Deadline-Aware Scheduling of Distributed Quantum Circuits in Near-Term Quantum Cloud

Nour Dehaini[1], Christia Chahoud[1], and Mahdi Chehimi[1]

[1]American University of Beirut
nad29@mail.aub.edu, cec13@mail.aub.edu, mc127@aub.edu.lb

***Abstract*—Distributed quantum computing (DQC) enables scalable quantum computations by distributing large quantum circuits on multiple quantum processing units (QPUs) in the quantum cloud. In DQC, after partitioning quantum circuits, they must be scheduled and executed on heterogenous QPUs while balancing latency, overhead, QPU communication resource limits. However, since fully functioning quantum communication networks have not been realized yet, near-term quantum clouds will only rely on local operations and classical communication (LOCC) settings between QPUs, without entangled quantum links. Additionally, existing DQC scheduling frameworks do not account for user-defined execution deadlines and adopt inefficient wire cutting techniques. Accordingly, in this work, a *deadline-aware DQC scheduling framework* with efficient wire cutting for near-term quantum cloud is proposed. The proposed framework schedules partitioned quantum subcircuits while accounting for circuit deadlines and QPU capacity limits. It also captures dependencies between partitioned subcircuits and distributes the execution of the sampling shots on different QPUs to have efficient wire cutting and faster execution. In this regard, a deadline-aware circuit scheduling optimization problem is formulated, and solved using simulated annealing. Simulation results show a marked improvement over existing *shot-agnostic* frameworks under urgent deadlines, reaching a 12.8% increase in requests served before their deadlines. Additionally, the proposed framework serves 8.16% more requests, on average, compared to state-of-the-art *dependency-agnostic* baseline frameworks, and by 9.60% versus the *dependency-and-shot-agnostic* baseline, all while achieving a smaller makespan of the DQC execution. Moreover, the proposed framework serves 23.7%, 24.5%, and 25.38% more requests compared to *greedy*, *list scheduling*, and *random* schedulers, respectively.**

***Index Terms*—Distributed quantum computing, deadline-aware scheduling, circuit cutting, shot distribution.**

I. Introduction

Quantum cloud services incorporate several quantum processing units (QPUs) dedicated to serve user-submitted requests to execute quantum circuits for different quantum applications. However, todays noisy, intermediate-scale quantum (NISQ) QPUs have a limited number of noisy qubits and cannot execute deep or large-scale quantum circuits [1]. In this regard, distributed quantum computing (DQC) is a framework that overcomes these limitations by partitioning more complex quantum circuits and executing them on multiple QPUs [2]. Thus, DQC allows the execution of larger quantum circuits than what any single QPU device can execute alone.

This interconnection of QPUs requires the quantum cloud to have quantum communication networks to exchange quantum states between the different devices through entanglement distribution, teleportation, and related protocols [2]. However, these networking capabilities are still at an early stage, limiting the near-term scalability of DQC in cloud-based settings [3]. As a result, recent works have shifted toward exploring *circuit cutting* and *local operations and classical communication* (LOCC) as near-term quantum cloud DQC enablers without requiring quantum communication links between QPUs [1].

In this circuit cutting compiling step of the quantum cloud, large quantum circuits are partitioned into smaller subcircuits that can be executed independently and *classically* recombined [4], [5]. The two most commonly considered techniques are *wire cutting* [4], which cuts qubit wires and replace them with independent local state preparations and measurements, and *gate cutting* [5], which decomposes entangling gates into local operations with probabilistic reconstruction [6]. Both methods enable distributed execution using local operations, though at the cost of sampling overhead that scales exponentially with the number of cuts. The partitioned subcircuits must then be scheduled and executed on the cloud's QPUs while accounting for the user-specific requirements and QPU constraints.

Additionally, to reduce the number of generated subcircuits and reduce the sampling overhead, an LOCC wire-cutting technique was recently developed [7], which requires classical coordination between measurement and preparation subcircuits. However, this technique introduces dependencies between the subcircuits, which makes the task of DQC scheduling more challenging because of the several resulting constraints. When multiple users submit large quantum circuits to be partitioned, distributed, and scheduled across heterogeneous QPUs, the scheduler must balance competing objectives: minimizing execution time, respecting subcircuits dependencies and device constraints, and meeting user-specific deadlines, which makes the design and optimization of the quantum cloud scheduler a challenging task.

A. Related Works

Prior works have examined some of the aforementioned DQC scheduling challenges in isolation, but, to the best of our knowledge, no unified framework was developed to jointly address them in near-term quantum cloud. In general, recent efforts have explored DQC scheduling in quantum cloud settings with quantum communication networks between QPUs [8]–[13]. For instance, the work in [8] investigated how to schedule DQC jobs across QPUs utlizing entangle-

ment distribution, while the work in [9] presented a simulation framework for DQC that integrates computational and quantum networking aspects. Additionally, the work in [10] focused on resource management and circuit scheduling for data-centerscale heterogeneous QPU quantum communication networks. Moreover, in [11], the authors proposed scheduling frameworks for quantum network operations required to support non-local gates in DQC. Furthermore, the work in [12] proposed a framework implementing circuit cutting techniques with entanglement-based distribution on QPUs, and the work in [13] proposed a circuit cutting and mapping framework for large-size circuits, using wire cuts and gate cuts with remote entanglement between nodes to avoid fully independent subcircuits which minimizes the sampling overhead. However, none of the works in [8]–[13] considered near-term quantum cloud with LOCC only, as they all assumed the presence of shared entanglement between QPUs. Additionally, none of these works considered user-specified deadlines for the submitted quantum circuits, which is critical to represent real quantum applications. On top of that, all these works consider the execution of all sampling shots of the partitioned subcircuits on the same QPU, which is not optimized and can result in inefficiencies in DQC execution.

For near-term LOCC-based quantum cloud, recent works considered LOCC circuit cutting to solve the challenges of automating optimal cut placements and reducing sampling overhead [5], [7], [14], but without considering the resulting circuit scheduling problems. Additionally, some recent efforts explored DQC scheduling in these near-term cloud setups [6], [15]–[17]. For instance, the authors in [15] investigated adaptive job scheduling for DQC in cloud environments on a set of QPUs connected via LOCC and compared between four scheduling techniques. In [16], the authors introduced an integer linear programming scheduler in an LOCC-only setting that maps the subcircuits across QPUs, while respecting device-specific runtime limits. Similarly, in [6], the authors provide the Qdislib library for circuit cutting and focus on the parallel execution of quantum subcircuits across CPUs, GPUs, and QPUs via PyCOMPS framework. Nevertheless, none of the works in [6], [15], [16] considered user-definied deadlines nor developed a deadline-aware DQC job scheduling framework. Additionally, the works in [6], [15], [16] consider the execution of all sampling shots of the partitioned subcircuits on the the same QPU, which is not optimized and could result in delays in the DQC job execution time.

