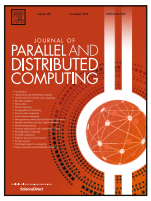
Journal of Parallel and Distributed Computing 216 (2026) 105303

Contents lists available at ScienceDirect

Journal of Parallel and Distributed Computing

journal homepage: www.elsevier.com/locate/jpdc



Comprehensive overview of quantum serverless: Elastic integration of quantum and classical computing

Dimitar Mileski [a], Marjan Gusev [a], Sashko Ristov [b,*]

[a] Sts Cyril and Methodius University in Skopje, Faculty of Computer Science and Engineering, Skopje, North Macedonia
[b] University of Innsbruck, Austria

ARTICLE INFO

Keywords:
Quantum computing
Serverless
Integration
Scalability
Elasticity

ABSTRACT

Quantum computing is increasingly accessed through cloud-based hybrid environments, but practical adoption depends on more than exposing remote quantum backends. It also requires managed abstractions for orchestrating classical and quantum stages, handling queues, binding backends, and integrating surrounding workflow logic. This paper presents a comprehensive survey of Quantum Serverless from that perspective. Our contribution is twofold: (1) we consolidate the key architectural components of Quantum Serverless into a unified reference architecture; and (2) we evaluate current industry and research offerings through a KPI-driven framework covering ecosystem coverage, service operability, queue handling, pricing flexibility, near-real-time readiness, and production maturity. Using this framework, we analyze 18 quantum cloud platforms and show that the ecosystem is growing but remains unevenly mature. Queue-based execution is already a baseline capability, whereas stronger serverless characteristics, such as Function-as-a-Service, Containers-as-a-Service, near-real-time responsiveness, and production readiness, remain limited. Overall, current Quantum Serverless is best understood as a queue-driven hybrid orchestration model rather than a fully mature cloud-native serverless layer. The survey identifies the main maturity gaps and outlines directions for future development.

1. Introduction

Quantum computing is progressing from laboratory-scale experimentation toward cloud-accessible hybrid computing environments in which quantum and classical resources are combined within a single workflow. Yet, access to quantum hardware alone is not sufficient for broad adoption. Users must still coordinate classical pre-processing and post-processing, select backends, submit jobs, monitor queues, retrieve results, and manage the surrounding execution environment. As in the early evolution of cloud computing, practical adoption depends not only on access to infrastructure, but also on the emergence of higher-level managed abstractions that make complex resources easier to use.

Serverless computing is one such abstraction model [1]. In classical cloud systems, serverless platforms allow developers to focus on application logic while the provider manages provisioning, scaling, scheduling, monitoring, and much of the operational complexity. Two common forms are Function-as-a-Service (FaaS), where short-lived event-triggered functions are executed on demand, and Containers-as-a-Service (CaaS), where containerized applications are deployed while the platform automates runtime and scaling behavior. These models helped shift cloud value from raw infrastructure exposure toward managed execution services and developer-oriented ecosystems.

The term *Quantum Serverless* was introduced by IBM [2] in 2021 to describe an execution model in which quantum programs are integrated with classical cloud services while infrastructure-level concerns are abstracted from end users. In this setting, the developer still defines the quantum circuit and the surrounding classical logic, but the platform increasingly manages orchestration concerns such as job submission, queue handling, scheduling, retries, backend binding, and scalable execution of classical tasks around quantum calls. Thus, the key question is no longer only which quantum hardware is available, but also which platforms provide the software abstractions and operational mechanisms needed for usable hybrid quantum-classical computing.

This shift motivates the main message of this paper: the maturity of Quantum Serverless depends less on the mere availability of quantum backends and more on the quality of orchestration, software ecosystems, service abstractions, and production-readiness of hybrid execution environments. Current platforms expose an expanding set of quantum resources, but they differ substantially in how far they move from hardware access toward managed, cloud-native quantum services.

* Corresponding author.
E-mail addresses: dimitar.mileski@finki.ukim.mk (D. Mileski), marjan.gushev@finki.ukim.mk (M. Gusev), sashko.ristov@uibk.ac.at (S. Ristov).

https://doi.org/10.1016/j.jpdc.2026.105303
Received 7 September 2025; Received in revised form 13 April 2026; Accepted 20 June 2026
Available online 22 June 2026

This paper addresses that gap through a cross-provider survey and evaluation of Quantum Serverless. Unlike prior contributions that are mainly software and platform-specific [3], general advances in the quantum computing stack [4,5], or IBM/Qiskit-centered [6], we provide a broader comparative analysis based on a unified architectural view and a KPI-driven evaluation framework. Our contribution is twofold:

1. we consolidate the key architectural components of Quantum Serverless into a unified reference view; and
2. we evaluate current industry and research offerings through declarative and quantifiable key performance indicators that capture ecosystem coverage, service operability, queue handling, pricing flexibility, Near Real-time readiness, and production maturity.

This study is guided by the following research questions:

- RQ_1: What are the key architectural components of Quantum Serverless computing?
- RQ_2: Given KPIs derived from these architectural components, how are quantum and cloud stakeholders progressing toward practical Quantum Serverless computing?

