Journal of Computer Science

Research Article

Active Reset and Scheduling in Fault-Tolerant Quantum-Classical Cloud Computing

Mohamed El-Dosuky

Department of Computer Science, Arab East Colleges, Saudi Arabia
Department of Computer Science, Faculty of Computers and Information, Mansoura University, Egypt

Article history
Received: 12-08-2025
Revised: 03-03-2026
Accepted: 20-05-2026

Email:
maldosuky@arabeast.edu.sa

Abstract: Quantum-classical cloud computing refers to a hybrid computing model that integrates quantum computing with classical computing resources, typically hosted on the cloud. The objective of this paper is threefold: First, to scrutinize the computing paradigm of quantum-classical cloud computing; second, to study the three pillars of the underpinning techniques: Active reset, scheduling, and fault tolerance; third, to demonstrate the realization of the aforementioned techniques. The active reset method leverages measurement and feedback to reset the qubit much faster and with high fidelity. The proposed scheduling algorithm is designed to efficiently manage quantum job execution in a fault-tolerant quantum-classical cloud computing environment. The paper also implements a Bayesian optimization process to optimize the hyper-parameters of a quantum error correction scheme. The research demonstrates significant performance gains across all three technical pillars of the hybrid cloud model. Specifically, the implementation of active reset reduced the qubit initialization time by 90 %, dropping from the 100 μs required for passive relaxation to just 10 μs while maintaining a high reset fidelity of 99.6 %. In terms of cloud resource management, the Shortest Job Next (SJN) scheduling algorithm drastically improved efficiency compared to FCFS, reducing peak wait times from 1.75 s to near-zero for many tasks. Furthermore, the application of Bayesian optimization to faulttolerant parameters successfully minimized the logical error rate from an initial 0.238 to 0.106, marking a 55 % improvement in the reliability of the quantum-classical computing environment.

Keywords: Scheduling, Active Reset, Cloud Computing, Quantum Computing, Fault-Tolerance

Introduction

Quantum-Classical Cloud Computing refers to a hybrid computing model that integrates quantum computing with classical computing resources, typically hosted on the cloud (Golec et al., 2024). This model aims to leverage the strengths of both quantum and classical computing to solve complex problems more efficiently than either can alone.

Various quantum programming platforms have been created based on specific hardware solutions. Key tools include:

- Qiskit: Developed by IBM in 2017, it allows programming at the algorithm and circuit level (Wille et al., 2019)
- Cirq: An open-source framework from Google (Heim et al., 2020)
- PyQuil: Created by Rigetti Computing to run programs on real quantum hardware via cloud services (Hibat-Allah et al., 2024)

Major quantum cloud providers, including Amazon (Braket), Microsoft (Quantum Development Kit), and IBM (Quantum Experience), have begun offering public access to quantum computing services, although the available processing power remains limited (Singh and Bhangu, 2023). There are many toolkits for simulating quantum computing environments, such as iQuantum (Nguyen et al., 2024).

The transition from Noisy Intermediate-Scale Quantum (NISQ) devices to Fault-Tolerant Quantum

Computing (FTQC) is emphasized in John Preskill's Q2B 2023 speech (Preskill, 2023). NISQ devices are beneficial for scientific research, but because of their large error rates, they haven't yet demonstrated quantum benefits that are economically viable. Quantum Error Correction (QEC) is necessary for quantum computing to have an economic impact. However, existing techniques, like as the surface code, have a significant hardware overhead because they require a large number of physical qubits (Fowler et al., 2012). Alternative QEC methods, such as erasure conversion (Scholl et al., 2023), biased noise (Aliferis and Preskill, 2008), and high-rate quantum codes (Fujiwara et al., 2015) are proposed to lessen this.

The objective of this paper is threefold: First, to scrutinize the nascent computing paradigm of quantum-classical cloud computing; second, to study the three pillars of the underpinning techniques: Active reset, scheduling, and fault tolerance; third, to demonstrate the realization of the aforementioned techniques.

Quantum-Classical Cloud Computing

Quantum-classical cloud computing is a hybrid computing paradigm that merges the strengths of quantum and classical processors, accessible through cloud infrastructure (Fan and Han, 2022). In this model, quantum computing is not intended to replace classical systems but to augment them by handling specific tasks that are infeasible or inefficient for classical machines alone. Quantum processors (QPUs), which manipulate information using quantum bits (qubits), are used in tandem with conventional CPUs/GPUs to tackle problems such as optimization, simulation, and machine learning (Rosenfeld et al., 2023). The cloud serves as the bridge between these two domains, allowing seamless offloading of tasks and enabling remote access to quantum hardware from anywhere in the world. This approach opens the door to innovation while abstracting away the complexities of managing quantum systems locally.

Figure 1 illustrates the architecture of quantum-classical cloud computing, showcasing how different system components interact in a hybrid environment. At the top, the user application interfaces with the system through an SDK or API layer, which handles communication with the cloud infrastructure. Within the cloud, a workflow orchestrator and resource scheduler manage the allocation of tasks between classical and quantum resources. Classical computing tasks are executed on CPUs or GPUs, with additional pre- and post-processing handled alongside. Quantum tasks are offloaded to a QPU (Quantum Processing Unit) backend, which includes specialized modules such as a quantum circuit executor and error mitigation layer. Results from quantum computation are returned to the classical processor, integrated, and passed back through the SDK to the user application. This flow highlights the synergy between classical and quantum systems enabled by cloud-based orchestration.

The key value proposition of quantum-classical hybrid computing lies in the synergy between the two types of processors. Quantum systems are exceptionally well-suited to certain mathematical problems, such as matrix manipulations and solving differential equations, which appear frequently in chemistry, finance, and AI. Meanwhile, classical systems remain indispensable for control, data handling, and broader computational logic. Cloud infrastructure enables this symbiosis by orchestrating workflows that divide tasks between quantum and classical environments. Algorithms such as the Variational Quantum Eigensolver (VQE) (Tilly et al., 2022) and Quantum Approximate Optimization Algorithm (QAOA) (Blekos et al., 2024) are early examples of such hybrid applications.