In this regard, the work in [17] is the only existing work to address distributing both partitioned circuits and their sampling shots across multiple QPUs, analyzing its effect on minimizing the runtime. However, the shot distribution framework in [17] is not optimized and only allocates the shots uniformly over available QPUs or via user-specified rules, and, again, user-defined deadlines were not considered in [17].

To the best of our knowledge, *no existing work considered DQC scheduling in near-term multi-user quantum clouds based on LOCC only, where users specify the required application-specific execution deadlines for their submitted quantum circuits. Furthermore, no existing framework considered optimizing the subcircuit sampling shot distribution across heterogeneous QPUs to minimize execution time.*

B. Contributions

The main contribution of this work is the first deadline-aware DQC scheduling framework that supports multi-user workloads in a near-term quantum cloud with heterogeneous QPUs under an LOCC-only setting. In particular, we propose the first model that jointly considers circuit deadlines, efficient wire cutting, and sampling shots distribution in DQC scheduling. Towards this goal, we make key contributions:

- We introduce a novel DQC scheduling framework that, unlike state-of-the-art works, adopts the recently-developed shot efficient LOCC wire-cut technique [7] in an LOCC-based near-term quantum cloud. Particularly, we capture dependencies between the measurement and prepare subcircuits and develop the first framework to optimize the subcircuits' scheduling under these realistic constraints.
- We formulate a deadline-aware DQC scheduling optimization framework that allows the distribution and optimization of subcircuits' sampling shots across heterogeneous QPUs, with different maximum possible circuit depths and qubits, while accounting for LOCC wire-cut dependencies and QPU eligibility and capacity limits, with the objective of maximizing the number of served user requests that meet their deadlines.
- We conduct extensive simulations comparing our *proposed* framework with *greedy*, *list*, and *random* schedulers, as well as state-of-the-art *shot-agnostic* and *dependency-agnostic* baselines. Under urgent deadlines, the *proposed* framework serves 12.8% more requests than *shot-agnostic* scheme, and with mixed deadlines, it serves 8.16% and 9.60% more requests than dependency-agnostic and dependency-and-shot-agnostic baselines, while also reducing makespan, and 23.7%, 24.5%, and 25.38% more requests than *greedy*, *list*, and *random* schedulers, respectively.

II. System Model

We consider a near-term quantum cloud, shown in Fig. 1, with a set $\mathcal{M}$ of M heterogeneous QPUs. Each QPU $m \in \mathcal{M}$ is characterized by a qubit capacity Q_m and a maximum feasible circuit depth D_m that maintains acceptable fidelity. The QPUs are interconnected via classical communication links only, reflecting today's near-term capabilities in which quantum networks are not yet mature enough to provide reliable entanglement between QPUs [1], [18].

The quantum cloud serves multiple users who submit a set $\mathcal{U}$ of U quantum computing requests. Each request $i \in \mathcal{U}$ is for running a quantum circuit C_i coming from a certain application (e.g., quantum optimization, search, order finding, sensing, ..., etc). Each circuit C_i of request $i \in \mathcal{U}$ arrives with a user-specified deadline τ_i that specifies the maximum duration by which the circuit must be completed. The submitted circuits

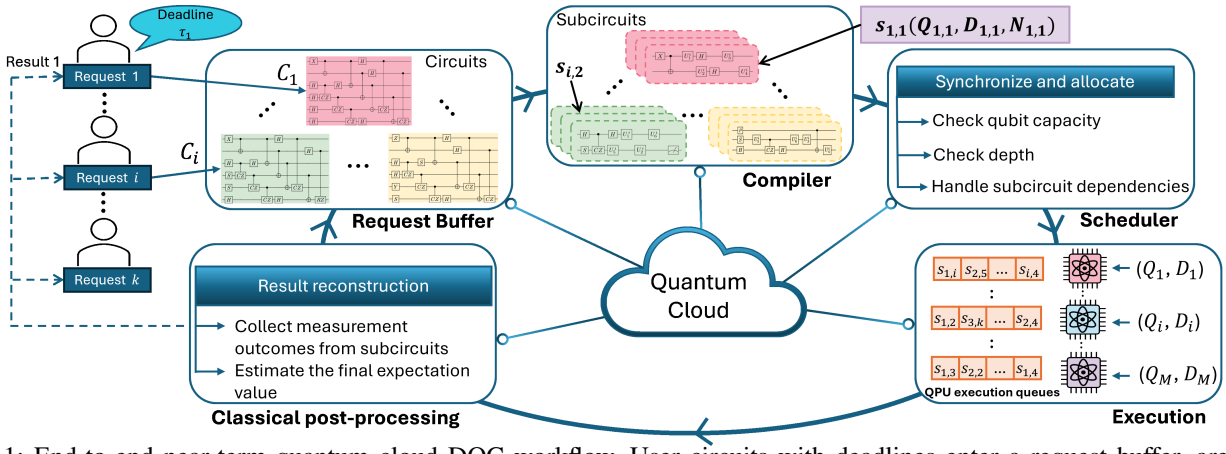


Fig. 1: End-to-end near-term quantum cloud DQC workflow. User circuits with deadlines enter a request buffer, are cut into subcircuits by the compiler, scheduled on eligible QPUs under qubit count, depth and precedence constraints. Then, final results are reconstructed with classical post-processing.

are collected by the request buffer in the quantum cloud where jobs are stored before compilation (Fig. 1) .

A. Quantum Cloud Compiler

The cloud's compiler (Fig. 1) partitions each circuit C_i into subcircuits $\mathcal{S}_i = \{s_{i,1}, ..., s_{i,k}\}$, with $k = |\mathcal{S}_i|$, using wire cuts [7] and gate cuts [5]. This partitioning is performed only on large circuits that cannot be executed on a single QPU, which is the scenario this work focuses on. The compiler ensures that the hardware constraints of the QPUs, such as qubit count and depth capacity, are met for at least one QPU for each $s_{i,j} \in S_i$ to make sure all subcircuits are feasible. Each subcircuit $s_{i,j}$ is described by its qubit demand $Q_{i,j}$, which is the number of qubits needed for the circuit, depth $D_{i,j}$, which is the number of its layers, and a shot count $N_{i,j}$, which is the number of shots, or samples, that need to be run in order to obtain an average result from the circuit.

