Adaptive Job Scheduling in Quantum Clouds Using Reinforcement Learning

Waylon Luo
Kent State University
Kent, OH, USA

Jiapeng Zhao
Cisco
San Jose, CA, USA

Tong Zhan
Meta
Menlo Park, CA, USA

Qiang Guan
Kent State University
Kent, OH, USA

ABSTRACT

Present-day quantum systems face critical bottlenecks, including limited qubit counts, brief coherence intervals, and high susceptibility to errors—all of which obstruct the execution of large and complex circuits. The advancement of quantum algorithms has outpaced the capabilities of existing quantum hardware, making it difficult to scale computations effectively. Additionally, inconsistencies in hardware performance and pervasive quantum noise undermine system stability and computational accuracy. To optimize quantum workloads under these constraints, strategic approaches to task scheduling and resource coordination are essential. One of the persistent challenges in this domain is how to efficiently divide and execute large circuits across multiple quantum processors (QPUs), especially in error-prone environments. In response, we introduce a simulation-based tool that supports distributed scheduling and concurrent execution of quantum jobs on networked QPUs connected via real-time classical channels. The tool models circuit decomposition for workloads that surpass individual QPU limits, allowing for parallel execution through inter-processor communication. Using this simulation environment, we compare four distinct scheduling techniques—among them, a model informed by reinforcement learning. These strategies are evaluated across multiple metrics, including runtime efficiency, fidelity preservation, and communication costs. Our analysis underscores the trade-offs inherent in each approach and highlights how parallelized, noise-aware scheduling can meaningfully improve computational throughput in distributed quantum infrastructures.

KEYWORDS

Quantum cloud simulations, quantum job scheduling, simulation frameworks

1 INTRODUCTION

Quantum computing has the potential to transform fields such as cryptography, materials science, and machine learning by addressing problems beyond the capabilities of classical systems [27]. The foundational concept of using quantum machines to simulate complex physical phenomena was first introduced by Richard Feynman [12]. Advances in both quantum hardware and algorithm design have recently brought this vision closer to reality, broadening the range of applications to areas such as pharmaceutical research [39, 44], structural biology [43], financial analytics [7, 13], optimization problems, and machine learning tasks [4].

Nevertheless, quantum computing still faces significant limitations due to hardware challenges, including restricted qubit numbers, fragile coherence, and substantial error rates. Quantum noise—originating from processes such as decoherence, gate infidelity, and measurement inaccuracies [10, 27]—remains a major hurdle to achieving scalable computations. To address these issues, major quantum cloud platforms, such as IBM Quantum and Quantinuum, provide access to real-time calibration metrics, including coherence lifetimes, gate operation errors, and readout fidelity [8, 9]. Incorporating this dynamic information into scheduling techniques helps limit error propagation and improve the fidelity of quantum executions.

A key technical bottleneck for large-scale quantum computation is the constrained connectivity among qubits within a single processor. One emerging solution is to link multiple quantum processing units (QPUs) through classical communication networks. Vazquez et al. [40] demonstrated the practical execution of quantum circuits distributed across two 127-qubit processors interconnected via real-time classical channels. Although this marks important progress toward building scalable quantum platforms, the problem of optimally distributing and scheduling large quantum circuits across multiple QPUs remains largely unresolved.

Effective scheduling plays a crucial role in distributing large quantum circuits across multiple QPUs, aiming to optimize execution time, resource allocation, and circuit fidelity. In this work, we investigate four distinct scheduling approaches: (1) *speed-optimized scheduling*, which focuses on minimizing runtime by uniformly distributing jobs across available QPUs without considering hardware-specific differences; (2) *fidelity-optimized scheduling*, which dynamically routes jobs to QPUs with superior calibration metrics, including lower gate and readout errors, using real-time hardware data; (3) *fair scheduling*, which partitions circuits and allocates them evenly among all processors to promote balanced resource usage; and (4) *reinforcement learning-based scheduling*, which applies machine learning techniques from the Gymnasium [30] framework to learn optimal allocation strategies.

The primary contributions of this study are summarized below.

- **Flexible quantum job orchestration** — The framework supports both built-in and user-defined scheduling policies, along with integration of noise models to optimize performance for specific applications.
- **Adaptive, error-informed scheduling** — We design and implement a dynamic scheduling method that uses calibration data to enhance circuit fidelity during execution.
- **Evaluation metrics** — We establish key benchmarks, including execution duration, fidelity levels, and communication overhead, to systematically assess scheduling effectiveness.
- **Trade-off analysis between speed and fidelity** — Our study highlights the inherent balance between minimizing runtime and maintaining computational accuracy in fast-execution and error-aware strategies.

- **Publicly available simulation platform** — We introduce the first open-source framework that supports scheduling of quantum circuits exceeding the capacity of a single QPU [35].

The remainder of this paper is structured as follows. Section 2 provides an overview of quantum computing and related prior work. Section 3 details the architectural design of the simulation framework. Section 4 defines the core problems addressed in this study. Section 5 introduces the proposed noise-aware scheduling strategies. Section 6 describes the performance evaluation metrics. Section 7 presents the experimental evaluation and results. Finally, Section 8 concludes the paper by summarizing the main contributions and insights.

2 RELATED WORK

In quantum computing, information is encoded in quantum bits (qubits), which can exist in a superposition of basis states $|0\rangle$ and $|1\rangle$, allowing them to represent both classical values 0 and 1 simultaneously [21]. Furthermore, qubits exhibit entanglement—a phenomenon where the state of one qubit becomes intrinsically linked to that of another, regardless of the physical distance between them. These fundamental properties allow quantum computers to address specific computational problems more efficiently than classical systems [27].

A quantum circuit, composed of quantum gates, is executed on a quantum processing unit (QPU) by mapping logical qubits to physical qubits. During circuit execution, a sequence of gate operations is applied. At the conclusion of the computation, qubit measurements are performed [21]. Today, researchers in machine learning [4], pharmacological discovery [44], quantum application tools [32, 33], quantum noise visualization [31], and financial modeling [13] are actively executing quantum circuits on quantum computers to explore applications such as molecular simulation, portfolio optimization, and risk analysis.

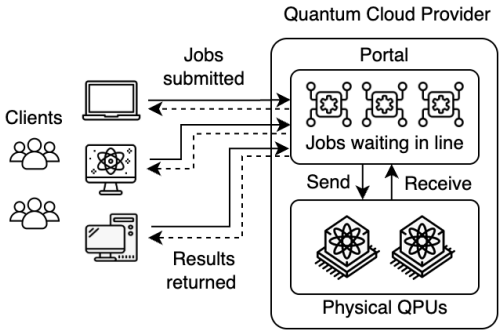


Figure 1: Overview of a quantum cloud computing system.

With advancements in quantum computing, Quantum Cloud Computing has emerged as a viable solution for remote access to quantum resources. It allows users to execute quantum algorithms on high-performance quantum processors via the cloud without owning costly hardware. Clients submit quantum jobs to a cloud-based portal, where jobs are queued and managed, as illustrated in Fig. 1. The portal dispatches jobs to available physical quantum processing units (QPUs) for execution. Upon completion, results are returned to the respective clients. This architecture captures the typical workflow and interaction between users and a quantum cloud provider. Companies such as IBM, Amazon Braket, and Microsoft provide public quantum platforms [5, 17, 28], though these systems remain in the early stages compared to established cloud services like AWS and Azure. Simulations are critical to bridging this gap, allowing researchers to investigate quantum workloads, optimize resource allocation, and assess noise impacts in a cost-effective, flexible, and scalable manner.