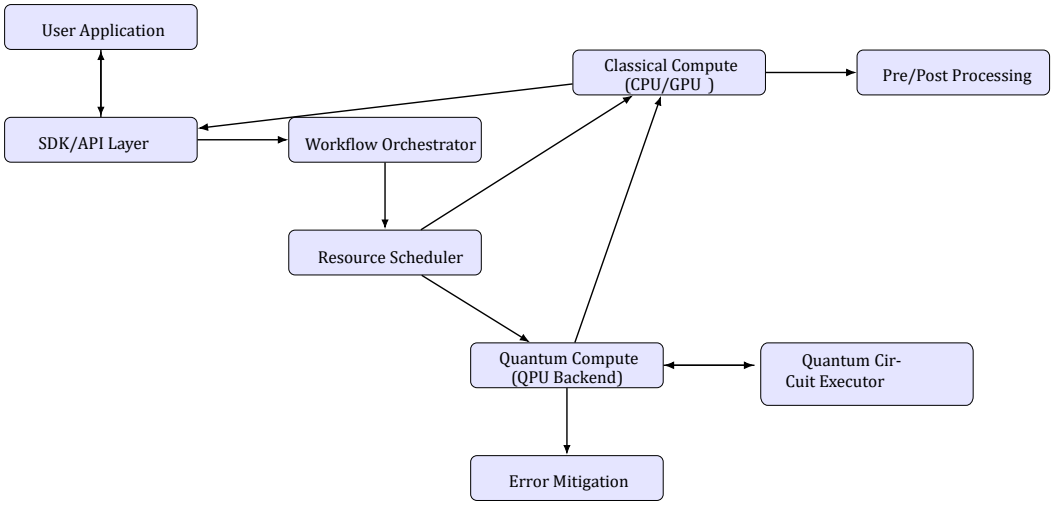

Fig. 1: Quantum-Classical Cloud Computing system

Furthermore, by hosting quantum resources on the cloud, leading companies like IBM, Amazon, Microsoft, and Google are making quantum technology more accessible, democratizing its use and accelerating its development. The growth of SDKs and developer tools like Qiskit, Cirq, Braket SDK, and PennyLane is evidence of a maturing ecosystem around this model.

Despite its promise, the hybrid model faces several fundamental challenges that hinder its widespread practical use. Chief among them is latency. Quantum computers currently operate remotely over the cloud, meaning each quantum-classical interaction involves significant communication overhead. This can seriously disrupt real-time applications and iterative quantum-classical loops. Moreover, today's quantum hardware remains in the NISQ (Noisy Intermediate-Scale Quantum) era (Cheng et al., 2023), characterized by high error rates and a limited number of usable qubits, which restricts the complexity and depth of quantum circuits that can be reliably executed. Another issue is the lack of standardization across platforms: Each provider offers different hardware architectures, gate sets, and programming interfaces, complicating cross-platform development. Lastly, as data travels across networks and into third-party quantum systems, security and privacy concerns become prominent, especially for industries handling sensitive or proprietary data.

Quantum-classical cloud computing is already beginning to reshape the strategies of various high-tech industries. In pharmaceuticals, companies like Pfizer are exploring quantum-enhanced simulations of molecular structures, which could vastly accelerate drug discovery (Raju et al., 2024). Financial institutions, including JP Morgan and Goldman Sachs, are investing in quantum algorithms for portfolio optimization, risk analysis, and fraud detection (Rajagopalan et al., 2024). In materials science and energy, enterprises are applying quantum techniques to simulate chemical reactions and develop new materials (Deglmann et al., 2015). This hybrid model is fostering a shift toward "quantum readiness," where companies build internal expertise, pilot quantum projects, and invest in long-term infrastructure despite the technology's immaturity. The emergence of Quantum-as-a-Service (QaaS) models on the cloud allows businesses to experiment at lower cost and risk, further catalyzing enterprise adoption and creating a competitive landscape where early movers may gain substantial advantages (Barletta et al., 2023). Recently, workforce task execution scheduling using gate-based quantum computers is proposed (Takeori et al., 2024). The use of 127-qubit quantum processors from IBM to address workforce scheduling problems involving between 500 and 874 binary decision variables, along with more than 1,000 constraints.

Methods

Quantum job scheduling is a critical process in quantum computing that ensures efficient execution of quantum jobs on available processors. The provided flowchart in Figure 2 illustrates a systematic approach to scheduling quantum jobs using various algorithms and decision-making processes.

The flowchart begins by checking whether quantum jobs are available. If no jobs are available, the system either visualizes performance metrics or handles errors and retries the job generation. If jobs are available, they are scheduled using one of the following algorithms:

- First-Come, First-Served (FCFS): Jobs are enqueued in the order they arrive and processed sequentially
- Shortest Job Next (SJN): Jobs with the shortest execution time are prioritized
- Round Robin (RR): Jobs are processed in a cyclic manner, ensuring fairness
- Priority Scheduling: Jobs are enqueued based on their priority levels

Once a scheduling algorithm assigns jobs to the queue, a job is selected and assigned to one of the available quantum processors. The execution process consists of assigning the selected job to Processor 1, Processor 2, or Processor 3, executing the job on the designated processor, and logging execution time and wait time for performance evaluation.

After execution, the system checks whether dynamic rescheduling is necessary. If required, the scheduling process is re-initiated to optimize performance. Otherwise, the system continues processing new jobs.

Algorithm 1 proposes active reset and scheduling in FTCQ. It is designed to efficiently manage quantum job execution in a fault-tolerant quantum-classical cloud computing environment. It begins by continuously retrieving jobs from a queue and checking the availability of idle qubits on the quantum device. If sufficient qubits are available, it allocates them to the job and performs an active reset procedure. This involves measuring each qubit and applying conditional reset operations to ensure they are in a known ground state before execution. The algorithm then applies error correction routines and verifies that the system's error rate is below a predefined fault-tolerance threshold. If the error rate is too high, the job is either rescheduled or the qubits are re-initialized. Once the system is prepared, the job is scheduled for execution, with ongoing monitoring and mid-circuit resets applied if supported. After execution, the states of the qubits are updated, allowing the system to continue processing subsequent jobs efficiently and reliably. This process enhances both the performance and stability of quantum computations in noisy and resource-constrained environments.