Specifically, when the compiler performs a gate cut, it replaces a two-qubit gate V through quasiprobability decomposition into a weighted sum of local operations [5]:

$$V = \sum_{r=1}^{R} a_r \left(A_r \otimes B_r\right), \qquad a_r \in \mathbb{R} \tag{1}$$

where each term $r \in \{1, \ldots, R\}$ in the decomposition corresponds to a pair of local single-qubit gates applied on two separate partitions, resulting in a total of $2R$ subcircuits. These subcircuits are executed independently, and their outcomes are classically recombined at the last stage of circuit cutting workflow, as shown in Fig. 1, using the quasiprobability weights $\{a_r\}$ to estimate the expectation value of the target observable. Because quantum measurements are intrinsically probabilistic, observables are estimated from repeated, independent runs, called "shots". However, gate cutting increases estimator variance, causing a sampling overhead where the required number of shots is multiplied by $(\sum_{r\in R} |a_r|)^2$ to reach the same accuracy as executing the nonlocal gate directly.

Similarly, when the compiler performs a wire cut, it replaces a qubit wire with a linear combination of measure-prepare subcircuits [4]:

$$\mathcal{I} = \sum_{b\in\Lambda} c_b \left(\mathcal{P}_b \circ \mathcal{M}_b\right), \qquad c_b \in \mathbb{R} \tag{2}$$

where $\mathcal{I}$ is the single-qubit identity channel, $\mathcal{M}_b$ is a measurement operator in basis $b \in \Lambda$ with $\Lambda = \{I, X, Y, Z\}$, $\mathcal{P}_b$ is the preparation channel that initializes the quantum state in the same basis b, and c_b are the quasiprobability decomposition weights. As with gate cuts, outcomes from the induced subcircuits are recombined using the quasiprobability weights $\{c_b\}$ and the shot overhead increases by $\left(\sum_b |c_b|\right)^2$.

To reduce the number of subcircuits and sampling overhead in the compiling process, we adopt the recently developed shot-efficient LOCC wire cutting technique [7]. Reducing the sampling overhead decreases the number of shots that must be executed to obtain a satisfying accuracy of the executed subcircuit. However, this induces a dependency in the execution since the measurement subcircuit must be executed before its corresponding preparation subcircuit. This is because the preparation subcircuit takes the classical measurement outcome as input. We represent the execution dependencies between the subcircuits by a precedence directed acyclic graph (DAG) $(\mathcal{S}_i, \mathcal{P}_i)$, where the vertices are the subcircuits S_i of circuit C_i and the edge set is $\mathcal{P}_i \subseteq \mathcal{S}_i \times \mathcal{S}_i$ defining the precedence relationships between subcircuits. We write $s_{i,u} \to s_{i,v} \in \mathcal{P}_i$ when $s_{i,v}$ must be executed after $s_{i,u}$.

B. Quantum Cloud Scheduler and Post-processing

Thus, the compiler partitions the circuits in the request buffer using gate cuts and LOCC wire cuts. As shown in Fig. 1, the cloud's scheduler then receives from the compiler, for each circuit C_i, the subcircuits $s_{i,j}$ with parameters $(Q_{i,j}, D_{i,j}, N_{i,j})$. The scheduler estimates the per-shot runtime for each subcircuit as the total duration of subcircuit's layers, where each layer represents that set of gates which can be executed simultaneously [16]. The time of a layer is the time of the longest gate in that level, with two-qubit gates being assumed to dominate single-qubit gates. We represent

by t_1 the execution time of a single-qubit gate and by t_2 the execution time of a two-qubit gate, with $t2 \gg t1$. Therefore, for a subcircuit $s_{i,j}$ with $\kappa_{1,i,j}$ single-qubit layers and $\kappa_{2,i,j}$ two-qubit layers the runtime is estimated as [16]:

$$T_{i,j} \;=\; \kappa_{1,i,j}\, t_1 \;+\; \kappa_{2,i,j}\, t_2, \qquad t_2 \gg t_1. \tag{3}$$

In addition, the scheduler receives from the compiler a precedence DAG $(\mathcal{S}_i,\, \mathcal{P}_i)$ capturing dependencies induced by LOCC wire cuts. Subject to this DAG, dependent subcircuits execute only after their predecessors (once the needed classical outcomes are available) whether sequentially on the same QPU or on a different QPU, while independent subcircuits may run in parallel across multiple QPUs. The scheduler assigns each subcircuit to a suitable QPU in terms of qubit count and depth and may distribute the shots of one subcircuit across multiple eligible QPUs. The precise optimzation model is presented in section III.

Finally, each QPU executes its queue of assigned subcircuits and sends measurement outcomes to a classical workstation that recombines results using the quasiprobability weights: c_b for wire cuts a_r for gate cuts. Then, the classical workstation sends the final expectation values for each circuit to the corresponding user (Fig. 1).

III. Deadline-Aware Circuit Scheduling Optimization Formulation

In this section, we first derive the subcircuit and circuit execution time expressions. Then, we formulate a deadline-aware optimization framework that allows the distribution and optimization of subcircuit shots across heterogeneous QPUs, while taking into account the LOCC wire-cut dependencies, QPU heterogeneity, to maximize the number of served requests of the different users within their deadlines, while also minimizing the circuits completion time.

A. Circuit Execution Time Calculation

As discussed in Section II, we allow splitting the shots of a subcircuit across QPUs to increase parallelism and speed up execution. First, for each subcircuit $s_{i,j}$, we define the eligibility set $\mathcal{M}_{i,j} = \{m \in \mathcal{M} : Q_{i,j} \le Q_m, D_{i,j} \le D_m\}$ which has all the QPUs whose qubit and depth limits allow executing $s_{i,j}$. Then, we introduce a shot-allocation variable $y_{i,j,m}$ equal to the number of shots of $s_{i,j}$ executed on QPU $m \in \mathcal{M}_{i,j}$. We also introduce $z_i \in \{0,1\}$ which indicates whether circuit C_i is served on time before its deadline (z_i =1) or discarded ($z_i = 0$). We refer to the execution of subcircuit $s_{i,j}$ for $y_{i,j,m}$ shots on QPU $m \in \mathcal{M}_{i,j}$ as the *fragment* (i,j,m). Each *fragment* (i,j,m) has a beginning execution time $b_{i,j,m}$, which is a control variable, and an end execution time defined using the compiler's per-shot runtime $T_{i,j}$ as

$$e_{i,j,m}(b_{i,j,m}, y_{i,j,m}, z_i) \;=\; \begin{cases} 0, & z_i = 0, \\ b_{i,j,m} + T_{i,j} \cdot y_{i,j,m}, & z_i = 1, \end{cases} \tag{4}$$