Today's QPUs have a limited number of qubits, which constrains the size of quantum circuits that can be executed natively. One strategy to address these limitations is *circuit cutting*, where a large quantum circuit is partitioned into smaller subcircuits that run independently on smaller QPUs. Classical post-processing is then used to reconstruct the final result [38]. While circuit cutting is effective for certain problems, it introduces additional computational overhead and may be impractical when synchronous execution across multiple QPUs is required. In such cases, alternative techniques, such as real-time classical communication between QPUs, must be considered.

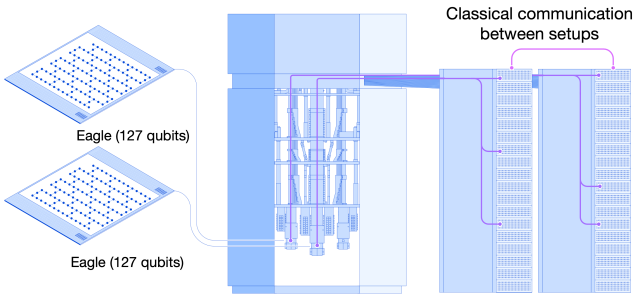


Figure 2: Connecting two QPUs via real-time classical communication. Image source: Vazquez et al. (2024) [40].

To address this limitation, Vazquez et al. conducted an experiment that combined two 127-qubit QPUs into a single virtual QPU using real-time classical communications [40], as illustrated in Fig. 2. For example, executing a 150-qubit circuit on 127-qubit QPUs such as *ibm_eagle* requires splitting the circuit into smaller subcircuits and distributing them across two or more QPUs through classical communication channels. In this context, classical communication refers to the absence of quantum links between devices, relying instead on classical data transfer. This approach allows multiple processors to cooperate on a single computation by exchanging classical data mid-circuit, thereby extending the computational reach of current quantum hardware.

In the quantum computing field, simulations span from hardware-level noise modeling to high-level scheduling algorithm testing. Several existing frameworks have addressed various aspects of cloud and quantum simulation, but differ significantly from our work in both scope and functionality. DRAS-CQSim [11] and GymCloudSim [14] focus on classical high-performance cluster scheduling and energy-driven cloud scaling, respectively, without addressing quantum-specific or multi-QPU challenges. QuNetSim [6] and NetSquid [3] provide discrete-event simulations for quantum

Table 1: Comparison of Quantum Cloud and Quantum Simulation Frameworks

Framework	Research Focus	Research Methodology	Noise Aware	Combined QPUs
QuNetSim [6]	Quantum network simulation	Discrete-event simulation	✘	✘
NetSquid [3]	Quantum networking and communication	Discrete-event simulation	✓	✘
QuEST [16]	High-performance quantum state simulation	State-vector simulation	✘	✘
PAS [1]	Lightweight quantum circuit simulation	State-vector simulation	✘	✘
QXTools [2]	Distributed quantum circuit simulation in Julia	Distributed simulation	✘	✘
ProjectQ [36]	Quantum programming framework	Compiler and circuit simulation	✘	✘
iQuantum [22]	Quantum cloud modeling and simulation	Discrete-event simulation	✘	✘
QSimPy [23]	Learning-centric simulation for quantum clouds	Discrete-event simulation	✘	✘
QuCloud [18]	Qubit mapping for multi-programming	Algorithmic modeling	✘	✘
QuMC [25]	Hardware-aware multi-programming compiler	Compiler-based simulation	✓	✘
Ravi et al. [29]	Quantum cloud job scheduling framework	Simulation-based evaluation	✘	✘
QURE [37]	Resource estimation in quantum clouds	Estimation modeling	✘	✘
QuTiP [15]	Open quantum system dynamics simulation	Monte Carlo simulation	✓	✘
This work	Large-scale circuit simulation	Discrete-event simulation	✓	✓

networks, where NetSquid incorporates noise models; however, both concentrate solely on network communication rather than distributed computation. QuEST [16], PAS [1], and QXTools [2] emphasize quantum circuit simulation with high performance or distributed execution but do not address resource variability or scheduling in a cloud setting. Similarly, ProjectQ [36] offers a quantum programming framework with circuit simulation capabilities but lacks cloud and scheduling support. iQuantum [22] and QSimPy [23] propose quantum cloud simulation environments but emphasize modeling- and learning-centric frameworks without incorporating noise awareness or explicit job distribution across multiple QPUs. QuCloud [18] and QuMC [25] address multi-programming on single quantum processors by optimizing qubit mapping and compiler techniques. Although QuMC incorporates hardware noise awareness, it remains limited to single-device settings. Ravi et al. [29] present a quantum job scheduling framework, yet without integrating noise models or real-time communication constraints. QURE [37] focuses on estimating resource requirements in quantum cloud systems rather than executing or scheduling jobs. QuTiP [15] simulates open quantum system dynamics using Monte Carlo methods, providing noise modeling but targeting physical dynamics instead of distributed quantum computing. The related work comparison is presented in Table 1.

Existing approaches address quantum circuit execution, networking, or classical workload management in isolation. Our noise-aware extension of QCloudSim [19] is the first framework to simulate quantum circuits that exceed the capacity of a single device that supports circuit partitioning, synchronized execution, and inter-device communication.

3 SIMULATION ARCHITECTURE

Our simulation framework adopts a layered architecture to model quantum cloud infrastructures. As illustrated in Fig. 3, the framework comprises four main layers: the **Python Layer**, the **SimPy Library**, which provides the underlying simulation capabilities; the **Quantum Cloud Simulation Layer**, which models quantum

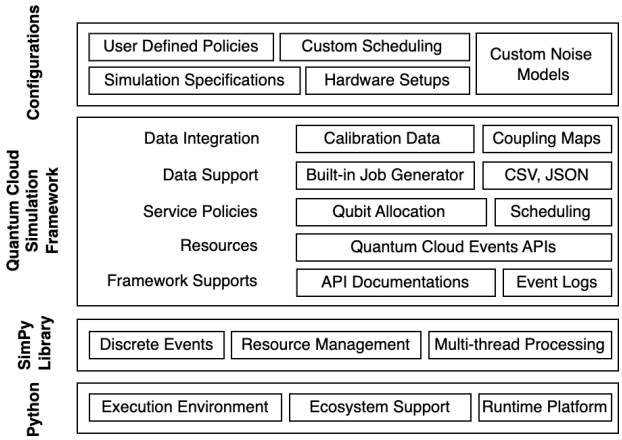


Figure 3: The architecture of the simulation framework consists of four layers: Python Layer, SimPy Library, Quantum Cloud Simulation Framework, and Configurations.

cloud-specific components and operations; and the **Configurations Layer**, where users define simulation specifications.

The **Python Layer** underpins the framework by supporting the simulation ecosystem and providing a programming environment integrated with essential scientific libraries, including `NumPy` for numerical operations, `networkx` for graph organization, and `Matplotlib` for visualization and analysis.

The **SimPy Library** [34] serves as the core event-driven simulation engine, managing discrete-event scheduling and resource allocation. Built on SimPy, our framework extends its functionality to support quantum cloud workload modeling and execution.

