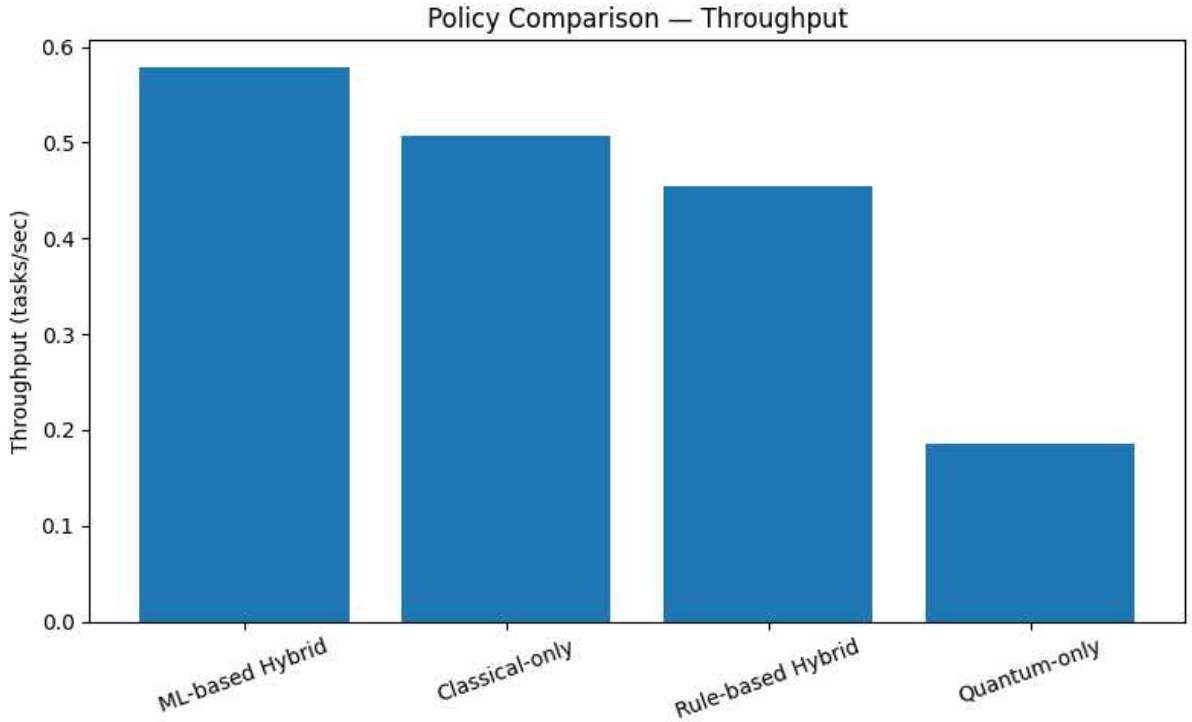


Figure 7. Policy comparison in throughput.

These results confirm that indiscriminate quantum offloading is not appropriate in the present setting. The quantum-only policy performed poorly because the simulated QPU path carried high overhead on small graphs. The classical-only policy performed reasonably well because most tasks were CPU-favorable. However, it still failed to exploit the subset of tasks that became QPU-favorable under certain queue regimes. The rule-based hybrid scheduler improved over the quantum-only baseline but remained inferior to the learned scheduler. This indicates that fixed heuristics were too rigid to capture the variability introduced by system regime, queue pressure, and workload structure. By contrast, the ML-based scheduler adapted to these interactions and translated that adaptability into better system-level outcomes.

4.8 Feature Importance and Scheduling Behavior

Feature-importance analysis revealed that the most influential variables in the best model were:

1. `queue_cpu_sec`
2. `system_regime`
3. `priority`
4. `queue_qpu_sec`
5. `gates_per_edge`
6. `max_degree`
7. `num_edges`
8. `estimated_gate_count`
9. `runtime_ratio`
10. `avg_degree`

Among these, `queue_cpu_sec` had by far the largest influence, followed by `system_regime` and queue-related variables as shown in Figure 8 below. This indicates that the scheduler relied primarily on dynamic system-state conditions when making placement decisions, while still using graph- and circuit-related complexity variables as secondary refinements. Figures 9 and 10 highlights the policy decision distribution across CPU and QPU selections, and CPU vs QPU total cost on test tasks respectively.