DOI: 10.3844/jcssp.2026.2254.2265

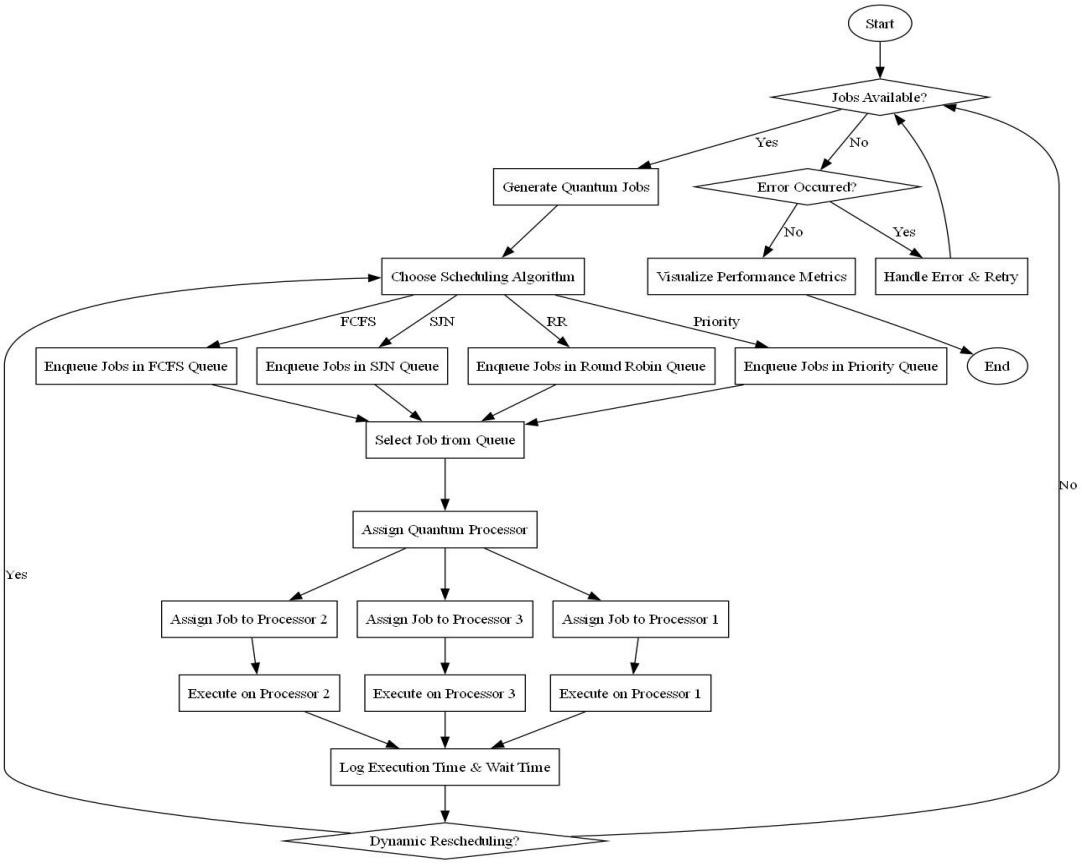

Fig. 2: Quantum Scheduling Flowchart

Algorithm 1 Active Reset and Scheduling in FTQC

Input: Job queue Q, Quantum device D, Fault tolerance threshold θ
Output: Schedule S for execution, reset state of qubits
while True **do**
  $J \leftarrow$ Dequeue next job from Q
  if J is NULL **then**
    break
  end if
  Check qubit availability on device D
  if Idle qubits $<$ Required qubits for J **then**
    Wait or Requeue J
    continue
  end if
  Allocate qubits Q_J for job J
  for each $q \in Q_J$ **do**
    if q is not in ground state **then**
      Apply measurement on q
      Apply conditional reset gate (e.g., X if measured 1)
    end if
  end for
  Apply error correction routines on Q_J
  if Error rate $> \theta$ **then**
    Reinitialize qubits or reassign job
    continue
  end if
  Schedule job J on D
  Append J to execution schedule S
  Monitor execution; apply mid-circuit resets if supported
  Update qubit state post-execution
end while
return S

Algorithm 2 proposes quantum-aware scheduling algorithm which is designed for quantum–classical cloud environments where hardware noise and qubit instability are dominant constraints. For each incoming quantum job, the scheduler extracts resource requirements such as the number of qubits, circuit depth, and the need for mid-circuit measurements. It then evaluates all available quantum devices by jointly considering their physical characteristics, including coherence times (T1, T2), intrinsic error rates, and qubit availability. A reliability score is computed for each device by balancing decoherence risk and hardware error rates. Jobs are assigned to the device that maximizes this reliability score while satisfying a predefined fault-tolerance threshold. Before execution, allocated qubits undergo active reset using measurement and feedback, and if required, mid-circuit measurements with classical feed-forward are enabled. During execution, the system continuously monitors error syndromes and decoherence violations; jobs that exceed the acceptable error threshold are aborted and rescheduled. This strategy ensures that scheduling decisions are driven by quantum hardware fidelity rather than arrival order alone, making it suitable for both NISQ and fault-tolerant quantum cloud systems.

Algorithm 2 Quantum-Aware Scheduling for Fault-Tolerant Quantum Cloud

Require: Job queue $\mathcal{J}$, Quantum devices $\mathcal{D}$, Fault tolerance threshold θ, Decoherence limits (T_1, T_2)
Ensure: Quantum-aware execution schedule $\mathcal{S}$

while $\mathcal{J} \neq \emptyset$ **do**
  $J \leftarrow$ Dequeue job from $\mathcal{J}$
  Extract job profile: required qubits q_J, circuit depth d_J, mid-circuit measurement flag m_J
  for all devices $D_i \in \mathcal{D}$ **do**
    Obtain device parameters: error rate e_i, coherence times (T_1^i, T_2^i), available qubits q_i
    Estimate execution time $\tau_{J,i}$
    Compute decoherence risk:

$$\delta_{J,i} = \frac{\tau_{J,i}}{\min(T_1^i, T_2^i)}$$

    Compute reliability score:

$$R_{J,i} = \alpha(1 - e_i) + \beta(1 - \delta_{J,i})$$

  end for
  Select device $D^* = \arg\max R_{J,i}$
  if $R_{J,D^*} < \theta$ or $q_{D^*} < q_J$ **then**
    Requeue J with increased priority
    continue
  end if
  Allocate qubits on D^*
  Perform active reset using measurement and feedback
  if $m_J =$ **true then**
    Enable mid-circuit measurements
    Apply classical feed-forward corrections
  end if
  Execute quantum circuit of J on D^*
  Monitor error syndromes and decoherence violations
  if Observed error rate $> \theta$ **then**
    Abort execution and requeue J
  else
    Append J to schedule $\mathcal{S}$
  end if
end while
  return $\mathcal{S}$

Active Reset

Active qubit reset on a single qubit is originally proposed in (Karalekas et al., 2020) as a control-flow graph (CFG) (Alfred et al., 2007). This paper instead translated it as a flowchart for a better comprehension. The active reset flowchart, shown in Figure 3, begins execution by initializing the variable c that represents the number of active reset rounds that need to be performed. From there, the control flow transitions to the measurement block M, which is responsible for performing quantum measurement. Each time the flowchart enters block M, it increments a feedback counter k, keeping track of how many reset attempts have occurred. The measurement result at M determines the next step: If the outcome is 0, the flowchart goes to the idle block I, indicating no feedback action is necessary; however, if the result is 1, it proceeds to the feedback block X, where corrective action is taken based on the measurement. After executing either block I or X, the flowchart evaluates whether the total number of feedback rounds performed (k) has reached the preset limit c. If k is still less than c, control returns to block M to perform another measurement-feedback cycle. Once k equals c, indicating all required reset rounds are complete, the program transitions to block U, which marks the start of the main body of the quantum program. This structured loop ensures that the system undergoes a fixed number of feedback-controlled resets before advancing to the core quantum operation.

As shown in Table 1, active reset provides the most balanced solution for quantum cloud environments by achieving fast and high-fidelity qubit initialization without relying on platform-specific dissipation mechanisms, making it particularly suitable for mid-circuit measurements and dynamic scheduling.