The **Quantum Cloud Simulation Layer** forms the backbone of the framework, modeling key quantum cloud components. This layer provides APIs for simulating core entities, framework utilities, and documentation to guide users in developing and extending the simulation. Customizable job scheduling and allocation strategies reside here. Users can implement tailored strategies based on

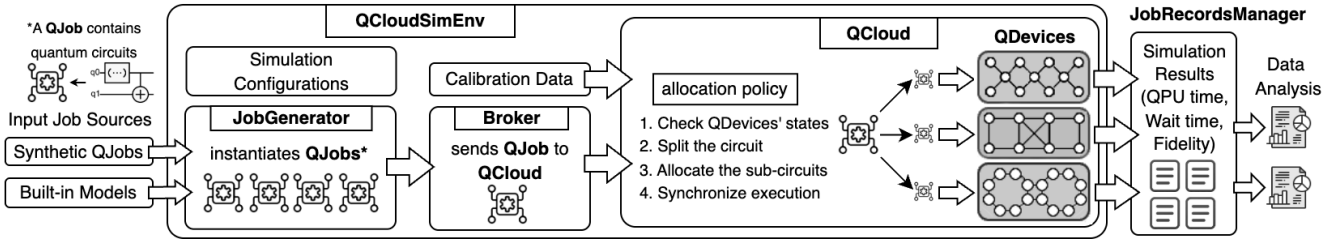


Figure 4: The ecosystem of the framework simulates the end-to-end orchestration of quantum jobs, beginning from job sources (e.g., CSV/JSON files, or built-in models), through the framework environment—comprising the JobGenerator, Broker, and local queue management—to execution on quantum devices. Simulation results are processed and visualized by the JobRecordsManager, providing insights into system performance and execution metrics.

optimization objectives. The layer also supports stochastic job generation and deterministic job flow through external data formats such as CSV and JSON. Centralized data management tracks job lifecycles, supporting post-simulation workload analysis. Furthermore, this layer integrates a library of QPU profiles, including coupling maps, device performance metrics, and calibration data.

The **Configurations Layer** sits at the top and provides interfaces for defining policies and simulation parameters without altering the core architecture. Users must specify scheduling and allocation policies, simulation parameters, and hardware configurations before running simulations.

The quantum cloud simulation environment, or **`QCloudSimEnv`**, orchestrates job flow through modular subcomponents. It serves as the core component of the framework. `QCloudSimEnv` consists of a `QCloud`, one or more `QuantumDevice` instances, a `JobGenerator`, a `JobRecordsManager`, and a `Broker`.

The quantum cloud, or **`QCloud`**, initialized within `QCloudSimEnv`, manages quantum devices, allocates large circuits, and handles device communication. Allocation policies for large-scale circuits are implemented in this entity. These policies govern circuit partitioning and distribution across devices. In this work, jobs with large quantum circuits are partitioned into smaller circuits and allocated across `QuantumDevices`.

The **`QuantumDevice`** class, representing quantum devices and extended from `BaseQDevice`, defines qubit topology, operational characteristics, and resource management. The `QuantumDevice` subclass models graph-based qubit topologies to represent connectivity or coupling maps for superconducting devices. A specialized subclass, `IBM_QuantumDevice`, further refines the model to capture IBM quantum hardware attributes, including Circuit Layer Operations Per Second (`CLOPS`) [41] and error scores derived from calibration data.

The **`Broker`** acts as the intermediary between job requests and available quantum devices, managing device selection, resource allocation, and job execution strategies. By extending the abstract `Broker` class, users may create a `CustomBroker` to implement custom algorithms and optimize scheduling according to specific objectives.

A quantum job, or **`QJob`**, encapsulates attributes and behaviors representing a quantum task. Each `QJob` instance includes several parameters: `job_id`, a unique identifier; `num_qubits`, the maximum number of qubits required; `depth`, the circuit depth; `num_shots`, the number of repetitions per measurement; and `arrival_time`, the job's arrival time. `QJob` serves as an abstraction for simulating job scheduling and resource allocation on quantum devices. In this work, each `QJob` is assumed to contain a single quantum circuit.

The **`JobGenerator`** component produces `QJobs` through predefined dispatching mechanisms. For deterministic simulations, jobs can be loaded from external CSV files that specify fields such as job ID, number of shots, arrival time, circuit depth, and qubit requirements. The `JobGenerator` reads the data and schedules `QJobs` to arrive at the designated times. If no arrival time is specified, the current timestamp is assigned by default. This deterministic mode supports benchmarking, debugging, and comparative performance analysis under controlled conditions.

The **`JobRecordsManager`** component tracks the lifecycle of quantum jobs, logs key events, and maintains records of system activity. It monitors job-related events, including `arrival`, `start`, `finish`, and `fidelity`, which provide data for analyzing performance metrics such as wait times, execution durations, and throughput.

Calibration data provides real-time information about the performance of a quantum processor, including qubit coherence times, gate fidelities, readout errors, and other hardware metrics. These parameters are critical for assessing the reliability of quantum computations.

Together, these components define the framework's ecosystem. As shown in Fig. 4, the framework integrates multiple modules to manage the generation, allocation, execution, and analysis of quantum jobs (`QJobs`). Jobs can originate from various sources, including standardized benchmarks, synthetic datasets, or user-defined models. The central component, `QCloudSimEnv`, coordinates job flow through its modular subcomponents. The `JobGenerator` creates `QJobs` with defined circuits and metadata, including resource requirements and execution parameters. These jobs are passed to the `Broker`, which schedules them based on customizable allocation policies and dispatches them to quantum devices within the `QCloud`. Each `QuantumDevice` is configured with specific characteristics, such as qubit connectivity and gate fidelity, and maintains a local queue for job execution. Allocation policies determine how jobs are distributed across devices to optimize system performance. The `JobRecordsManager` collects simulation results, capturing execution metrics such as QPU time, wait time, and fidelity. These records can be used in post-simulation analysis to evaluate and refine quantum cloud resource management strategies.

4 PROBLEM DEFINITION

Efficient allocation of large quantum jobs across multiple quantum processors is a central challenge in distributed quantum computing. This problem is shaped by limited qubit capacity, hardware variability, and fidelity loss incurred during inter-device communication. We formally define the allocation problem and present a reinforcement learning-based approach to address it. Firstly, we consider a quantum cloud comprised of a set of quantum processing units (QPUs), denoted as

$$\mathcal{D} = \{D_1, D_2, \ldots, D_{|\mathcal{D}|}\}.$$

Each device $D_i \in \mathcal{D}$ is characterized by the tuple

$$D_i = \big(C_i,\ E_i,\ L_i,\ G_i\big),$$

where $C_i \in \mathbb{N}$ is the qubit capacity (i.e., the number of available qubits, $E_i \in [0,1]$ is the device's error score (e.g., a combination of single- and two-qubit error rates), $L_i \in \mathbb{R}^+$ is the device throughput in circuit layer operations per second (CLOPS) [41], and $G_i = (V_i, E_i')$ is the qubit connectivity graph, with $|V_i| = C_i$.

A *job* J arriving to the cloud is defined by

$$J = \big(q,\ d,\ s,\ t_2\big),$$

where $q \in \mathbb{N}$ is the total number of qubits required, $d \in \mathbb{N}$ is the circuit depth, $s \in \mathbb{N}$ is the number of shots to execute, and $t_2 \in \mathbb{N}$ is the number of two-qubit gates.