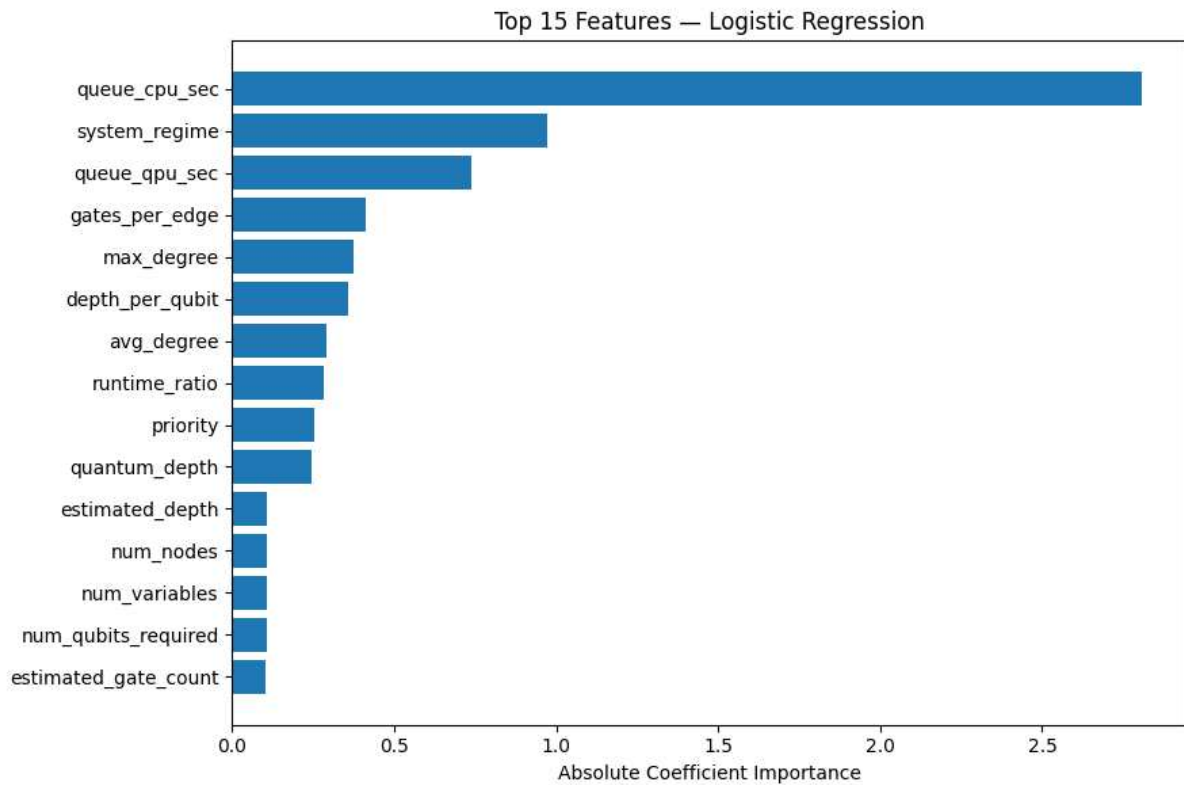


Figure 8. Top feature importance of the best ML scheduler.