The completion time of a subcircuit $s_{i,j} \in \mathcal{S}_i$ with all its shots is the latest *fragment* end-time over all eligible QPUs,

$$e_{i,j}(b_{i,j,m}, y_{i,j,m}, z_i) \;=\; \begin{cases} 0, & z_i = 0, \\ \max\limits_{m \in \mathcal{M}_{i,j}} e_{i,j,m}(b_{i,j,m}, y_{i,j,m}, 1), & z_i = 1, \end{cases} \tag{5}$$

and the circuit C_i completion time $\forall i \in \mathcal{U}$ is the latest subcircuit end-time,

$$T_i^{comp}(b_{i,j,m}, y_{i,j,m}, z_i) \;=\; \begin{cases} 0, & z_i = 0, \\ \max\limits_{j=1,\dots,k} e_{i,j}(b_{i,j,m}, y_{i,j,m}, 1), & z_i = 1. \end{cases} \tag{6}$$

B. Subcircuits Scheduling Optimization Problem

Now, we formulate the scheduling problem to allocate subcircuits across heterogeneous QPUs and synchronize their execution to maximize the number of circuits that are completed before their deadlines and to encourage early completion by awarding a bonus when a circuit finishes within 80% of its deadline. To maximize the served requests $i \in \mathcal{U}$, where $|\mathcal{U}| = U$, the controllable optimization variables in our proposed optimization problem are: 1) $\mathbf{z} = [z_1, \dots, z_i, \dots, z_U]$, 2) $\mathbf{B} = \left[b_{i,j,m}\right]_{i\in\mathcal{U},\ j=1,\dots,k,\ m\in\mathcal{M}_{i,j}}$, 3) $\mathbf{Y} = \left[y_{i,j,m}\right]_{i\in\mathcal{U},\ j=1,\dots,k, m\in\mathcal{M}_{i,j}}$. Accordingly, the deadline-aware and shot-aware scheduling optimization problem can be formulated as follows:

$$\mathcal{O}1: \quad \max_{\mathbf{z},\mathbf{B},\mathbf{Y}} \sum_{i\in\mathcal{U}} z_i \;+\alpha \sum_{i\in\mathcal{U}} \mathbb{1}(T_i^{comp} \le 0.8\cdot\tau_i) \tag{7a}$$

$$\text{s.t.} \quad \sum_{m\in\mathcal{M}_{i,j}} y_{i,j,m} \;=\; N_{i,j}\cdot z_i \quad \forall i,j \tag{7b}$$

$$z_i = 1 \implies T_i^{comp} \le \tau_i \quad \forall i \in \mathcal{U} \tag{7c}$$

$$b_{i,v,m} \ge e_{i,u} \;\; \forall i,\; \forall (s_{i,u} \to s_{i,v}) \in \mathcal{P}_i,\; \forall m \tag{7d}$$

$$y_{i,j,m}\cdot y_{i',j',m} \ge 0 \implies b_{i,j,m} \ge e_{i',j',m} \vee b_{i',j',m} \ge e_{i,j,m} \tag{7e}$$

$$y_{i,j,m} \in \mathbb{N}, \quad b_{i,j,m} \in \mathbb{N}, \quad e_{i,j,m} \in \mathbb{N}. \tag{7f}$$

Constraint (7b) ensures that for every subcircuit $s_{i,j}$, all of its shots $N_{i,j}$, are fully allocated across eligible QPUs when the circuit C_i is served on time; otherwise, all subcircuits are dropped by not allocating shots for them ($y_{i,j,m} = 0$). Constraint (7c) ensures that if circuit C_i is considered served on time (z_i =1), its completion time must not exceed its deadline τ_i. This model tries to maximize how many are on-time by maximizing $\sum_{i\in\mathcal{U}} z_i$. Constraint (7d) makes sure that the precedence conditions from wire cuts are met. A successor subcircuit $s_{i,v}$ (preparation subcircuit) cannot start any of its *fragments* on any QPU until its required predecessor $s_{i,u}$ (paired measurement subcircuit) has finished all its shots (7b). Constraint (7e) guarantees a QPU runs on at most one *fragment* at a time. Finally, constraint (7f) ensures nonnegativity for allocations and time variables.

C. Proposed Solution

To solve problem $\mathcal{O}_1$, we propose a simulated annealing algorithm, a metaheuristic that searches for a near-optimal solution in a large space (Algorithm 1). We start from a

Algorithm 1 Simulated Annealing Algorithm for $\mathcal{O}1$

Initialize the current solution $\mathbf{z},\ \boldsymbol{B},\ \boldsymbol{Y}$ to a feasible solution in the search space
Set initial temperature $\tau_{\text{sol}} = \tau_0$
Define K, the number of performed iterations for each temperature level
Initialize the optimal solution $\mathbf{z}^*,\ \boldsymbol{B^*},\ \boldsymbol{Y^*}$ to the current solution
while $\tau_{\text{sol}} > \tau_{\min}$ **do**
  for $k = 1$ to K **do**
    Generate random neighbors $\mathbf{z}',\ \boldsymbol{B'},\ \boldsymbol{Y'}$ via the following priorities:
    (1) **Add circuit:** set $z'_i = 1$ for some $i \in \mathcal{U}$
    (2) **Permute selected circuits:** permute the selected circuits for scheduling.
    (3) **Local moves on** Y **and** B**:** change shot splits Y' and start times B'
    if generated neighbors $\mathbf{z}',\ \boldsymbol{B'},\ \boldsymbol{Y'}$ satisfy $\mathcal{O}1$ constraints (7b), (7c), (7d) (7e) and (7f) **then**
      Calculate $\Delta O = O(\mathbf{z}', \boldsymbol{B'}, \boldsymbol{Y'}) - O(\mathbf{z}, \boldsymbol{B}, \boldsymbol{Y})$
      if $\Delta O \geq 0$ **or** $\exp(\Delta O/\tau_{\text{sol}}) > r \in \text{Uniform}[0,1]$ **then**
        Update $\mathbf{z},\ \boldsymbol{B},\ \boldsymbol{Y}$ to $\mathbf{z}',\ \boldsymbol{B'},\ \boldsymbol{Y'}$, respectively
      end if
    end if
  end for
  if $O(\mathbf{z}',\ \boldsymbol{B'},\ \boldsymbol{Y'}) \geq O(\mathbf{z}^*,\ \boldsymbol{B^*},\ \boldsymbol{Y^*})$ **then**
    Update the best solution $\mathbf{z}^*,\ \boldsymbol{B^*},\ \boldsymbol{Y^*}$ to $\mathbf{z}',\ \boldsymbol{B'},\ \boldsymbol{Y'}$, respectively
  end if
  Update τ_{sol} according to adopted cooling schedule, e.g., exponential cooling, as $\tau_{\text{sol}} = \tau_{\text{sol}} \cdot \alpha_{\text{sol}}$
end while
return the best solution found, $\mathbf{z}^*,\ \boldsymbol{B^*},\ \boldsymbol{Y^*}$.