An *allocation* of J to a subset of k devices is a vector

$$\mathbf{a} = \big(a_1, a_2, \ldots, a_k\big) \quad \text{such that} \quad \sum_{i=1}^{k} a_i = q \quad \text{and} \quad a_i \le C_{\pi(i)},$$

where $\{\pi(1), \ldots, \pi(k)\} \subseteq \{1, \ldots, |\mathcal{D}|\}$ indexes the selected devices. We further require that each device $D_{\pi(i)}$ contains a connected subgraph of size a_i, i.e.,

$$\exists\, S_i \subseteq V_{\pi(i)},\ |S_i| = a_i,\ G_{\pi(i)}[S_i] \text{ is connected.}$$

Once allocated, the job incurs **Processing time** and **Device fidelity**.

Processing time on device $D_{\pi(i)}$ is defined as

$$\tau_i = \frac{M\,K\,s\,\log_2(QV_{\pi(i)})}{L_{\pi(i)}}, \tag{1}$$

where M and K are constants capturing IBM-specific metrics, $s \in \mathbb{N}$ denotes the number of shots to execute, $QV_{\pi(i)}$ is the quantum volume of the device $D_{\pi(i)}$, and $L_i \in \mathbb{R}^+$ is the circuit layer operations per second (CLOPS) [41].

The overall job runtime is $\tau(\mathbf{a}) = \max_{i=1,\ldots,k} \tau_i$. Device fidelity on $D_{\pi(i)}$ is defined as

$$F_i = \big(1-\bar{\epsilon}_{1q}\big)^d \times \big(1-\bar{\epsilon}_{ro}\big)^{\sqrt{\frac{q}{k}}} \times \big(1-\bar{\epsilon}_{2q}\big)^{\sqrt[4]{\frac{t_2}{k}}},$$

where $\bar{\epsilon}_{1q}, \bar{\epsilon}_{ro}, \bar{\epsilon}_{2q}$ are the device's average single-qubit, readout, and two-qubit errors, respectively.

Devices must then exchange classical and quantum data; each of the $k-1$ inter-device links imposes a communication penalty

$$P = \beta^{k-1}, \quad \beta \in (0,1),$$

so that the *final job fidelity* is

$$F(\mathbf{a}) = \Big(\tfrac{1}{k}\sum_{i=1}^{k} F_i\Big) \times P.$$

Given a job J and the device set $\mathcal{D}$, our goal is to find an allocation $\mathbf{a}$ and a choice of devices $\{\pi(1), \ldots, \pi(k)\}$ that simultaneously

$$\max_{\mathbf{a},\,\pi(\cdot)} F(\mathbf{a}) \quad \text{and} \quad \min_{\mathbf{a},\,\pi(\cdot)} \tau(\mathbf{a}),$$

subject to the capacity and connectivity constraints above.

Since this work focuses on large-scale quantum circuits, the jobs are prepared with certain size of circuits. The constraint for required qubits for a job i is expressed as:

$$\max(|C|_1, |C|_2, \ldots, |C|_N) < q_i < \sum_{j=1}^{N} |C|_j.$$

In that way, we make sure that all circuits are large enough to require partitioning and allocating across multiple quantum devices while remaining small enough to fit within the total available qubits in the quantum cloud $\mathbb{Q}$.

4.1 Reinforcement Learning Formulation

To address the complexity of allocating large quantum jobs across quantum devices, we formulate the problem as a Markov Decision Process (MDP) and implement a reinforcement learning (RL) agent using Proximal Policy Optimization (PPO). The environment is modeled using the Gymnasium API with the following components:

State: A continuous vector consisting of normalized job and device features. Specifically:

- Job parameters: normalized qubit count $q/q_{\max}$, where $q_{\max} = 50$.
- Device features for each of the $k = 5$ devices (padded with zeros if fewer): normalized container level $C_i/150$, error score E_i, and normalized CLOPS $K_i/10^6$.

The resulting state vector has a dimensionality of $1 + 3k = 16$.

Action: A continuous vector $\mathbf{a} = [a_1, \ldots, a_k]$ representing unnormalized allocation weights. The final allocation is computed as:

$$\hat{a}_i = \frac{a_i}{\sum_j a_j + \epsilon} \cdot q$$

followed by rounding and adjustment to ensure $\sum_i \hat{a}_i = q$.

Reward: The scalar reward is the average circuit fidelity achieved across the allocated devices:

$$R = \frac{1}{k'}\sum_{i=1}^{k'} F_i$$

where k' is the number of devices used, and F_i is the fidelity on device i, incorporating gate error, readout error, and (optionally suppressed) two-qubit error.

The RL agent is trained in simulation by generating randomized jobs and interacting with the environment over single-step episodes. Once trained, the policy network is deployed to produce allocation decisions in simulation: given a new job and the current system state, the model outputs allocation ratios, which are then used to request and reserve resources across multiple QPUs. After quantum job execution, inter-device communication is simulated, and final

fidelity is computed with a communication penalty. This approach provides adaptive, noise-aware allocation in distributed quantum systems.

5 ALLOCATION STRATEGIES

Our framework supports four allocation modes that share a unified scheduling workflow. For each incoming quantum job, the scheduler selects a subset of devices, partitions the job accordingly, and orchestrates parallel execution with inter-device communication when necessary. The primary distinction among the allocation modes lies in the *device selection policy*, which determines which devices are chosen for execution.

5.1 Allocation Workflow

Algorithm 1 summarizes the unified allocation workflow shared across all scheduling modes. Once devices are selected using one of the policies, the subsequent steps — including qubit partitioning, parallel execution, inter-device communication, and fidelity computation — follow a common execution path.

Algorithm 1 Unified Allocation Workflow

1: **Input:** Quantum job J, set of available devices D
2: **Output:** Allocation and execution of J across selected devices
3: Select scheduling (Speed-based, Error-aware, Fair, RL-based)
4: Identify candidate devices $D_{\text{selected}} \subseteq D$ based on scheduling policy
5: Partition job J into sub-jobs according to the number and capacity of D_{selected}
6: **for** each device d in D_{selected} **do**
7: Request and reserve necessary qubits from d
8: Execute the assigned sub-job on d
9: **end for**
10: **for** each pair of devices with dependent sub-jobs **do**
11: Perform classical communication to synchronize results
12: **end for**
13: Compute final fidelity, apply communication penalties if needed
14: Release qubits and log completion of J

Speed-based Scheduling. This policy prioritizes minimizing total execution time by selecting devices with the fastest processing capability, without considering noise levels.

Fidelity-optimized Scheduling. This policy aims to maximize circuit fidelity by selecting devices with the lowest gate and readout errors based on calibration data. The calculation of the error score is detailed in Section 5.4. Selected devices are sorted to prioritize those with lower error scores.

Fair Scheduling. This policy balances load by selecting devices with the lowest current utilization, aiming to prevent resource contention and evenly distribute workloads.

Reinforcement Learning Scheduling. This policy leverages a trained reinforcement learning agent to select devices based on real-time system state observations. The RL agent outputs allocation fractions that reflect learned trade-offs between device availability, error rates, and workload distribution.

5.2 Qubit Partitioning and Allocation

Once devices are selected, the job's qubits are partitioned and assigned to available devices. Ideally, the subset of allocated qubits on each device forms a connected subgraph within the device's qubit topology graph $G(V, E)$. However, finding an optimal connected subgraph is computationally intractable due to the combinatorial explosion of possible subgraphs.

For example, selecting 10 qubits out of 127 requires searching over $C(127, 10) \approx 209$ trillion combinations. To avoid prohibitive computational costs, our implementation adopts a simplified black-box abstraction that assumes allocated qubits form a connected subgraph, which is a practical assumption for devices with high connectivity. This approach enables fast and scalable allocation while focusing on evaluating broader scheduling performance.