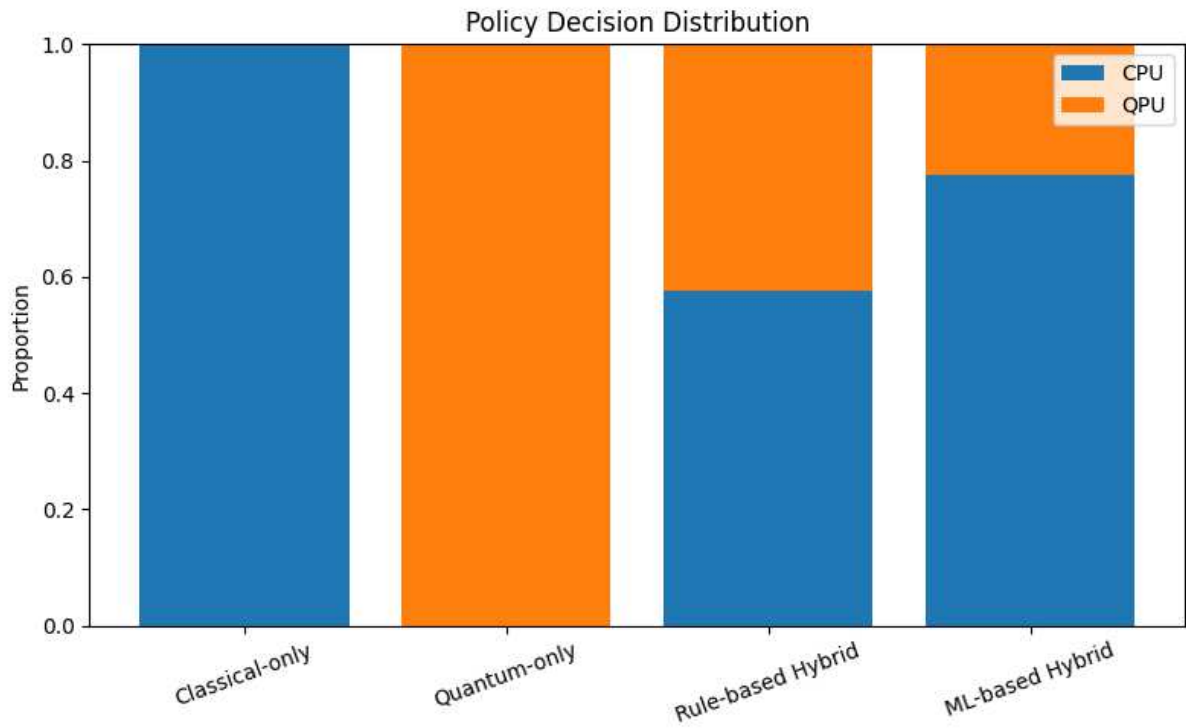


Figure 9. Policy decision distribution across CPU and QPU selections.