Quantum Job Scheduling

Figure 4 shows a class diagram representing the structure of a quantum job scheduling system, outlining the relationships between key components such as the scheduler, quantum jobs, processors, and performance metrics. The Scheduler class is responsible for managing quantum job execution. It maintains a job queue and determines how jobs are assigned to processors based on a scheduling algorithm.

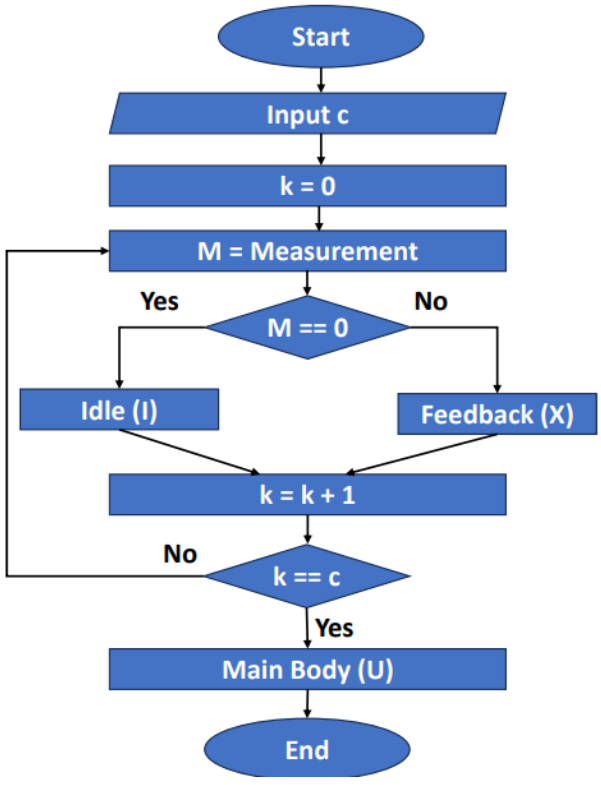

Fig. 3: Active Reset Flowchart

Table 1: Critical Comparison of Qubit Reset Techniques

Reset Method	Reset Speed	Reset Fidelity	Hardware Overhead	Generality
Passive Relaxation	Slow (limited by T_1)	Medium	Minimal	High
Engineered Dissipative Reset	Moderate	Medium–High	Specialized coupling circuits	Platform-dependent
Active Reset (Measurement + Feedback)	Fast	High (readout-dependent)	Classical control & fast readout	High
Algorithmic / Virtual Reset	Instant (logical)	Logical only	None	Algorithm-dependent

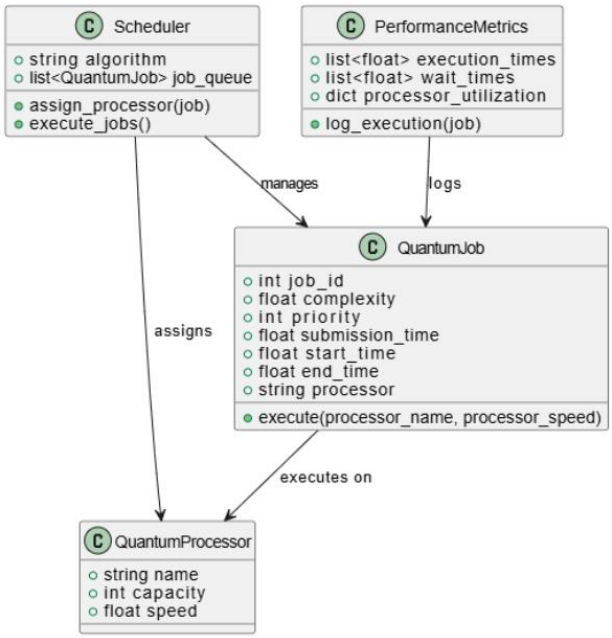

Fig. 4: Scheduling Class Diagram

It has attributes such as algorithm, which defines the scheduling strategy (e.g., FCFS, SJN, RR, Priority Scheduling), and job queue, which stores a list of quantum jobs awaiting execution. The methods include assign processor (job), which allocates a job to an available quantum processor, and execute jobs(), which manages the execution of jobs in the queue.

The Quantum Job class represents a job in the quantum system. It contains attributes such as job id (a unique identifier for the job), complexity (representing the computational complexity), priority (defining the priority level), submission time, start time, and end time (tracking execution timeline), and processor (storing the name of the assigned processor). It also has a method execute (processor name, processor speed), which executes the job on a specified processor.

The Quantum Processor class models a quantum processor available for job execution, with attributes such as name (identifying the processor), capacity (defining processing capacity), and speed (specifying processing speed). The Performance Metrics class logs execution performance and maintains statistics. Its attributes include execution times (storing execution durations for completed jobs), wait times (tracking waiting times before execution), and processor utilization (a dictionary recording processor workloads). The method log execution (job) logs execution details for analysis.

The relationships between these classes are structured for efficient scheduling and execution. The Scheduler manages the job queue and assigns jobs to Quantum Processor instances. The Quantum Job class is assigned to a processor and executes tasks, while the Performance Metrics class logs execution data and performance statistics. This structured approach ensures efficient job allocation and execution tracking. Future improvements could include load balancing strategies and real-time job priority adjustments.

From NISQ to FTQC

The circuit in Figure 5 implements a simple bit-flip quantum error correction protocol. It encodes a logical qubit into three physical qubits using a repetition code, simulates a bit-flip error on one of them, and then detects and corrects the error using two ancilla qubits for syndrome measurement. The ancilla qubits gather parity information to identify which qubit (if any) has flipped, and conditional Pauli-X gates correct the error accordingly. The logical qubit is then decoded, and the final measurement checks whether the original state has been successfully recovered. This demonstrates how quantum redundancy and syndrome-based correction can protect against single-qubit bit-flip errors.

Now, let us visualize two quantum circuits: One representing a Noisy Intermediate-Scale Quantum (NISQ) circuit (Figure 6) and the other illustrating a Fault-Tolerant Quantum Computing (FTQC) circuit (Figure 7). The NISQ circuit runs on a two-qubit system, applying a Hadamard gate followed by a CNOT (to create entanglement), and then an RX rotation on qubit 0-mimicking a shallow, realistic operation suitable for today's NISQ devices with limited error correction. In contrast, the FTQC circuit runs on a five-qubit device and introduces more structured entanglement and redundancy: It uses multiple CNOT and Hadamard gates across data and ancillary qubits to distribute information and potentially detect/correct errors, embodying principles of fault tolerance.