5.3 Complexity Analysis

The computational complexity of the allocation workflow is primarily determined by the device selection and job partitioning steps. Device selection requires evaluating all available devices, resulting in a complexity of $O(n)$, where n is the number of candidate devices. The job partitioning step divides the job across the selected devices, which operates in linear time with respect to the number of selected devices, yielding a complexity of $O(m)$, where m is the number of selected devices. Execution and communication steps are simulated in an event-driven fashion and do not introduce significant overhead in the scheduling process. Therefore, the overall complexity of the algorithm is dominated by device evaluation and selection, and can be expressed as $O(n + m)$. This makes the workflow efficient and scalable for quantum cloud environments, where the number of devices is typically moderate.

5.4 Error Score

Calibration data provides real-time information about the performance of a quantum processor, including qubit coherence times, gate fidelities, readout errors, and other hardware metrics. Based on this data, we define an *error score* to quantify the overall device quality by combining readout errors, single-qubit gate errors, and two-qubit gate errors through a weighted formula:

$$\text{error\_score} = \alpha \cdot \frac{\sum_i \epsilon_{\text{ro},i}}{C} + \theta \cdot \epsilon_{1\text{q}} + \gamma \cdot \frac{\sum_j \epsilon_{2\text{q},j}}{t_2} \tag{2}$$

where:

- $\epsilon_{\text{ro},i}$ represents the readout error for qubit i, and C is the number of qubits in the device.
- $\epsilon_{1\text{q}}$ is the error rate of the single-qubit RX gate.
- $\epsilon_{2\text{q},j}$ denotes the two-qubit gate error for gate j, and t_2 is the total number of two-qubit gates.

The weighting factors α, θ, and γ are assigned values of 0.5, 0.3, and 0.2, respectively. Readout errors are assigned the highest weight because they directly impact the correctness of measurement outcomes, making them especially critical to the fidelity of quantum computations. Single-qubit and two-qubit gate errors are weighted lower, with single-qubit errors receiving slightly higher weight. This reflects the observation that although two-qubit gates tend to have higher individual error rates, they may appear less frequently depending on circuit structures, while single-qubit gates

are more common. This balance ensures that both the severity of errors and the frequency of operations are appropriately considered. The weighting scheme follows conventions in prior quantum error characterization studies [20] and can be adjusted as necessary for different quantum workloads.

6 PERFORMANCE METRICS

In this work, we implemented and tested allocation algorithms by evaluating their performance using three metrics: execution time, fidelity, and communication overhead.

6.1 Processing Time

The processing time (τ) of a quantum job is computed based on the Circuit Layer Operations Per Second (CLOPS) and quantum volume (QV), metrics established by IBM [41]. CLOPS is a benchmark that measures the speed at which a quantum processor executes quantum circuits. QV evaluates the capability of a quantum system by considering factors such as qubit count, connectivity, and gate fidelity, and determines the largest circuit depth a device can reliably handle. The processing time is calculated by using Eq. (1).

The numerator, $M\,K\,s\,log_2(QV)$, in Eq. (1) represents the total computational workload, accounting for the number of circuits, qubits, iterations, and circuit depth. The denominator, L or ($CLOPS$), is a normalization factor, reflecting the quantum processor's execution speed. For instance, consider a quantum job with the following parameters: $M = 100$, $K = 10$, $s = 40{,}000$, and $log_2(QV) = 7$ (circuit depth). The values for M and K are referenced from [41]. With `ibm_brussels` QPU, which has a CLOPS rating of 220,000, the estimated execution time for this job applying Eq. (1) is approximately 21 minutes.

6.2 Fidelity

We calculate fidelity based on the calibration data provided by IBM. Error sources include single-qubit gate errors, two-qubit gate errors, and readout errors. We estimate the overall fidelity using the following formulation:

1. Single-Qubit Fidelity: Calculated from single-qubit error, which arises during the execution of single-qubit gates, such as Pauli-X or Hadamard gates, the single-qubit fidelity is estimated as:

$$F_{1\mathrm{q}} = (1 - \bar{\epsilon}_{1\mathrm{q}})^{d} \tag{3}$$

where $\bar{\epsilon}_{1\mathrm{q}}$ is the average error rate of single-qubit gates, and d is the depth of the quantum circuit.

This follows the assumption that single-qubit gate errors compound independently over multiple gate applications, as described in quantum error models [24].

2. Two-Qubit Fidelity: The execution of two-qubit gates, such as CNOT or CZ gates, also gives rise to errors. From those errors, the two-qubit fidelity is estimated as:

$$F_{2\mathrm{q}} = (1 - \bar{\epsilon}_{2\mathrm{q}})^{\sqrt[4]{\frac{t_2}{k}}}, \tag{4}$$

where $\bar{\epsilon}_{2q}$ is the error rate for two-qubit gates that belong to the connectivity of two qubits in graph G, t_2 is the number of two-qubit gates in the circuit, and k is the number of devices utilized. Equation (4) computes the overall two-qubit fidelity as the product of individual gate survival probabilities, assuming independent errors for each two-qubit gate in the connectivity graph.

3. Readout Fidelity: Readout errors are caused by imperfect qubit measurement due to detector inefficiency or thermal condition of the device [26]. Readout fidelity accounts for measurement errors and is computed as:

$$F_{\mathrm{readout}} = (1 - \bar{\epsilon}_{\mathrm{ro}})^{\sqrt{\frac{q}{k}}} \tag{5}$$

where $\bar{\epsilon}_{\mathrm{ro}}$ is the average readout error per qubit, q is the number of qubits in the circuit, and k is the number of devices utilized.

This exponent scaling moderates the fidelity degradation compared to naive linear models, yielding more realistic estimates [20].

6.3 Overall Fidelity Estimation

The fidelity of a job i on each quantum device, F_{i}, is computed as the product of single-qubit, two-qubit, and readout fidelities, as shown in Equation (6):

$$F_{\mathrm{dev}} = F_{1\mathrm{q}} \times F_{2\mathrm{q}} \times F_{\mathrm{readout}} \tag{6}$$

In multi-device scenarios, additional fidelity loss arises due to inter-device communication overhead.

6.4 Inter-Device Communication Penalty

Quantum jobs partitioned across multiple QPUs require classical communication of intermediate measurement outcomes or quantum state parameters. Due to technological limitations, our current model assumes classical communication of intermediate measurement outcomes between devices. To account for this, we introduce a communication penalty factor ϕ:

$$F_{\mathrm{final}} = \bar{F}_{\mathrm{dev}} \times \phi^{(N\mathrm{devices}-1)} \tag{7}$$

Here, $\phi = 0.95$ per inter-device connection follows empirical fidelity degradation observed experimentally in hybrid quantum computing setups. The variable $\bar{F}_{\mathrm{dev}}$ denotes the average fidelity across associated quantum devices, and Ndevices is the total number of interconnected quantum devices.

This simplified penalty-based model reflects typical fidelity degradation caused by classical communication latency, control synchronization issues, and inter-device calibration differences. Although simplified, this approach aligns with similar models employed in contemporary quantum computing research [40], where fidelity degradation is empirically estimated.

6.5 Communication Overhead Modeling

Distributed quantum computing requires frequent classical communication, particularly when qubit measurements or classical control information must be transmitted between quantum processors during algorithm execution. Our model focuses specifically on classical communication latency, which is critical in current distributed quantum cloud architectures.