feasible solution $(\mathbf{z}, \mathbf{Y}, \mathbf{B})$ that satisfies all constraints of $\mathcal{O}_1$, set it as both the current and the best solution, choose a high initial temperature, and run a fixed number of iterations at each temperature level. In each iteration, we create a neighbor in the following priority: (i) *add a circuit* by including an unscheduled C_i (set $z_i = 1$) and fixing $(\mathbf{Y}, \mathbf{B})$ accordingly. If no circuit can be added, (ii) *permute the selected set for scheduling* by swapping one selected circuit with another dropped one and repairing $(\mathbf{Y}, \mathbf{B})$. If neither works, (iii) we try a local move on $(\mathbf{Y}, \mathbf{B})$ that change execution order, QPU assignment, or shot splitting, such as: pushing a *fragment* forward or backward on its QPU to reduce gaps, moving a subcircuit to another QPU, or splitting subcircuits shots across QPUs. Every candidate neighbor $(\mathbf{z}', \mathbf{Y}', \mathbf{B}')$ is checked for feasibility and evaluated by calculating the difference in the objective function. If it is better, the neighbor is accepted; otherwise, we may still accept it with a probability given by the Metropolis rule to help escape local minima. The temperature is reduced using an exponential cooling schedule until it reaches τ_{min}, and the best solution is returned.

IV. Simulation Results and Analysis

In this section, we conduct simulations to evaluate the efficiency and performance of our proposed deadline-aware scheduling framework. Unless stated otherwise, we adopt the following *default setup* for our experiment. We consider $M = 5$ heterogeneous QPUs. For each QPU $m \in \mathcal{M}$, the qubit capacity Q_m and the maximum feasible circuit depth D_m are sampled from a uniform distribution such that $Q_m \sim \text{Uniform}[10, 20]$ qubits [20], $D_m \sim \text{Uniform}[10, 20]$ layers, $\forall m \in \mathcal{M}$. A compiler partitions each circuit C_i of request $i \in \mathcal{U}$ into subcircuits using either a gate cut or a wire cut. Following prior work [7] and [21], we treat the compilers circuit-cutting decomposition time as negligible compared to QPU execution time. Moreover, we also treat the final classical post-processing, where results are weighted and averaged, as negligible relative to the dominant QPU execution time, which is primarily affected by the sampling overhead and the number of shots. We consider CNOT gate cutting, where a decomposed gate produces 12 independent subcircuits with no precedence [5]. The LOCC wire cut produces 6 subcircuits with 3 measurement and 3 preparation subcircuits. A measurement subcircuit must finish before its corresponding preparation subcircuit, so we sample three precedence edges. We assume that classical communication between measurement and preparation subcircuits does not significantly increase the execution runtime, so can be practically ignored [7]. In contrast, for the old wire cut method without dependencies, it results in 16 independent subcircuits [4], [22]. For each subcircuit $s_{i,j} \in S_i$, we sample $Q_{i,j}$ and $D_{i,j}$ from uniform distributions, such that $Q_{i,j} \sim \text{Uniform}[5, 20]$ qubits [22], and $D_{i,j} \sim \text{Uniform}[5, 20]$ layers [14], respectively, until the subcircuit is eligible on at least one QPU to ensure feasibility. Additionally, we sample each subcircuits single-qubit and two-qubit layer counts uniformly, where $\kappa_{1,i,j} \sim \text{Uniform}[2, 10]$ single-qubit gate layers, and $\kappa_{2,i,j} \sim \text{Uniform}[3, 10]$ two-qubit gate layers, respectively. The sampling is done while ensuring that $\kappa_{1,i,j} + \kappa_{2,i,j} = D_{i,j}$, and hence we set the subcircuit per-shot runtime as $T_{i,j} = \kappa_{1,i,j}\, t_1 + \kappa_{2,i,j}\, t_2$, [16]. To obtain reliable expectation values with high precision, we fix an uncut baseline of N_0 shots per circuit, noting that achieving

TABLE I: Summary of Simulation Parameters

Parameter	Description	Value
M	Number of QPUs in the quantum cloud	5
N_0	Original number of shots	10000 [19]
t_1	Single-qubit gate execution time	$t_1 = 1$ [16]
t_2	Two-qubit gate execution time	$t_2 = 10$ [16]
γ_g^2	Sampling overhead for gate cut	9 [5]
γ_{lw}^2	Sampling overhead for LOCC wire cut	9 [7]
γ_{ow}^2	Sampling overhead for old wire cut	16 [4]

TABLE II: Comparison of scheduling methods with the optimal solution.

Method	T_1^{comp}	T_2^{comp}	Served Requests
Optimal (exhaustive)	190	300	2
Proposed 1	192	300	2
Greedy	134	—	1
List	140	—	1
Random	240	338	2
Shot-agnostic	230	—	1
Dependency-agnostic	240	—	1
Dependency-and-shot-agnostic	240	—	1

precision ϵ typically requires $N_0 = \mathcal{O}(1/\epsilon^2)$ [23]. After cutting, the required number of shots scales with the sampling overhead factor γ^2 of the chosen cut method, so the total shot budget per circuit becomes $N_0\,\gamma^2$. We divide the circuit's total shots across the resulting subcircuits in proportion to the quasiprobability weights. For the cut methods considered in our work, these weights are equal, so the shots are split evenly across subcircuits [4], [5], [7]. The gates execution time, N_0, and sampling overhead values adopted in the default setup, are shown in Table I. Finally, for each circuit C_i, we compute a lower bound T_i^{ref} which is the smallest achievable completion time, assuming its subcircuits may execute simultaneously in parallel across eligible QPUs while respecting LOCC wire-cut precedency. Then, to model different user urgency levels, we sample a deadline coefficient $d_{c,i} \sim \text{Uniform}[3,\,10]$, and set the actual deadline as $\tau_i = d_{c,i}\,T_i^{\text{ref}}, \forall i \in \mathcal{U}$.