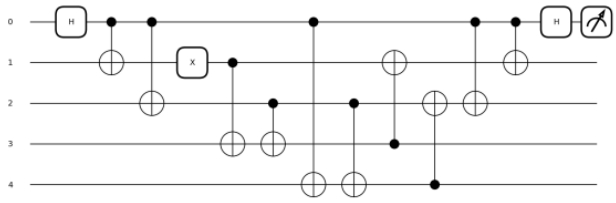

Fig. 5: A simple bit-flip quantum error correction circuit

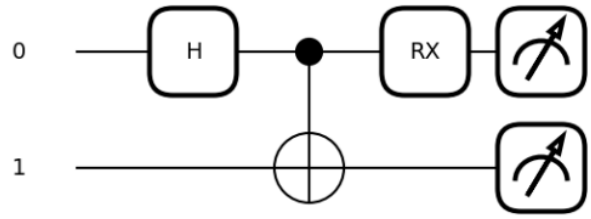

Fig. 6: NISQ circuit

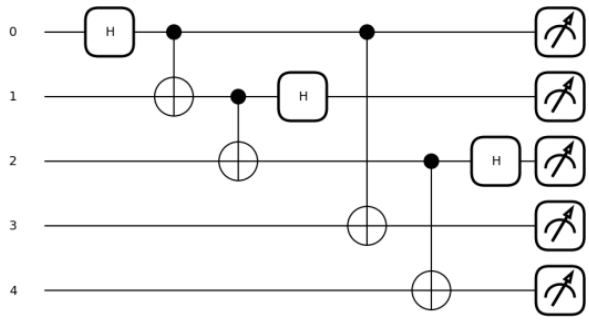

Fig. 7: FTQC circuit

The basic layout of a fault-tolerant quantum computing (FTQC) circuit using a surface code architecture, specifically arranged on a conceptual 3x3 qubit grid is shown in Figure 8. The goal of surface codes is to protect quantum information using spatially distributed entanglement and stabilizer checks, which detect and help correct errors without directly measuring the logical qubit. In this example, the circuit begins by applying a Hadamard gate to qubit 0 to place it in a superposition state, then distributes this information across the 3x3 grid of qubits (wires 0–8) via a series of CNOT gates to entangle the logical qubit with its neighbors mimicking the structure of a surface code. Additional CNOT gates between the lower rows simulate stabilizer interactions (like parity checks), and Hadamard gates on selected qubits simulate part of the Z-basis stabilizer operations. While this implementation is not a full surface code (as it lacks proper syndrome extraction and error correction logic), it serves as a foundational visualization of how surface code-style entanglement and layout could be constructed in a 9-qubit fault-tolerant system.

In the context of fault-tolerant quantum computing, a surface code circuit refers to the repeated, structured sequence of local quantum gates and measurements used to protect logical qubits against noise. Physical qubits are arranged on a two-dimensional lattice where data qubits store quantum information and ancilla (syndrome) qubits are periodically initialized, entangled with neighboring data qubits via CNOT gates, and then measured to extract stabilizer syndromes without collapsing the logical state.

These circuits continuously detect bit-flip and phase-flip errors, and by repeating the measurements over time, real errors can be distinguished from measurement noise, forming space–time error patterns that are decoded classically. Fault tolerance arises because logical information is encoded non-locally across many qubits, so that a minimum number of correlated physical errors is required to cause a logical failure, making surface code circuits a practical and scalable foundation for reliable quantum computation.

This approach of FTQC is a simplified version of a surface code. A full implementation of surface codes requires additional functionality for syndrome extraction, error correction logic, and many layers of operations.

The rest of this subsection implements a Bayesian optimization process to optimize the hyper-parameters of a quantum error correction scheme, simulating fault-tolerant performance. The mock quantum code performance function serves as a mock objective function that simulates the error rate of a quantum code based on three hyper-parameters: Code distance, logical error rate, and syndrome extraction frequency. The optimization is done using the skopt library, which searches the hyper-parameter space defined by Integer (for code distance and syndrome frequency) and Real (for the logical error rate). This returns the optimized parameters and the minimum error rate.

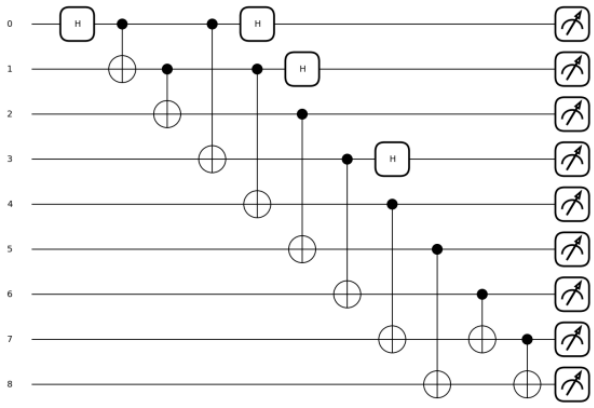

Fig. 8: FTQC Surface Code Circuit

The optimization process is visualized in two ways: First, by building and drawing a directed graph that traces the evolution of the parameters over the optimization iterations, and second, by plotting the convergence of the optimization using the plot convergence function. The goal is to minimize the simulated error rate by adjusting these hyper-parameters, representing an exploration of the design space for a fault-tolerant quantum error-correction scheme.

The mock objective function simulates the error rate for a given set of hyper-parameters. The performance is given by:

$$\text{performance} = \frac{1}{d} + LER + \frac{1}{f} + N(0,0.01) \quad (1)$$

Where:

- d represents the code distance
- LER is the logical error rate
- f is the syndrome extraction frequency
- $N(0,0.01)$ represents Gaussian noise with mean 0 and standard deviation 0.01

The goal of Bayesian optimization is to minimize the error rate by adjusting the hyper-parameters. The optimization process is:

$$\theta_{opt} = \arg\min_{\theta} f(\theta) \quad (2)$$

Where

- $f(\theta)$ is the objective function (mock quantum code performance),
- θ = (distance, logical error rate, syndrome freq) are the hyper-parameters to be optimized
- θopt represents the optimal hyper-parameters that minimize the error rate

The convergence of the optimization process can be expressed as:

$$Convergence = \min_{i} f(\theta_i) \quad (3)$$

Where:

- $f(\theta i)$ is the value of the objective function at the $i^{\text{-th}}$ iteration
- The convergence plot shows how the error rate decreases over iterations

Implementation and Results

Active Reset

The optimal quadrature histograms of active reset for a single Aspen-4 qubit are shown in Figure 9 (Karalekas et al., 2020). The qubit is initially prepared in an equal superposition state at capture 0, followed by three successive rounds. On Aspen-4, the relaxation times (T1) of qubits typically range from 10 to 20 µs. In the passive reset approach, achieving a reset fidelity F > 99% requires waiting approximately 5T1 = 100µs between experimental shots, since 1−e −5 ≈ 0.993. This approach relies on the natural relaxation of the qubit to the ground state. By contrast, the active reset method leverages measurement and feedback to reset the qubit much faster. Specifically, three rounds of measurement and feedback can be completed in about 10µs. After the third round of active reset, a high reset fidelity of F = 99.6% is achieved, significantly reducing the time required between quantum operations.