We model this classical communication overhead as proportional to the number of qubits involved and an experimentally motivated per-qubit latency parameter:

$$\tau_{\mathrm{comm}} = N_{\mathrm{qubits}} \cdot \lambda \tag{8}$$

where N_{qubits} is the number of qubits whose measurement outcomes or classical control parameters are communicated between devices, and λ is the per-qubit classical communication latency. The value is set to 0.02 seconds per qubit. It can vary based on existing networked quantum computer deployments.

Equation (8) adapts the classical communication complexity model [42], explicitly focusing on classical data transfer between quantum devices. Communication is modeled as a blocking operation, delaying job execution by τ_{comm} before resuming computations.

This simplified latency-based model captures the essence of classical communication constraints prevalent in current experimental setups [40]. However, it does not explicitly model more advanced quantum communication techniques such as entanglement swapping or teleportation, which remain experimental and challenging to implement at scale.

6.6 RL Policy Training

To implement the *RL-based* allocation strategy, we trained a reinforcement learning policy using the Proximal Policy Optimization (PPO) algorithm. The training environment, `QCloudGymEnv`, was constructed to simulate real-world quantum cloud scheduling scenarios, with five IBM quantum processors (Strasbourg, Brussels, Kyiv, Québec, and Kawasaki) initialized using calibration data collected in March 2025. The objective of the reinforcement learning agent is to allocate quantum jobs to these devices so as to maximize the expected circuit fidelity while respecting device constraints.

The training process was conducted over 100,000 timesteps. Fig. 5 illustrates the training progress, showing the relationship between average episode reward and entropy loss. The average episode reward, which correlates with job fidelity, exhibits a steady improvement during the early stages of training and begins to plateau around 0.70 as the model converges. Concurrently, entropy loss gradually decreases from approximately −7 to −2, indicating the agent's transition from exploration to exploitation as it learns more deterministic allocation policies.

The PPO agent was trained using a multi-layer perceptron (MLP) policy, due to its suitability for low-dimensional, structured input spaces and its efficiency in learning single-step allocation decisions without temporal dependencies. The environment terminates each episode after a single allocation decision, making the problem a single-step decision-making task. As observed in the training curve, the learning process stabilized after around 40,000 to 50,000 timesteps, demonstrating that the agent was able to discover effective allocation strategies that yield consistently high fidelity across distributed devices.

Overall, the training results suggest that reinforcement learning is capable of automatically learning device allocation policies that balance fidelity and resource utilization without the need for hand-tuned heuristic rules. Further performance gains could potentially be achieved with extended training or communication-aware reward shaping, which are left as future work.

7 CASE STUDY

In this section, we analyze the performance of four allocation modes. We evaluate performance metrics and identify the trade-offs among

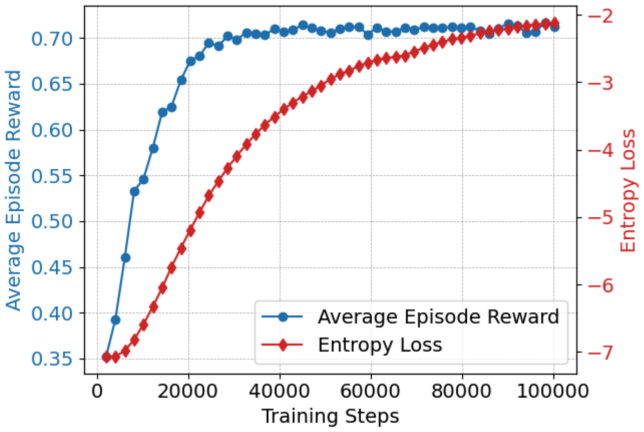


Figure 5: PPO training progress showing average episode reward (left y-axis) and entropy loss (right y-axis) over training steps.

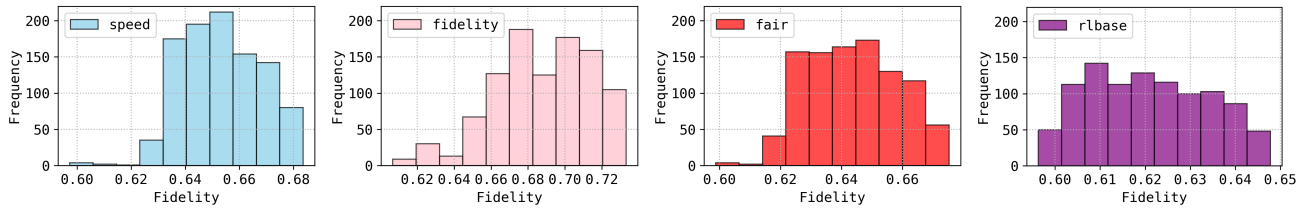
them. For this experiment, 1,000 synthetic jobs were generated. Each job requires between 130 and 250 qubits, has a circuit depth ranging from 5 to 20, and the number of shots between 10,000 and 100,000. The number of shots is chosen within a reasonable range to reflect typical execution settings in current quantum experiments. These synthetic jobs were specifically designed to necessitate splitting across multiple devices, thus emphasizing distributed execution. This method of intentionally creating jobs that exceed individual device capacities is relatively new, and as such, standard benchmarks for such large-scale, distributed quantum jobs have not yet been established in existing literature. The gate sets used in these jobs are abstracted to the number of single-qubit and two-qubit gates, without specifying explicit gate types. The jobs were tested using five simulated IBM quantum devices. All devices have 127 qubits and quantum volumes of 127. Among these devices, the QPUs *ibm_strasbourg* and *ibm_brussels* have the highest CLOPS (220,000), while *ibm_quebec*, *ibm_kawasaki*, and *ibm_kyiv* have CLOPS values of 32,000, 29,000, and 30,000, respectively.

Mode	T_{sim} (s)	$\mu_F \pm \sigma_F$	T_{comm} (s)
speed	108 775.38	0.65332 ± 0.01438	5 707.80
fidelity	209 873.02	0.68781 ± 0.02605	3 822.74
fair	108 978.16	0.64373 ± 0.01478	5 685.30
rlbase	106 206.21	0.62087 ± 0.01301	6 105.52

Table 2: Performance of allocation strategies on 1,000 large circuits.

The performance of each strategy assessed in Table 2 is using the following metrics:

- **Total simulation time** T_{sim}: wall-clock time until all jobs complete (in seconds).
- **Average fidelity** $\mu_F \pm \sigma_F$: mean and standard deviation of final circuit fidelities.
- **Communication time** T_{comm}: total inter-device communication delay summed over all jobs (in seconds).



(a) Speed-Optimized (b) Fidelity-Optimized (c) Fair Allocation (d) RL-Based Allocation

Figure 6: Fidelity distributions of quantum jobs under four allocation strategies in the simulated quantum cloud environment. Each strategy demonstrates distinct fidelity patterns, reflecting trade-offs between speed, fidelity, fairness, and reinforcement learning-based adaptive allocation.

Speed vs. Fidelity. The *speed* strategy minimizes simulation time (1.09×10^5 s) by aggressively splitting jobs across all available QPUs, achieving moderate fidelity (0.6533) at the cost of high communication overhead (5.7 ks). In contrast, the *fidelity* strategy prioritizes low-error devices, achieving the highest fidelity (0.6878), but with significantly longer runtime (2.10×10^5 s). Its communication overhead is also lower (3.8 ks), indicating less fragmentation.

Fair Allocation. The *fair* strategy distributes workload evenly across devices, resulting in identical runtime to the *speed* policy (1.09×10^5 s) and slightly lower fidelity (0.6438). This suggests that equal partitioning may overlook hardware variability, underutilizing high-fidelity QPUs.