In order to evaluate the performance of our *proposed* framework and solution algorithm, we compare it against several state-of-the-art scheduling algorithms. For a fair comparison, we consider all benchmark algorithms to be deadline-aware, although they are not originally designed to consider the deadlines associated with the incoming requests [8], [9], [11]. The benchmarks we compare against are as follows: 1) *greedy:* earliest-deadline circuits are served first, and subcircuits can split their shots evenly across free eligible QPUs while respecting precedence constraints [11], 2) *random:* a scheduling algorithm that builds feasible schedules by randomly ordering subcircuits and assigning them to eligible QPUs, and can also stochastically split a subcircuits shots across multiple eligible QPUs, 3) *list:* which follows the classic list scheduling algorithm, processing subcircuits in a precedence-respecting order and evenly splitting shots across free eligible QPUs [8], [9], 4) *shot-agnostic:* which represents the state-of-the-art approach in which shots are not distributed, treating each subcircuits shots as a single job, 5) *dependency-agnostic:* which uses the widely adopted wire-cut method [4] instead of LOCC wire cuts, obtaining independent subcircuits, 6) *dependency-and-shot-agnostic:* also uses the wire-cut method [4] and never splits a subcircuits shots between QPUs. We compare all these models with our *proposed* scheduling framework which is both shot and dependency-aware.

We begin our simulations by running an exhaustive search solver to the *proposed* optimization problem $\mathcal{O}_1$ to identify the *optimal* schedule, and compare it against our *proposed* simulated-annealing solver and the other benchmarks. We consider an instance with two requested circuits (C_1, C_2) to be cut and scheduled with original $N_0 = 10$ shots per circuit. Table II summarizes the results, where the optimal schedule served both circuits with $T_1^{\text{comp}} = 190$ and $T_2^{\text{comp}} = 300$. Our *proposed* framework also served both circuits, matching completion time of C_2 exactly and finishing C_1 at $T_1^{\text{comp}} = 192$, which is a negligible 1.05% performance gap. The *random* scheduler also served both circuits but with significantly longer completion times. The other benchmarks served at most one circuit in this setup because C_2 misses its deadline. In terms of runtime, our *proposed* approach produced a near-optimal schedule with a 97.9% runtime reduction compared to the optimal exhaustive search solution. Accordingly, we use the *proposed* scheduler for the remaining experiments at larger scales without running the exhaustive search solution.

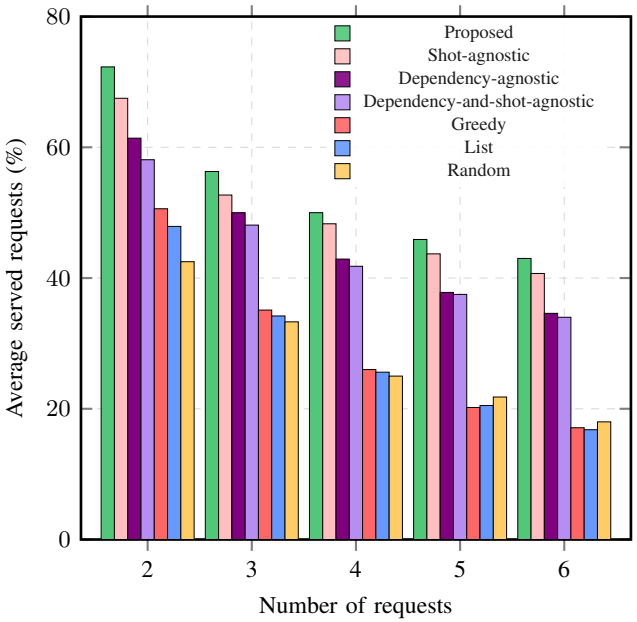


Fig. 2: Average served requests vs. Number of requests.

Next, in Fig. 2, we examine the effects of increasing the number of requests on the average served requests for the different benchmarks over 50 independent Monte Carlo runs. We vary the number of user requests from 2 to 6 and observe from Fig. 2 that the average served requests decreases as the number of requests grows. This trend is expected: with more requests, a larger number of subcircuits compete for the same limited QPU queue space, leading to higher contention. Across all evaluated number of requests, our *proposed* approach consistently achieves the highest average number of served requests. Specifically, it outperforms the *shot-agnostic* baseline by an average of 2.92% (ranging from 1.7% to 4.8%), the *dependency-agnostic* baseline by 8.16% on average (with gains between 6.3% and 10.9%), and the *dependency-and-shot-agnostic* baseline by 9.60% on average (ranging from 8.2% to 14.2%). The improvements over the heuristic benchmarks are even more pronounced, reaching 23.7% over *greedy*, 24.5% over *list*, and 25.38% over *random* scheduling algorithms.

Next, Fig. 3 reports the average makespan as the number of requests varies, using the same setup as Fig. 2. As expected, makespan increases for all benchmarks as queues grow longer with higher load. Under light load (2 requests), the *dependency-agnostic* scheme is faster than *proposed* by 2.8%, since the wire-cutting method of [4] produces independent subcircuits that can start immediately without LOCC precedence constraints. In contrast, the *proposed* approach enforces LOCC dependencies between measurement and preparation subcircuits, requiring each prepare subcircuit to wait for its paired measurement subcircuit to finish. As the number of requests increases, however, the *proposed* framework achieves

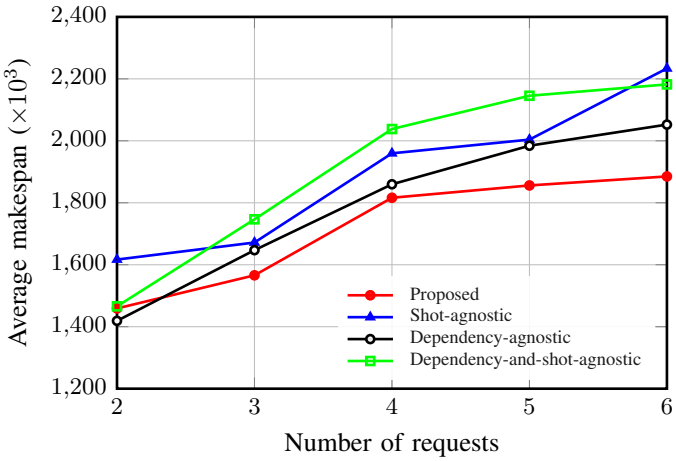
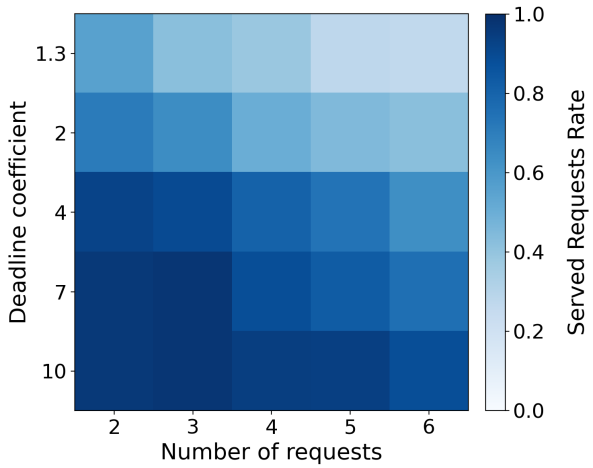