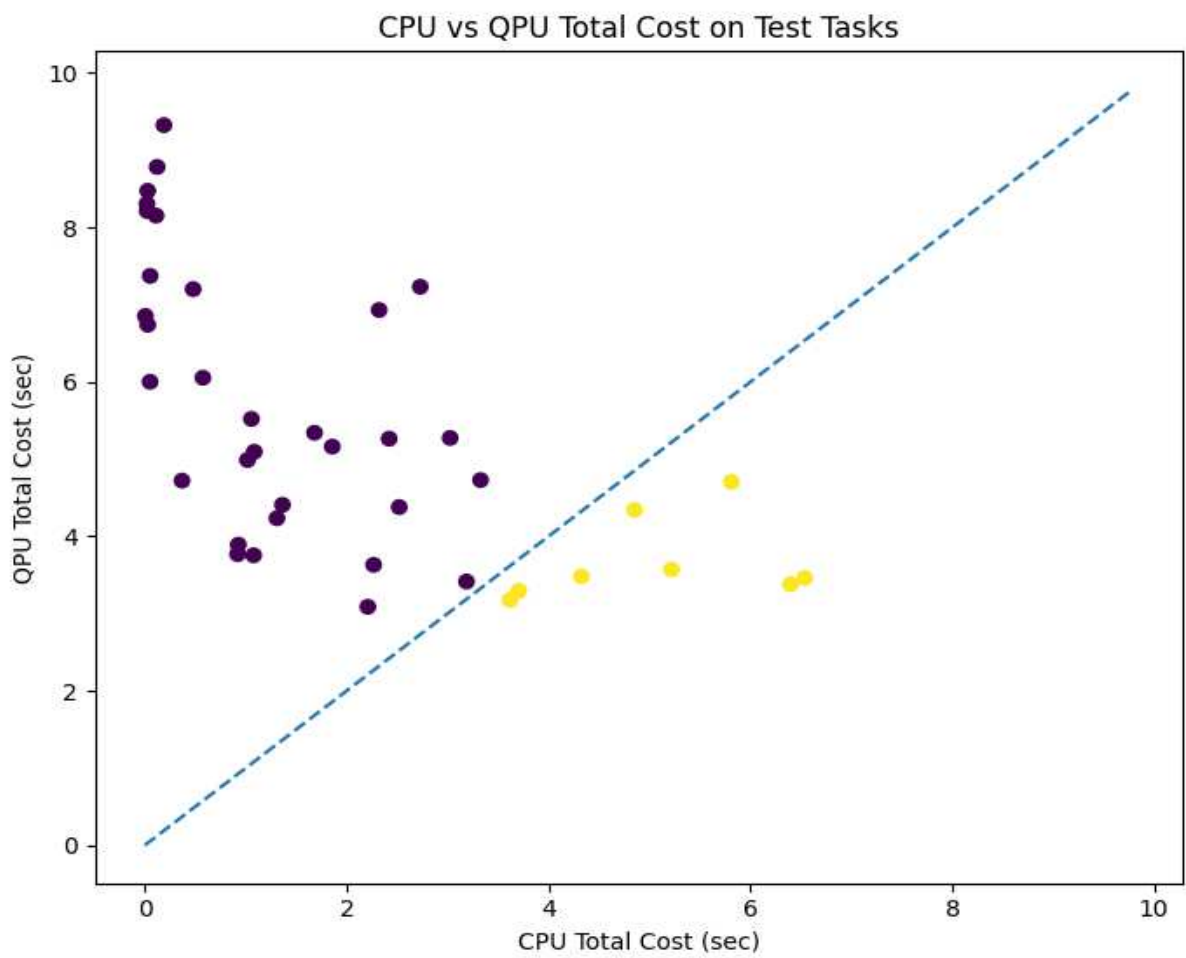


Figure 10. CPU vs QPU total cost on test tasks.

Table 8 below shows the top influential scheduling features.

Table 8: Top influential scheduling features.

Feature	Importance
queue_cpu_sec	2.809104
system_regime	0.973737
queue_gpu_sec	0.738524
gates_per_edge	0.413276
max_degree	0.373624
depth_per_qubit	0.357757
avg_degree	0.289529
runtime_ratio	0.284382
Priority	0.253644
quantum_depth	0.243718
estimated_depth	0.106421
num_nodes	0.106421
num_variables	0.106421
num_qubits_required	0.106421
estimated_gate_count	0.103140

The dominance of queue-aware variables has an important implication: hybrid scheduling in practical environments is likely to depend more on real-time system conditions than on static beliefs about problem suitability. A task that is structurally compatible with quantum execution may still be better assigned to the CPU if the QPU is congested. Conversely, quantum placement may become beneficial when CPU queue pressure becomes large enough to offset the overhead of quantum execution.

At the same time, graph and circuit features still contributed meaningfully to the decision process. Variables such as number of edges, maximum degree, and estimated gate count helped the scheduler distinguish among tasks with different structural complexity. The learned policy therefore combined resource-awareness and workload-awareness, which is exactly the desired behavior in a hybrid environment.

4.9 Task-Size Analysis

The distribution of preferred execution targets across task sizes showed that task size alone did not completely determine placement. Although smaller tasks tended to remain CPU-favorable more often, the final decision still depended strongly on queue regime and system-state conditions. This confirms that simple size-based thresholds would not have been sufficient to produce optimal scheduling behavior. Tables 9 and 10 shows the best target distribution by task size and per-size target distribution respectively.