RL-Based Strategy. The *rlbase* policy, trained via PPO to maximize job fidelity, yields the shortest runtime (1.00×10^5 s) but the lowest fidelity (0.6209) and highest communication overhead (6.3 ks). The flat fidelity distribution indicates that the learned strategy still explores allocations that cause excessive inter-device interaction, reducing fidelity despite faster job turnover.

7.1 Fidelity Distributions

The impact of each allocation strategy on the resulting job fidelity is illustrated in Fig. 6. The *Fair Allocation* and *Speed-Optimized* strategies produce relatively narrow distributions, with fidelities concentrated around 0.65, indicating more deterministic but suboptimal fidelity outcomes. In contrast, the *Fidelity-Optimized* strategy exhibits a right-shifted and bimodal distribution, successfully pushing a significant portion of jobs above 0.66 fidelity, which reflects its explicit focus on error-aware scheduling. Meanwhile, the *RL-Based Allocation* strategy shows a flatter and broader distribution between 0.60 and 0.64. This pattern highlights the reinforcement learning agent's tendency to explore diverse allocation configurations, trading off fidelity for potential adaptability. Overall, these fidelity profiles reveal the inherent trade-offs among speed, fairness, fidelity, and learning-driven allocation modes in distributed quantum job scheduling.

7.2 Discussion

The results highlight a fundamental trade-off in quantum-cloud scheduling between execution efficiency and output quality. The *speed* and *fair* strategies are effective in reducing overall simulation time, but this efficiency comes at the cost of lower fidelity in the resulting quantum circuits. In contrast, the *fidelity*-based strategy prioritizes job quality by assigning workloads to devices with lower error rates, which improves circuit fidelity but leads to increased scheduling delays. The *rlbase* model offers a middle ground by balancing execution time and adaptive exploration. However, its fidelity performance remains limited, suggesting that additional mechanisms—such as communication-aware reward shaping—may be necessary to better guide the learning process in distributed quantum environments.

Future enhancements should explore refined fidelity models, real-time calibration-aware scheduling, and strategies to further minimize communication overhead, potentially leading to more pronounced differences in fidelity outcomes. We also note that all fidelity measurements in this work are based on theoretical estimations derived from reported error rates, qubit usage, and circuit depth. If industrial or experimental data becomes available, it could be used to calibrate our fidelity model and validate the simulation workflow.

8 CONCLUSION

In this work, we developed a simulation framework to explore quantum circuit execution across distributed quantum processors connected via real-time classical communication. Building on the architectural model proposed by Vazquez et al. [40], our framework supports the coordinated scheduling of circuits that exceed the capacity of a single QPU, reflecting the anticipated shift toward multi-device execution in cloud-based quantum computing. We implemented and evaluated four distinct allocation strategies—*Speed*, *Fidelity*, *Fair*, and *Reinforcement Learning*—to examine trade-offs between execution time, circuit fidelity, and load distribution. The results demonstrate how different scheduling policies can impact system performance and workload quality. As quantum hardware continues to advance, our framework becomes instrumental in simulating and analyzing scalable quantum job allocation strategies in distributed settings.

The source code and integrated data used for the simulation experiments in this study are publicly accessible in the GitHub and Zenodo repositories [35].

ACKNOWLEDGMENTS

This work was partially sponsored by NSF 2230111, 2238734, and 2311950.

REFERENCES

[1] H. Bian, J. Huang, J. Tang, R. Dong, L. Wu, and X. Wang. 2023. PAS: A new powerful and simple quantum computing simulator. *Software: Practice and Experience* 53, 1 (2023), 142–159.

[2] John Brennan, Lee J O'Riordan, K. G. Hanley, Myles Doyle, Momme Allalen, David Brayford, Luigi Iapichino, and Niall Moran. 2022. QXTools: A Julia framework for distributed quantum circuit simulation. *J. Open Source Softw.* 7 (2022), 3711. https://api.semanticscholar.org/CorpusID:246886257

[3] Tim Coopmans, Robert Knegjens, Axel Dahlberg, David Maier, Loek Nijsten, Julio de Oliveira Filho, Martijn Papendrecht, Julian Rabbie, Filip Rozpędek, Matthew Skrzypczyk, Leon Wubben, Walter de Jong, Damian Podareanu, Ariana Torres-Knoop, David Elkouss, and Stephanie Wehner. 2020. NetSquid, a NETwork Simulator for QUantum Information using Discrete events. *Communications Physics* 4 (2020), 1–15. https://api.semanticscholar.org/CorpusID:235967111

[4] Erik P. DeBenedictis. 2018. A Future with Quantum Machine Learning. *Computer* 51, 2 (2018), 68–71. https://doi.org/10.1109/MC.2018.1451646

[5] Simon J. Devitt. 2016. Performing quantum computing experiments in the cloud. *Phys. Rev. A* 94 (Sep 2016), 032329. Issue 3. https://doi.org/10.1103/PhysRevA.94.032329

[6] S. Diadamo, J. Notzel, B. Zanger, and M. M. Bese. 2021. QuNetSim: a software framework for quantum networks. *IEEE Transactions on Quantum Engineering* 2 (2021), 1–12.

[7] Daniel J Egger, Claudio Gambella, Jakub Marecek, Scott McFaddin, Martin Mevissen, Rudy Raymond, Andrea Simonetto, Stefan Woerner, and Elena Yndurain. 2020. Quantum computing for finance: State-of-the-art and future prospects. *IEEE Transactions on Quantum Engineering* 1 (2020), 1–24.

[8] Abhinav Kandala et al. 2019. Error mitigation extends the computational reach of a noisy quantum processor. *Nature* 567 (2019), 491–495. https://doi.org/10.1038/s41586-019-1040-7

[9] David C. McKay et al. 2019. Qiskit Backend Specifications for OpenQASM and OpenPulse Experiments. *arXiv* (2019). arXiv:arXiv:1809.03452 https://arxiv.org/abs/1809.03452

[10] Jay M. Gambetta et al. 2017. Building logical qubits in a superconducting quantum computing system. *Nature Physics* 13 (2017), 1050–1056. https://doi.org/10.1038/nphys4118

[11] Y. Fan and Z. Lan. 2021. DRAS-CQSim: A reinforcement learning based framework for HPC cluster scheduling. *Software Impacts* 8 (may 2021), 100077.

[12] Richard P Feynman. 2018. Simulating physics with computers. In *Feynman and computation*. cRc Press, 133–153.

[13] Paul Griffin and Ritesh Sampat. 2021. Quantum Computing for Supply Chain Finance. In *2021 IEEE International Conference on Services Computing (SCC)*. 456–459. https://doi.org/10.1109/SCC53864.2021.00066

[14] S. N. Agos Jawaddi and A. Ismail. 2024. Integrating OpenAI Gym and CloudSim Plus: A simulation environment for DRL agent training in energy-driven cloud scaling. *Simulation Modelling Practice and Theory* 130 (jan 2024), 102858.

[15] J.R. Johansson, P.D. Nation, and Franco Nori. 2013. QuTiP 2: A Python framework for the dynamics of open quantum systems. *Computer Physics Communications* 184, 4 (2013), 1234–1240. https://doi.org/10.1016/j.cpc.2012.11.019

[16] Tyson Jones, Anna Brown, Ian Bush, and Simon C Benjamin. 2019. QuEST and High Performance Simulation of Quantum Computers. *Sci Rep* 9, 1 (2019), 10736.