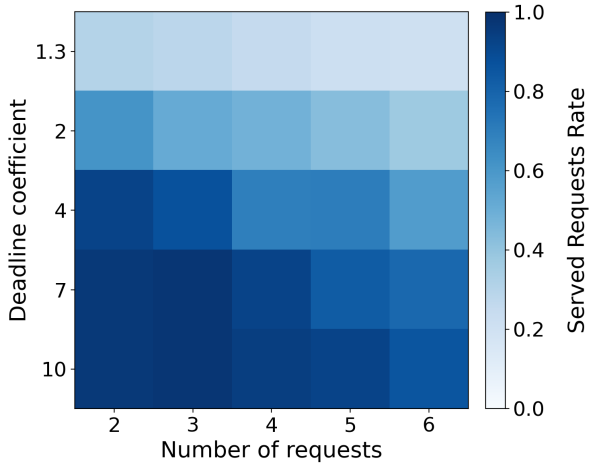
Fig. 3: Average makespan vs number of requests.

a lower makespan despite respecting these dependencies. Its LOCC wire cut generates fewer subcircuits and imposes a smaller sampling overhead, reducing the total amount of work queued on the QPUs. Moreover, the waiting periods of preparation subcircuits become naturally masked at higher load, as subcircuits from other circuits occupy the QPUs during those intervals. On average, *proposed* reduces makespan by 4.34% relative to *dependency-agnostic* (ranging from 2.8% at 2 requests to 8.2% at 6 requests) frameworks. Compared to *shot-agnostic*, it lowers makespan by an average of 9.28% (range: 6.33% to 15.60%). Also, against *dependency-and-shot-agnostic*, reductions vary from 0.46% to 13.61%, averaging 9.76%. Taken together, Figs. 2 and 3 show that the *proposed* scheduler provides a clear advantage, since it generally serves more requests while achieving a shorter makespan than all competing baselines. The combination of efficient LOCC wire cuts and informed shot distribution keeps the QPUs more effectively utilized, translating into both higher served throughput and faster overall completion times.

Back to Fig. 2, we observe that the performance gap between the *proposed* and *shot-agnostic* frameworks is relatively modest, with an average difference of 2.92% in served requests. This behavior is primarily due to the randomized deadline coefficients, which were sampled from the interval $[3, 10]$. The presence of multiple circuits with generous deadlines reduces the urgency of scheduling decisions, allowing the *shot-agnostic* scheme to admit only slightly fewer requests than the *proposed* method under these mixed-deadline conditions. To more systematically investigate the influence of deadline tightness, we, next, conduct an experiment that jointly varies the deadline coefficient and the number of user requests, and show the results in Fig. 4. Using the same default experimental setup, we sweep the deadline factor $d_c \in \{1.3, 2, 4, 7, 10\}$, covering the full spectrum from very tight to very lenient deadlines. For each value of d_c, we vary the number of requests from 2 to 6. In each sweep, all requests share the same deadline factor, and we report the average number of served requests across multiple runs to capture the combined effect of system load and deadline urgency. Figs. 4a and 4b



(a) Proposed

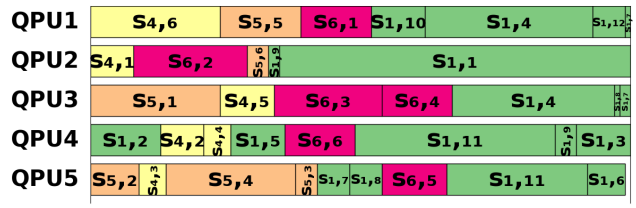


(b) Shot-agnostic

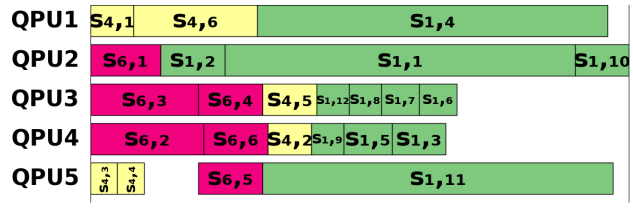
Fig. 4: Served requests rate vs deadline factor and number of requests.

show that for both the *proposed* and *shot-agnostic* frameworks, the average number of served requests decreases as deadlines tighten. When deadlines are generous, the two approaches perform similarly: at $d_c = 10$, the *proposed* method offers only a 0.8% improvement, and at $d_c = 7$ the gain is 1.2% on average. The differences become more pronounced as deadlines grow tighter. At $d_c = 4$, the *proposed* approach outperforms *shot-agnostic* by 4.8% on average (and by 5% at 6 requests). For $d_c = 2$, the average improvement increases to 6.2%, with a notable 13% gain at 3 requests. Under the tightest constraint, $d_c = 1.3$, the advantage becomes substantial, reaching 12.8% on average, including 14% gains at 3 and 4 requests and a peak of 25% at 2 requests. This demonstrates that the benefits of the *proposed* scheme become increasingly significant as user deadlines grow more urgent.

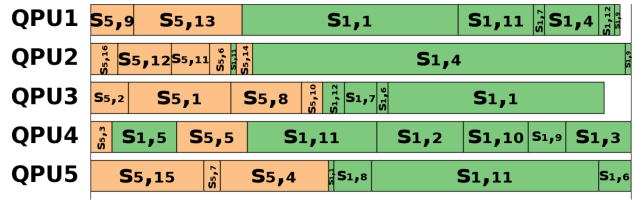
Finally, in Fig. 5 we visualize the schedules produced by different benchmarks for a single instance with six circuits in the request buffer, where C_1 and C_2 are partitioned via gate cuts, while C_3C_6 use wire cuts. As shown in Figs. 5a and 5b, the circuits partitioned via LOCC wire cuts yield six subcircuits



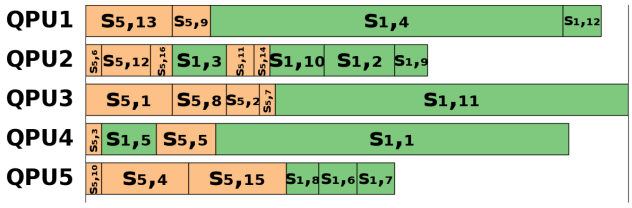
(a) Proposed



(b) Shot-agnostic



(c) Dependency-agnostic.



(d) Dependency-and-shot-agnostic.