Table 9: Best target distribution by task size

num_nodes	CPU	QPU
6	35	5
8	31	9
10	33	7
12	33	7

14	34	6
----	----	---

Table 10: Per-size target distribution

Task Size	CPU Count	QPU Count	Total Tasks	CPU (%)	QPU (%)
6	35	5	40	87.5	12.5
8	31	9	40	77.5	22.5
10	33	7	40	82.5	17.5
12	33	7	40	82.5	17.5
14	34	6	40	85.0	15.0

This result reinforces the main argument of the study: hybrid task placement should be treated as an adaptive decision problem rather than as a fixed rule based only on graph size or task category. Even within a relatively narrow workload family such as Max-Cut, placement decisions were shaped by the interaction of size, queue pressure, and system regime.

5. Conclusion

This study presented a machine-learning-based hybrid scheduler for task placement in classical–quantum computing environments. Using the Wiki-Vote graph as a real workload source, the study constructed a benchmark of connected subgraph-based Max-Cut tasks and profiled each task under both classical and quantum execution paths. A scheduler dataset was then built by combining structural task features, execution characteristics, and heterogeneous queue-state conditions.

The results showed that the proposed ML-based hybrid scheduler outperformed classical-only, quantum-only, and rule-based hybrid baselines. Among the evaluated models, Logistic Regression achieved the best predictive performance, with an accuracy of 0.975 and a QPU F1-score of 0.9412. At the system level, the learned scheduler achieved the lowest average completion time, the smallest makespan, the highest throughput, and the highest decision accuracy.

A key conclusion from this study is that hybrid task placement is governed not only by workload structure, but also by queue pressure, system regime, and execution overhead. In the present workload, queue-aware variables were the dominant factors in scheduler behavior. This shows that effective hybrid resource allocation should be adaptive and state-aware rather than based on rigid offloading rules.

Declarations

Funding

No funding was received for conducting this study.

Conflict of Interest:

The authors declare that they have no conflict of interest.

Data Availability

The datasets generated and/or analyzed during the current study are available from the corresponding author upon reasonable request.

Clinical Trial Number

Clinical trial number: Not applicable.

Consent to Publish

Consent to Publish declaration: Not applicable.

Consent to Participate

Consent to Participate declaration: Not applicable.

References

- Bahrani, S., Oliveira, R. D., Parra-Ullauri, J. M., Wang, R., & Simeonidou, D. (2024). *Resource Management and Circuit Scheduling for Distributed Quantum Computing Interconnect Networks*. <https://arxiv.org/pdf/2409.12675>
- Cao, X., Chen, C., Li, S., Lv, C., Li, J., & Wang, J. (2025). Research on computing task scheduling method for distributed heterogeneous parallel systems. *Scientific Reports*, 15(1), 8937. <https://doi.org/10.1038/S41598-025-94068-0>
- Chen, X., Zhang, H., Wu, C., Mao, S., Ji, Y., & Bennis, M. (2019). Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. *IEEE Internet of Things Journal*, 6(3), 4005–4018. <https://doi.org/10.1109/JIOT.2018.2876279>
- Du, N., Ji, Y., Wu, C., Hou, A., & Nie, W. (2025). Scheduling DAG-structured workloads based on whale optimization algorithm: N. Du et al. *SpringerN Du, Y Ji, C Wu, A Hou, W NieThe Journal of Supercomputing*, 2025•Springer, 81(8), 911. <https://doi.org/10.1007/S11227-025-07415-3>
- Esposito, A., & Haus, U.-U. (2025). *SLURM Heterogeneous Jobs for Hybrid Classical-Quantum Workflows*. <https://arxiv.org/pdf/2506.03846>
- Ge, X., Wu, R. B., & Rabitz, H. (2022). The optimization landscape of hybrid quantum–classical algorithms: From quantum control to NISQ applications. *Annual Reviews in Control*, 54, 314–323. <https://doi.org/10.1016/J.ARCONTROL.2022.06.001>
- Herrman, R., Lotshaw, P. C., Ostrowski, J., Humble, T. S., & Siopsis, G. (123 C.E.). Multi-angle quantum approximate optimization algorithm. *Nature.ComR Herrman, PC Lotshaw, J*

Ostrowski, TS Humble, G Siopsis *Scientific Reports*, 2022•nature.Com.
<https://doi.org/10.1038/s41598-022-10555-8>