Scheduling Execution and Waiting

Figure 10 illustrates the performance of the First-Come-First-Serve (FCFS) scheduling algorithm for quantum job processing. The top chart displays the execution times for 15 quantum jobs, showing variability in how long each job takes to run, with durations ranging from around 1.5 to 4 seconds. These times are independent of job position, highlighting that FCFS does not prioritize shorter tasks.

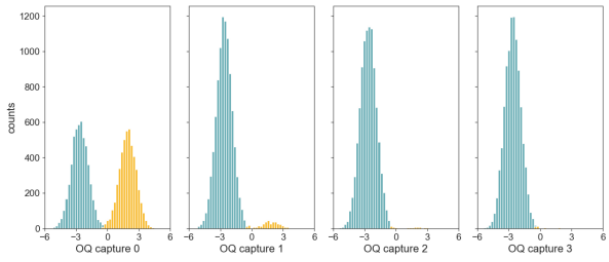

Fig. 9: Optimal Quadrature Histograms for Active Reset (Karalekas et al., 2020)

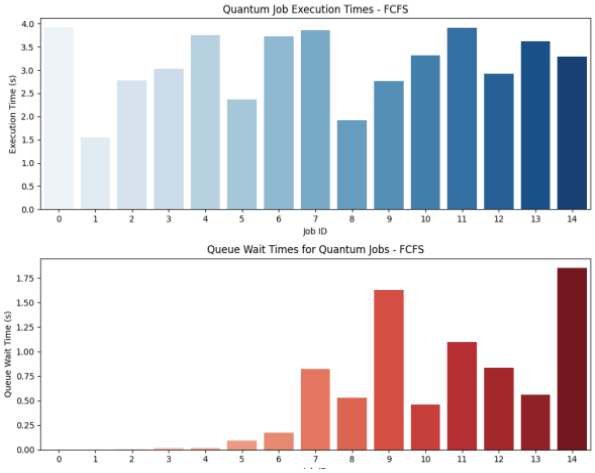

Fig. 10: Execution and Waiting Times for FCFS

The bottom chart shows the corresponding queue wait times, which generally increase with job ID. This is a direct consequence of the FCFS strategy, where each job must wait for all preceding jobs to complete before starting, regardless of its own execution time. As a result, later jobs experience longer delays even if they are shorter, leading to inefficiencies. This visualization highlights a key drawback of FCFS in systems where job durations vary significantly: It can cause increased average waiting times and poor overall responsiveness.

Figure 11 presents the performance of the Shortest Job Next (SJN) scheduling algorithm applied to the same set of quantum jobs. In the top chart, execution times are shown for each job, revealing that SJN favors shorter jobs first, as lower-duration jobs appear earlier in the sequence. This strategy aims to reduce the overall average waiting time by prioritizing jobs that can be completed quickly. The bottom chart confirms this by displaying significantly lower wait times compared to the FCFS approach. Many jobs have near-zero wait times, especially the shorter ones scheduled earlier, while only a few longer jobs experience moderate waiting. Overall, the SJN algorithm proves more efficient in minimizing queue delays and enhancing throughput by reducing the average wait time, particularly in systems where job durations vary widely.

FTQC

Figure 12 illustrates the performance of the optimization algorithm over 20 function evaluations. Initially, during the first few evaluations (from call 1 to 4), the minimum value of the objective function remains nearly constant at around 0.238. This plateau indicates that the algorithm was either exploring less promising regions of the search space or had not yet identified a significantly better solution. However, a notable improvement occurs at the 5th evaluation, where the minimum value sharply drops to approximately 0.188. This sudden decrease suggests that the algorithm discovered a much better solution, likely transitioning from a suboptimal region to one closer to a local or global optimum. After this breakthrough, the curve begins to flatten again, showing only marginal improvements from the 6th call onward. This slow convergence phase indicates that the optimizer is refining the solution but is likely encountering diminishing returns, either because it has nearly converged to an optimum or due to limitations in its ability to escape local minima. Overall, the plot reflects a typical optimization pattern: An initial exploration phase, followed by a significant improvement, and finally a slow convergence as the algorithm fine-tunes the best solution found.

The hyper-parameter ranges shown in Table 2 reflect practical design trade-offs in fault-tolerant quantum error correction, balancing physical resource overhead against logical reliability.

The implementation simulates a quantum error correction scheme based on a bit-flip code. The key component of the simulation is the encoding of a logical qubit into a 3-qubit code, where two CNOT gates entangle the qubits to protect the encoded logical qubit against bit-flip errors. Noise is introduced to the system by applying a bit-flip error to each qubit with a probability parameter p. To detect and correct errors, two stabilizers (corresponding to Z0Z1 and Z1Z2) are measured, and the results (syndromes) indicate whether and where errors have occurred. Based on these syndromes, an error correction function is used to apply corrective operations (Pauli-X gates) to the affected qubit.

The implementation computes the fidelity of the corrected state with the ideal encoded state (either $|000\rangle$ or $|111\rangle$), allowing for an assessment of the effectiveness of the error correction scheme under various noise conditions.

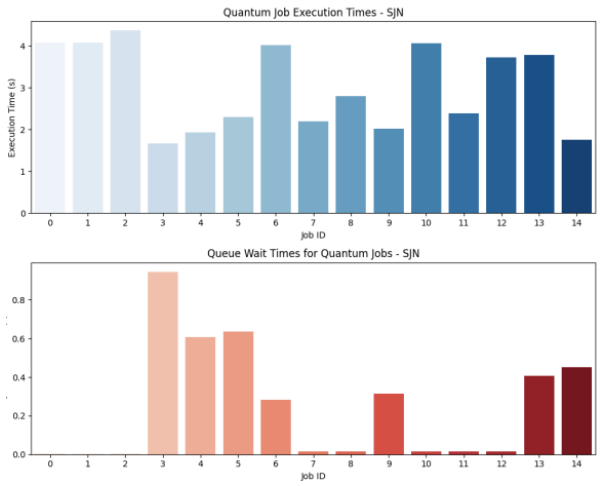

Fig. 11: Execution and Waiting Times for SJN

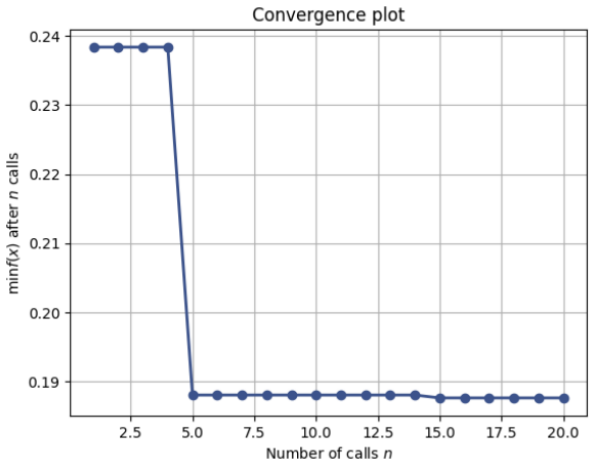

Fig. 12: Convergence Plot

Table 2: Hyper-parameter Search Space for Quantum Error Correction Optimization

Hyper-parameter	Symbol	Search Range	Description
Code Distance	d	$[3, 11]$	Number of physical qubits encoding one logical qubit; higher values improve error resilience at the cost of resources
Logical Error Rate	LER	$[10^{-4}, 10^{-2}]$	Probability of logical error after error correction, capturing residual noise effects
Syndrome Extraction Frequency	f	$[1, 10]$	Number of syndrome measurements per circuit execution, trading off error detection accuracy and overhead