[17] Frank Leymann, Johanna Barzen, Michael Falkenthal, Daniel Vietz, Benjamin Weder, and Karoline Wild. 2020. Quantum in the Cloud: Application Potentials and Research Opportunities. In *International Conference on Cloud Computing and Services Science*. https://api.semanticscholar.org/CorpusID:212717763

[18] Lei Liu and Xinglei Dou. 2021. Qucloud: A new qubit mapping mechanism for multi-programming quantum computing in cloud environment. In *2021 IEEE International symposium on high-performance computer architecture (HPCA)*. IEEE, 167–178.

[19] Waylon Luo, Betis Baheri, Travis Humble, Jiapeng Zhao, Tong Zhan, Rajan Maharjan, and Qiang Guan. 2025. A Digital Twin of Scalable Quantum Clouds. In *39th ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. 165–175. https://doi.org/10.1145/3726301.3732296

[20] Easwar Magesan, Jay M Gambetta, and Joseph Emerson. 2012. Characterizing quantum gates via randomized benchmarking. *Physical Review A* 85, 4 (2012), 042311. https://doi.org/10.1103/PhysRevA.85.042311

[21] N. David Mermin. 2007. *Quantum Computer Science: An Introduction*. Cambridge University Press.

[22] Hoa T. Nguyen, Muhammad Usman, and Rajkumar Buyya. 2023. iQuantum: A Case for Modeling and Simulation of Quantum Computing Environments. In *2023 IEEE International Conference on Quantum Software (QSW)*. 21–30. https://doi.org/10.1109/QSW59989.2023.00013

[23] Hoa T Nguyen, Muhammad Usman, and Rajkumar Buyya. 2024. QSimPy: A Learning-centric Simulation Framework for Quantum Cloud Resource Management. *arXiv preprint arXiv:2405.01021* (2024).

[24] Michael A. Nielsen and Isaac L. Chuang. 2010. *Quantum Computation and Quantum Information* (10th anniversary edition ed.). Cambridge University Press.

[25] Xiangyu Niu, Jianwei Li, Qing Wang, Jialin Yang, and Lei Liu. 2024. QuMC: Hardware-Aware Multi-Programming Compiler for Quantum Computing. *Computers, Materials & Continua* 79, 2 (2024), 1961–1974.

[26] NVIDIA. 2025. Readout Error Mitigation in CUDA Quantum. https://nvidia.github.io/cuda-quantum/latest/applications/python/readout_error_mitigation.html Accessed: 2025-03-06.

[27] John Preskill. 2018. Quantum Computing in the NISQ era and beyond. *Quantum* 2 (Aug. 2018), 79. https://doi.org/10.22331/q-2018-08-06-79

[28] Gokul Subramanian Ravi, Kaitlin N. Smith, Pranav Gokhale, and Frederic T. Chong. 2022. Quantum Computing in the Cloud: Analyzing job and machine characteristics. arXiv:2203.13121 [quant-ph]

[29] Subramaniam Ravi, Amandeep Singh, and Ravi Patel. 2023. Adaptive job scheduling framework for quantum cloud environments. *Journal of Quantum Computing* 78 (2023), 123–138.

[30] Rafael Rodrigues, Daniel Chan, Andy Kraft, Brandon Castaneda, et al. 2023. Gymnasium: A Standard API for Reinforcement Learning Environments. https://github.com/Farama-Foundation/Gymnasium. Version 0.29.1.

[31] Priyabrata Senapati, Tushar M. Athawale, David Pugmire, and Qiang Guan. 2023. Advancing Comprehension of Quantum Application Outputs: A Visualization Technique *(QCCC '23)*. Association for Computing Machinery, New York, NY, USA, 25–28. https://doi.org/10.1145/3588983.3596689

[32] Priyabrata Senapati, Samuel Yen-Chi Chen, Bo Fang, Tushar M. Athawale, Ang Li, Weiwen Jiang, Cheng Chang Lu, and Qiang Guan. 2024. PQML: Enabling the Predictive Reproducibility on NISQ Machines for Quantum ML Applications. In *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, Vol. 01. 1413–1424. https://doi.org/10.1109/QCE60285.2024.00168

[33] Priyabrata Senapati, Zhepeng Wang, Weiwen Jiang, Travis S Humble, Bo Fang, Shuai Xu, and Qiang Guan. 2023. Towards Redefining the Reproducibility in Quantum Computing: A Data Analysis Approach on NISQ Devices. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, Vol. 01. 468–474. https://doi.org/10.1109/QCE57702.2023.00060

[34] SimPy. 2024. Discrete event simulation for Python. https://simpy.readthedocs.io/en/latest/index.html.

[35] Quantum Cloud Simulation. 2025. Submission to ICPP 2025. https://github.com/QuantumCloudSimulation/ICPP2025. https://doi.org/10.5281/zenodo.15897774

[36] Damian Steiger, Thomas Häner, and Matthias Troyer. 2016. ProjectQ: An Open Source Software Framework for Quantum Computing. *Quantum* 2 (12 2016). https://doi.org/10.22331/q-2018-01-31-49

[37] Martin Suchara, Andrew Anderson, Harry Buhrman, and Andrew Schreier. 2024. QURE: Quantum Resource Estimator for Cloud Environments. *Computers, Materials & Continua* 79, 2 (2024), 1957–1974.

[38] Ho Lun Tang, Kunal Sharma, Yvette Norambuena, Mario Berta, and Leonard Wossnig. 2021. CutQC: Using small quantum computers for large quantum circuit evaluations. *npj Quantum Information* 7 (2021), 114.

[39] Mohammad Ghazi Vakili, Christoph Gorgulla, AkshatKumar Nigam, Dmitry Bezrukov, Daniel Varoli, Alex Aliper, Daniil Polykovsky, Krishna M Padmanabha Das, Jamie Snider, Anna Lyakisheva, et al. 2024. Quantum computing-enhanced algorithm unveils novel inhibitors for KRAS. *arXiv preprint arXiv:2402.08210* (2024).

[40] Almudena Carrera Vazquez, Caroline Tornow, Diego Ristè, Stefan Woerner, Maika Takita, and Daniel J. Egger. 2024. Combining quantum processors with real-time classical communication. *Nature* (2024). https://doi.org/10.1038/s41586-024-08178-2

[41] Andrew Wack, Hanhee Paik, Ali Javadi-Abhari, Petar Jurcevic, Ismael Faro, Jay Gambetta, and Blake Johnson. 2021. Quality, Speed, and Scale: three key attributes to measure the performance of near-term quantum computers. (10 2021).

[42] Andrew C. Yao. 1979. Some Complexity Questions Related to Distributive Computing. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing (STOC)*. 209–213. https://doi.org/10.1145/800135.804414

[43] Yuqi Zhang, Yuxin Yang, William Martin, Kingsten Lin, Zixu Wang, Cheng-Chang Lu, Weiwen Jiang, Ruth Nussinov, Joseph Loscalzo, Qiang Guan, et al. 2025. Prediction of Protein Three-dimensional Structures via a Hardware-Executable Quantum Computing Framework. *arXiv preprint arXiv:2506.22677* (2025).

[44] Maximillian Zinner, Florian Dahlhausen, Philip Boehme, Jan Ehlers, Linn Bieske, and Leonard Fehring. 2021. Quantum computing's potential for drug discovery: Early stage industry dynamics. *Drug Discovery Today* 26, 7 (2021), 1680–1688.