Huang, T. W., Lin, D. L., Lin, C. X., & Lin, Y. (2022). Taskflow: A Lightweight Parallel and Heterogeneous Task Graph Computing System. *IEEE Transactions on Parallel and Distributed Systems*, 33(6), 1303–1320. <https://doi.org/10.1109/TPDS.2021.3104255>

Javadi-Abhari, A., Treinish, M., Krsulich, K., Wood, C. J., Lishman, J., Gacon, J., Martiel, S., Nation, P. D., Bishop, L. S., Cross, A. W., Johnson, B. R., & Gambetta, J. M. (2024). *Quantum computing with Qiskit*. <https://arxiv.org/pdf/2405.08810>

Li, P., Xiao, Y., Yan, J., Li, X., International, X. W.-2024 5th, & 2024, undefined. (n.d.). Reinforcement learning for adaptive resource scheduling in complex system environments. *Ieeexplore.Ieee.Org* P Li, Y Xiao, J Yan, X Li, X Wang 2024 5th International Symposium on Computer Engineering and, 2024•ieeexplore.Ieee.Org. Retrieved May 5, 2026, from <https://ieeexplore.ieee.org/abstract/document/10810157/>

Liu, R., Piplani, R., & Toro, C. (2022). Deep reinforcement learning for dynamic scheduling of a flexible job shop. *Taylor & Francis* R Liu, R Piplani, C Toro *International Journal of Production Research*, 2022•Taylor & Francis, 60(13), 4049–4069.
<https://doi.org/10.1080/00207543.2022.2058432>

Luo, W., Zhao, J., Zhan, T., & Guan, Q. (2025). Adaptive job scheduling in quantum clouds using reinforcement learning. *Dl.Acm.Org* W Luo, J Zhao, T Zhan, Q Guan *Proceedings of the 54th International Conference on Parallel Processing*, 2025•dl.Acm.Org, 658–667.
<https://doi.org/10.1145/3754598.3754641>

Mack, J., Arda, S. E., Ogras, U. Y., & Akoglu, A. (2022). Performant, Multi-Objective Scheduling of Highly Interleaved Task Graphs on Heterogeneous System on Chip Devices. *IEEE Transactions on Parallel and Distributed Systems*, 33(9), 2148–2162.
<https://doi.org/10.1109/TPDS.2021.3135876>

Mali, S., Zeng, F., Adhikari, D., Ullah, I., Sensors, M. A.-K.-, & 2025, undefined. (n.d.). Federated reinforcement learning-based dynamic resource allocation and task scheduling in edge for IoT applications. *Mdpi.Com* S Mali, F Zeng, D Adhikari, I Ullah, MA Al-Khasawneh, O Alfarraj, F Alblehai *Sensors*, 2025•mdpi.Com. Retrieved May 5, 2026, from <https://www.mdpi.com/1424-8220/25/7/2197>

Nimbe, P., Weyori, B. A., Adekoya, A. F., & Awarayi, N. S. (2022). Implementation of Framework for Quantum-Classical and Classical-Quantum Conversion. *International Journal of Theoretical Physics* 2022 61:2, 61(2), 37-. <https://doi.org/10.1007/S10773-022-04975-5>

Shankar, A., & Kumari, A. (2025). *QoS-aware Scheduling of Periodic Real-time Task Graphs on Heterogeneous Pre-occupied MECs*. <https://arxiv.org/pdf/2506.12415>

Shehata, A., Groszkowski, P., Naughton, T., Gopalakrishnan Meena, M., Wong, E., Claudino, D., Ferreira da Silva, R., & Beck, T. (2026). Bridging paradigms: Designing for HPC-Quantum

convergence. *Future Generation Computer Systems*, 174, 107980.
<https://doi.org/10.1016/J.FUTURE.2025.107980>

Shen, W., Lin, W., Wu, W., Wu, H., & Li, K. (2025). Reinforcement learning-based task scheduling for heterogeneous computing in end-edge-cloud environment. *SpringerW Shen, W Lin, W Wu, H Wu, K LiCluster Computing*, 2025•*Springer*, 28(3).
<https://doi.org/10.1007/S10586-024-04828-2>