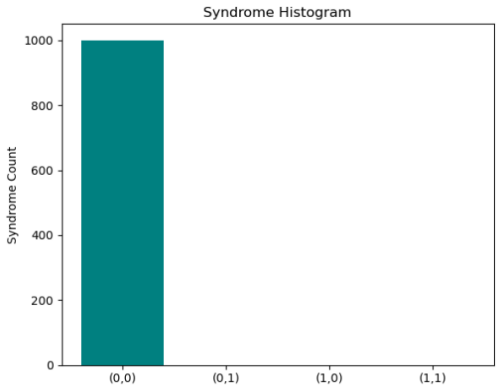

Fig. 13: Syndrome Histogram

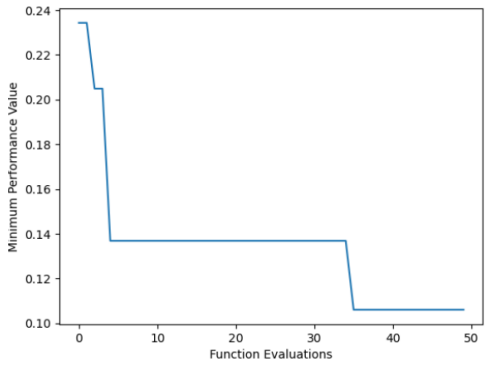

Fig. 14: Convergence of Quantum Code Performance

A dynamic animation is generated to visualize the performance of the error correction process. The animation includes a histogram showing the distribution of measured syndromes as shown in Figure 13.

Based on Eq. 1, the convergence of quantum code performance is shown in Figure 14. The provided convergence plot illustrates the optimization profile of a quantum code over 50 function evaluations, characterized by a distinct step-down pattern typical of heuristic or gradient-free optimizers. The process begins with an initial plateau at a performance value of approximately 0.235, followed by two rapid descents within the first five evaluations that bring the minimum value down to 0.138. A significant period of stagnation occurs between evaluations 5 and 33, suggesting the optimizer was either stuck in a local minimum or exploring less optimal regions of the parameter space. A final breakthrough occurs around evaluation 34, where the performance drops to its terminal value of approximately 0.106, remaining stable thereafter. This trajectory suggests that while the algorithm successfully identified a more optimal configuration late in the run, the long middle plateau may indicate a need for more aggressive exploration or a more sensitive optimizer to accelerate convergence in future simulations.

Table 3 shows that the advantage of active reset extends beyond fidelity, impacting scheduling predictability, system throughput, and scalability in quantum cloud environments, making it a key enabler for fault-tolerant and mid-circuit quantum workflows.

Table 3: Comparison of Active and Passive Qubit Reset Beyond Fidelity

Criterion	Passive Reset (Thermal Relaxation)	Active Reset (Measurement + Feedback)
Reset Latency	High, bounded by 4–5T_1 wait time	Low, typically a few microseconds
Throughput Impact	Reduces experiment repetition rate	Enables high repetition and reuse of qubits
Scheduler Predictability	Low (reset time depends on physical decay)	High (bounded and deterministic reset time)
Resource Utilization	Idle qubits during relaxation periods	Efficient qubit reuse with minimal idle time
Support for Mid-Circuit Operations	Not supported	Fully supported with feed-forward control
Compatibility with QEC / FTQC	Limited, unsuitable for frequent resets	Essential for syndrome extraction and recovery
Scalability in Cloud Environments	Poor under multi-tenant workloads	Scales well with dynamic and shared workloads
Sensitivity to Hardware Variability	High dependence on qubit T_1 heterogeneity	Mitigates variability via enforced reset protocol
Classical Control Overhead	Minimal	Requires fast readout and feedback logic
Suitability for Quantum-Aware Scheduling	Low	High

Table 4: Overhead Cost in Fault-Tolerant Quantum Computing (FTQC)

Resource Type	Scaling Behavior	Practical Impact
Physical qubits	$O(d^2)$ per logical qubit	10^3–10^4× increase in qubit count
Execution time	$O(d)$ per logical gate	Increased circuit depth and latency
Magic state distillation	Large constant overhead	Dominant cost for non-Clifford (T) gates
Classical decoding	Continuous real-time processing	Requires fast classical co-processors

Fault-Tolerant Quantum Computing (FTQC) incurs substantial overhead due to the need for quantum error correction. Table 4 summarizes the overhead. In leading schemes such as the Surface code, each logical qubit is encoded into O (d 2) physical qubits, where d is the code distance, often resulting in a 103–104× increase in qubit count. In addition, logical gate execution time scales as O(d) because repeated syndrome extraction cycles are required to suppress errors. A major contributor to overall cost is magic state distillation, which introduces a large constant overhead and can dominate resource usage for non-Clifford gates such as T gates. Furthermore, FTQC requires continuous real-time classical decoding and feed-forward control, adding significant engineering complexity. Together, these factors make FTQC resource-intensive in both hardware and time, but necessary to achieve reliable large-scale quantum computation.

Conclusion and Future Work

This paper focuses on the three pillars of active reset, scheduling, and fault tolerance. The active reset method leverages measurement and feedback to reset the qubit much faster and with high fidelity. The proposed scheduling algorithm is designed to efficiently manage quantum job execution in a fault-tolerant quantum-classical cloud computing environment.

The paper also implements a Bayesian optimization process to optimize the hyper-parameters of a quantum error correction scheme.

The quantum job scheduling flowchart provides a structured approach to job allocation and execution. Implementing optimizations such as load balancing and adaptive scheduling can significantly improve performance and reliability in quantum computing environments.

In evaluating quantum-classical cloud computing, it becomes clear that its strengths and weaknesses are tightly intertwined. On the positive side, it enables access to powerful quantum capabilities through cloud platforms, making advanced computational tools available to a broader audience without the need for specialized on-premise hardware. It also allows hybrid workflows that exploit the comparative advantages of each system. However, this comes with the trade-off of high latency and limited scalability due to the current state of quantum hardware. The cost model is manageable for experimental or academic work, but could become burdensome for large-scale enterprise deployment. Furthermore, the fragmented nature of quantum software ecosystems adds to the development complexity. As such, this paradigm should be seen not as a final destination but as a critical evolutionary step one that bridges current capabilities with the quantum future. While it's not yet transformative at scale, it is laying the groundwork for revolutionary advances in the coming decades.