Fig. 5: Scheduling outputs of different benchmarks.

each, with precedence respected so that a prepare fragment starts only after its paired measurement subcircuit finishes. Specifically, for C_4 the edges are $(s_{4,1} \to s_{4,2})$, $(s_{4,3} \to s_{4,4})$, and $(s_{4,6} \to s_{4,5})$; for C_5: $(s_{5,1} \to s_{5,6})$, $(s_{5,2} \to s_{5,5})$, and $(s_{5,4} \to s_{5,3})$; and for C_6: $(s_{6,1} \to s_{6,4})$, $(s_{6,2} \to s_{6,6})$, and $(s_{6,3} \to s_{6,5})$. By contrast, the gate cut produces 12 independent subcircuits with no precedence constraints. In this sample, *proposed* admits four circuits, whereas *shot-agnostic* serves only three. The tighter scheduling enabled by subcircuit shot splitting (e.g., $s_{1,4}$ split across QPU_1 and QPU_3) keeps the QPUs busier, providing enough capacity to admit a fourth circuit. In Figs. 5c and 5d, both benchmarks admit only two circuits. Here, the precedence edges mentioned above are no longer a constraint, since the older wire-cut technique produces fully independent subcircuits. However, the wire cut increases the number of subcircuits from 6 to 16 in C_5 and increases the sampling overhead, causing fewer deadlines to be met.

V. Conclusion

In this paper, we have studied the scheduling of DQC circuits in near-term quantum clouds with heterogeneous QPUs under LOCC constraints. Specifically, we proposed a deadline-aware scheduling optimization framework for that optimizes the per-subcircuit shot distribution across QPUs, captures subcircuit dependencies, and minimizes runtime to serve as many circuits as possible. Furthermore, we conducted several experiments to study the impact of shot distribution and the effect of adopting LOCC wire cuts, finding that both contribute to more served requests and lower makespans. Going forward, we aim to jointly optimize circuit cutting and scheduling on heterogeneous QPUs.

References

[1] D. Barral *et al.*, "Review of distributed quantum computing: From single qpu to high performance quantum computing," *Computer Science Review*, vol. 57, p. 100747, 2025.

[2] M. Caleffi, M. Amoretti, D. Ferrari, J. Illiano, A. Manzalini, and A. S. Cacciapuoti, "Distributed quantum computing: a survey," *Computer Networks*, vol. 254, p. 110672, 2024.

[3] M. Chehimi and W. Saad, "Physics-informed quantum communication networks: A vision toward the quantum internet," *IEEE network*, vol. 36, no. 5, pp. 32–38, 2022.

[4] T. Peng *et al.*, "Simulating large quantum circuits on a small quantum computer," *Phys. Rev. Let.*, vol. 125, no. 15, Oct. 2020.

[5] K. Mitarai and K. Fujii, "Constructing a virtual two-qubit gate by sampling single-qubit operations," *New Journal of Physics*, vol. 23, no. 2, p. 023021, Feb. 2021.

[6] M. Tejedor, B. Casas, J. Conejero, A. Cervera-Lierta, and R. M. Badia, "Distributed quantum circuit cutting for hybrid quantum-classical high-performance computing," *arXiv preprint arXiv:2505.01184*, 2025.

[7] H. Harada *et al.*, "Doubly optimal parallel wire cutting without ancilla qubits," *PRX Quantum*, vol. 5, no. 4, p. 040308, 2024.

[8] D. Ferrari, M. Bandini, and M. Amoretti, "Execution management of distributed quantum computing jobs," in *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 02, 2024, pp. 150–154.

[9] D. Ferrari and M. Amoretti, "A design framework for the simulation of distributed quantum computing," in *Proc. of the 2024 Workshop on High Performance and Quantum Computing Integration*, 2024, pp. 4–10.

[10] S. Bahrani *et al.*, "Resource management and circuit scheduling for distributed quantum computing interconnect networks," *arXiv preprint arXiv:2409.12675*, 2024.

[11] N. K. Chandra, E. Kaur, and K. P. Seshadreesan, "Network operations scheduling for distributed quantum computing," in *2024 IEEE 6th International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications (TPS-ISA)*. IEEE, 2024, pp. 506–515.

[12] Z. Du, W. Zhang, W. Wei, J. Chen, T. Han, Z. Liang, and Y. Mao, "Efficient circuit cutting and scheduling in a multi-node quantum system with dynamic epr pairs," *arXiv preprint arXiv:2412.18709*, 2024.

[13] L. Liu, "Larqucut: A new cutting and mapping approach for large-sized quantum circuits in distributed quantum computing (dqc) environments," *ACM Transactions on Architecture and Code Optimization*, 2025.

[14] S. Brandhofer, I. Polian, and K. Krsulich, "Optimal partitioning of quantum circuits using gate cuts and wire cuts," *IEEE Transactions on Quantum Engineering*, vol. 5, pp. 1–10, 2024.

[15] W. Luo, J. Zhao, T. Zhan, and Q. Guan, "Adaptive job scheduling in quantum clouds using reinforcement learning," *arXiv preprint arXiv:2506.10889*, 2025.

[16] D. Bhoumik, R. Majumdar, A. Saha, and S. Sur-Kolay, "Distributed scheduling of quantum circuits with noise and time optimization," *arXiv preprint arXiv:2309.06005*, 2023.

[17] G. Bisicchia, A. Bocci, J. García-Alonso, J. M. Murillo, and A. Brogi, "Cut&shoot: Cutting & distributing quantum circuits across multiple nisq computers," in *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 2. IEEE, 2024, pp. 187–192.

[18] A. Carrera Vazquez, C. Tornow, D. Riste, S. Woerner, M. Takita, and D. J. Egger, "Combining quantum processors with real-time classical communication," *Nature*, vol. 636, no. 8041, pp. 75–79, 2024.

[19] T.-H. Chen and W.-H. Hung, "Feasibility and limitations of generalized grover search algorithm-based quantum asymmetric cryptography: An implementation study on quantum hardware," *Electronics*, vol. 14, no. 19, 2025.

[20] S. Kan *et al.*, "Scalable circuit cutting and scheduling in a resource-constrained and distributed quantum system," in *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 1. IEEE, 2024, pp. 1077–1088.

[21] N. Tornow, E. Giortamis, and P. Bhatotia, "Scaling quantum computations via gate virtualization," *arXiv preprint arXiv:2406.18410*, 2024.

[22] W. Tang *et al.*, "Cutqc: using small quantum computers for large quantum circuit evaluations," in *Proceedings of the 26th ACM International conference on architectural support for programming languages and operating systems*, 2021, pp. 473–486.

[23] Z. Cai *et al.*, "Quantum error mitigation," *Rev. Mod. Phys.*, vol. 95, p. 045005, Dec 2023.