Shim, H. (2025). A demand-centered scheduling framework for shared supercomputing resources: modeling, metrics, and case insights. *Scientific Reports 2025 15:1*, 15(1), 17853-.
<https://doi.org/10.1038/s41598-025-02353-9>

Singh, N. B., & Saha, A. K. (2025). Exploring graph cuts in quantum approximate optimization algorithm. *Physica Scripta*, 100(11), 115104. <https://doi.org/10.1088/1402-4896/AE1906>

Śmierzchalski, T., Pawłowski, J., Przybysz, A., Paweła, Ł., Puchała, Z., Koniorczyk, M., Gardas, B., Deffner, S., & Domino, K. (123 C.E.). Hybrid quantum-classical computation for automatic guided vehicles scheduling. *Nature.ComT Śmierzchalski, J Pawłowski, A Przybysz, Ł Paweła, Z Puchała, M Koniorczyk, B GardasScientific Reports*, 2024•*nature.Com*, 14, 21809. <https://doi.org/10.1038/s41598-024-72101-y>

Vasile, M. A., Pop, F., Tutueanu, R. I., Cristea, V., & Kołodziej, J. (2015). Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing. *Future Generation Computer Systems*, 51, 61–71. <https://doi.org/10.1016/J.FUTURE.2014.11.019>

Vesselinova, N., Steinert, R., ... D. P.-R.-I., & 2020, undefined. (n.d.). Learning combinatorial optimization on graphs: A survey with applications to networking. *Ieeexplore.Ieee.OrgN Vesselinova, R Steinert, DF Perez-Ramirez, M BomanIEEE Access*, 2020•*ieeexplore.Ieee.Org*. Retrieved May 5, 2026, from <https://ieeexplore.ieee.org/abstract/document/9125934/>

Weidenfeller, J., Valor, L. C., Gacon, J., Tornow, C., Bello, L., Woerner, S., & Egger, D. J. (n.d.). Scaling of the quantum approximate optimization algorithm on superconducting qubit based hardware. *Quantum-Journal.OrgJ Weidenfeller, LC Valor, J Gacon, C Tornow, L Bello, S Woerner, DJ EggerQuantum*, 2022•*quantum-Journal.Org*. <https://doi.org/10.22331/q-2022-12-07-870>

Yeung, G., Borowiec, D., Yang, R., ... A. F.-... on P. and, & 2021, undefined. (n.d.). Horus: Interference-aware and prediction-based scheduling in deep learning systems. *Ieeexplore.Ieee.OrgG Yeung, D Borowiec, R Yang, A Friday, R Harper, P GarraghanIEEE Transactions on Parallel and Distributed Systems*, 2021•*ieeexplore.Ieee.Org*. Retrieved May 5, 2026, from <https://ieeexplore.ieee.org/abstract/document/9428512/>

Yu, J., Gao, M., Li, Y., Zhang, Z., Ip, W., Industrial, K. Y.-J. of, & 2022, undefined. (n.d.). Workflow performance prediction based on graph structure aware deep attention neural network. *Elsevier*. Retrieved May 6, 2026, from <https://www.sciencedirect.com/science/article/pii/S2452414X22000097>

Zhang, Z., Xu, C., Xu, S., Huang, L., & Zhang, J. (2024). Towards optimized scheduling and allocation of heterogeneous resource via graph-enhanced EPSO algorithm. *SpringerZ Zhang, C Xu, S Xu, L Huang, J ZhangJournal of Cloud Computing, 2024•Springer, 13(1), 108.*
<https://doi.org/10.1186/S13677-024-00670-4>

Zhou, L., Wang, S. T., Choi, S., Pichler, H., & Lukin, M. D. (2020). Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices. *APSL Zhou, ST Wang, S Choi, H Pichler, MD LukinPhysical Review X, 2020•APS, 10(2).*
<https://doi.org/10.1103/PHYSREVVX.10.021067>