Acknowledgment

The authors would like to express their sincere appreciation to all individuals and institutions who supported this work. Special thanks are extended to colleagues and reviewers for their valuable feedback and constructive comments.

Funding Information

This research received no external funding.

Ethics

This study did not involve human participants or animals. The authors declare that there are no ethical issues related to this research.

Conflict of Interest

The authors declare no conflict of interest.

References

Alfred, V. A., Monica, S. L., & Jeffrey, D. U. (2007). *Compilers principles, techniques & tools*.

Aliferis, P., & Preskill, J. (2008). Fault-tolerant quantum computation against biased noise. *Physical Review A*, *78*(5), 052331. https://doi.org/10.1103/physreva.78.052331

Barletta, V. S., Caivano, D., Lako, A., & Pal, A. (2023). Quantum as a Service Architecture for Security in a Smart City. *Quality of Information and Communications Technology*, *1871*, 76–89. https://doi.org/10.1007/978-3-031-43703-8_6

Blekos, K., Brand, D., Ceschini, A., Chou, C.-H., Li, R.-H., Pandya, K., & Summer, A. (2024). A review on Quantum Approximate Optimization Algorithm and its variants. *Physics Reports*, *1068*, 1–66. https://doi.org/10.1016/j.physrep.2024.03.002

Cheng, B., Deng, X.-H., Gu, X., He, Y., Hu, G., Huang, P., Li, J., Lin, B.-C., Lu, D., Lu, Y., Qiu, C., Wang, H., Xin, T., Yu, S., Yung, M.-H., Zeng, J., Zhang, S., Zhong, Y., Peng, X., Yu, D. (2023). Noisy intermediate-scale quantum computers. *Frontiers of Physics*, *18*(2), 21308. https://doi.org/10.1007/s11467-022-1249-z

Deglmann, P., Schäfer, A., & Lennartz, C. (2015). Application of quantum calculations in the chemical industry—An overview. *International Journal of Quantum Chemistry*, *115*(3), 107–136. https://doi.org/10.1002/qua.24811

Fan, L., & Han, Z. (2022). Hybrid Quantum-Classical Computing for Future Network Optimization. *IEEE Network*, *36*(5), 72–76. https://doi.org/10.1109/mnet.001.2200150

Fowler, A. G., Mariantoni, M., Martinis, J. M., & Cleland, A. N. (2012). Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, *86*(3), 032324. https://doi.org/10.1103/physreva.86.032324

Fujiwara, Y., Gruner, A., & Vandendriessche, P. (2015). High-Rate Quantum Low-Density Parity-Check Codes Assisted by Reliable Qubits. *IEEE Transactions on Information Theory*, *61*(4), 1860–1878. https://doi.org/10.1109/tit.2015.2398436

Golec, M., Hatay, E. S., Golec, M., Uyar, M., Golec, M., & Gill, S. S. (2024). Quantum cloud computing: Trends and challenges. *Journal of Economy and Technology*, *2*, 190–199. https://doi.org/10.1016/j.ject.2024.05.001

Heim, B., Soeken, M., Marshall, S., Granade, C., Roetteler, M., Geller, A., Troyer, M., & Svore, K. (2020). Quantum programming languages. *Nature Reviews Physics*, *2*(12), 709–722. https://doi.org/10.1038/s42254-020-00245-7

Hibat-Allah, M., Mauri, M., Carrasquilla, J., & Perdomo-Ortiz, A. (2024). A framework for demonstrating practical quantum advantage: comparing quantum against classical generative models. *Communications Physics*, *7*(1), 68. https://doi.org/10.1038/s42005-024-01552-6

Karalekas, P. J., Tezak, N. A., Peterson, E. C., Ryan, C. A., da Silva, M. P., & Smith, R. S. (2020). A quantum-classical cloud platform optimized for variational hybrid algorithms. *Quantum Science and Technology*, *5*(2), 024003. https://doi.org/10.1088/2058-9565/ab7559

Nguyen, H. T., Usman, M., & Buyya, R. (2024). iQuantum: A toolkit for modeling and simulation of quantum computing environments. *Software: Practice and Experience*, *54*(6), 1141–1171. https://doi.org/10.1002/spe.3331

Preskill. (2023). Crossing the quantum chasm: From NISQ to fault tolerance. *Quantum Computing, Communication, and Simulation III*. https://quantumfrontiers.com/2023/12/09/crossing-the-quantum-chasm-from-nisq-to-fault-tolerance/ Reference 16:

Rajagopalan, S., Sundari, T. M., & Gayathri, M. (2024). Applications of quantum computing in financial planning and financial control. *Industrial Quantum Computing: Algorithms, Blockchains, Industry 4.0*, 127–140. https://doi.org/10.1515/9783111354842-008

Raju, N., Quazi, F., & Kondle, P. (2024). Quantum Computing in Healthcare - Unlocking New Frontiers in Drug Discovery and Data Management. *International Journal of Global Innovations and Solutions (IJGIS)*, e90189c8-e5dab7bb. https://doi.org/10.21428/e90189c8.e5dab7bb

Rosenfeld, V., Breß, S., & Markl, V. (2023). Query Processing on Heterogeneous CPU/GPU Systems. *ACM Computing Surveys*, *55*(1), 1–38. https://doi.org/10.1145/3485126

Scholl, P., Shaw, A. L., Tsai, R. B.-S., Finkelstein, R., Choi, J., & Endres, M. (2023). Erasure conversion in a high-fidelity Rydberg quantum simulator. *Nature*, *622*(7982), 273–278. https://doi.org/10.1038/s41586-023-06516-4

Singh, J., & Bhangu, K. S. (2023). Contemporary Quantum Computing Use Cases: Taxonomy, Review and Challenges. *Archives of Computational Methods in Engineering*, *30*(1), 615–638. https://doi.org/10.1007/s11831-022-09809-5

Takeori, M., Shimada, N., Alevras, D., Parney, B., Sharma, D., Chu, Q., & Cena, B. (2024). Workforce Task Execution Scheduling Using Gate-Based Quantum Computers. *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, 315–321. https://doi.org/10.1109/qce60285.2024.00045

Tilly, J., Chen, H., Cao, S., Picozzi, D., Setia, K., Li, Y., Grant, E., Wossnig, L., Rungger, I., Booth, G. H., & Tennyson, J. (2022). The Variational Quantum Eigensolver: A review of methods and best practices. *Physics Reports*, *986*, 1–128. https://doi.org/10.1016/j.physrep.2022.08.003

Wille, R., Van Meter, R., & Naveh, Y. (2019). IBM's Qiskit Tool Chain: Working with and Developing for Real Quantum Computers. *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 1234–1240. https://doi.org/10.23919/date.2019.8715261