By answering these questions, the paper aims to do more than catalogue platforms. It interprets the current landscape of Quantum Serverless as an emerging transition from raw quantum access toward managed hybrid service models, identifies the main maturity gaps, and highlights where present solutions remain closer to queue-driven access than to fully elastic, production-grade serverless execution.

The rest of the paper is organized as follows. Section II presents the unified Quantum Serverless architecture. Section III reviews related work and the broader NISQ context. Section IV introduces the evaluation methodology and KPI framework. Section V evaluates current platforms and analyzes the results. Section VI discusses limitations, maturity gaps, and implications for the Quantum Serverless ecosystem. Section VII concludes the paper and outlines future directions.

2. Quantum serverless architecture

2.1. From classical serverless to quantum serverless

Serverless computing is a cloud execution model in which developers focus primarily on application logic, while the platform manages most infrastructure concerns, including provisioning, scaling, scheduling, monitoring, and runtime maintenance. In contrast to traditional cloud deployment, where users explicitly configure and manage servers or long-lived virtual machines, serverless platforms shift much of this operational responsibility to the provider.

Two widely used forms of serverless computing are Function-as-a-Service (FaaS) and Containers-as-a-Service (CaaS). In FaaS, users deploy short-lived functions that are triggered by events, such as API calls, messages, or workflow steps, and the platform automatically starts, scales, and terminates function instances as needed. In CaaS, users deploy containerized applications while the platform automates scheduling, autoscaling, and lifecycle management of the container runtime. Although these two models differ in granularity, both reduce the amount of infrastructure management exposed to the developer.

Prior serverless research has shown that higher-level workflow abstractions can be built on top of FaaS platforms, enabling portable function choreographies and reducing provider lock-in by abstracting from concrete function implementations and backend platforms [7]. More recent work indicates that abstraction can extend beyond function execution itself to supporting backend services, where workflow systems can control not only where functions run, but also how such services are selected, either at workflow level or through dynamic function-level binding [8]. Extensions to federated FaaS further show that workflow functions can be dynamically distributed across multiple providers and regions to improve scalability and performance [9].

The key contribution of classical serverless is therefore not the removal of computation infrastructure, but the abstraction of its operational complexity. Developers still write code and define execution logic, yet the platform increasingly takes responsibility for deployment, elasticity, fault handling, and resource coordination. This shift from raw infrastructure access toward managed execution services played an important role in the broader adoption of cloud computing.

Quantum Serverless extends the same abstraction principle to hybrid quantum-classical workloads. Instead of exposing only a remote quantum backend, a Quantum Serverless platform aims to manage the surrounding execution context in which quantum circuits are embedded. In this setting, the user still defines the quantum circuit and the associated classical logic, but the platform increasingly handles orchestration concerns such as job submission, queue management, backend binding, state tracking, retry handling, and the elastic execution of classical pre-processing and post-processing tasks around quantum calls. Thus, the move from classical serverless to Quantum Serverless is not a change in the idea of abstraction itself, but an extension of that idea to a more complex hybrid computing environment.

2.2. Motivating workflow: non-serverless vs quantum serverless

A practical workflow comparison helps clarify what is meant by Quantum Serverless. Consider a simple Bell-pair experiment executed on a remote quantum backend. In a conventional, non-serverless workflow, the user writes the circuit, transpiles it for a selected backend, manually chooses execution parameters, submits the job, monitors queue status, handles failures or timeouts, retrieves results, and then separately executes any classical pre-processing or post-processing required for the experiment. Although the quantum hardware is accessed remotely, the surrounding workflow logic and operational coordination remain largely the user's responsibility.

In a Quantum Serverless workflow, the user still defines the quantum circuit and the associated classical logic, but the hybrid workload is packaged as a function, task, or workflow unit that is managed by the platform. The platform then assumes responsibility for much of the surrounding operational complexity: it orchestrates the classical and quantum stages, manages job submission and backend binding, tracks execution state, reacts to queue conditions, handles retries and failures, and elastically scales parallel classical tasks such as parameter generation, result aggregation, or repeated circuit invocations. In this way, the abstraction does not remove the need to design quantum programs, but it reduces the infrastructure and coordination burden required to execute hybrid quantum-classical workflows.

This distinction is important because remote access to a quantum backend alone does not yet constitute Quantum Serverless. A platform becomes "serverless" in the stronger sense when it manages the execution context around the quantum call, rather than merely exposing hardware through an API. Thus, the main difference is not whether a quantum computer is accessed through the cloud, but whether the surrounding workflow is elevated from a manually coordinated process to a managed-service-level execution model. This perspective motivates the unified architecture presented next.

As a summary, non-serverless quantum cloud access exposes a backend, whereas Quantum Serverless aims to manage the hybrid workflow around that backend.

2.3. Unified quantum serverless architecture

Fig. 1 presents a unified architecture for Quantum Serverless, organized around the interaction between a Classical Data Center, a Quantum Data Center, and, when applicable, an External Quantum Hardware Provider. The purpose of this architecture is not only to expose quantum hardware through the cloud, but to provide a managed execution environment in which hybrid quantum-classical workloads can be submitted, coordinated, monitored, and scaled.

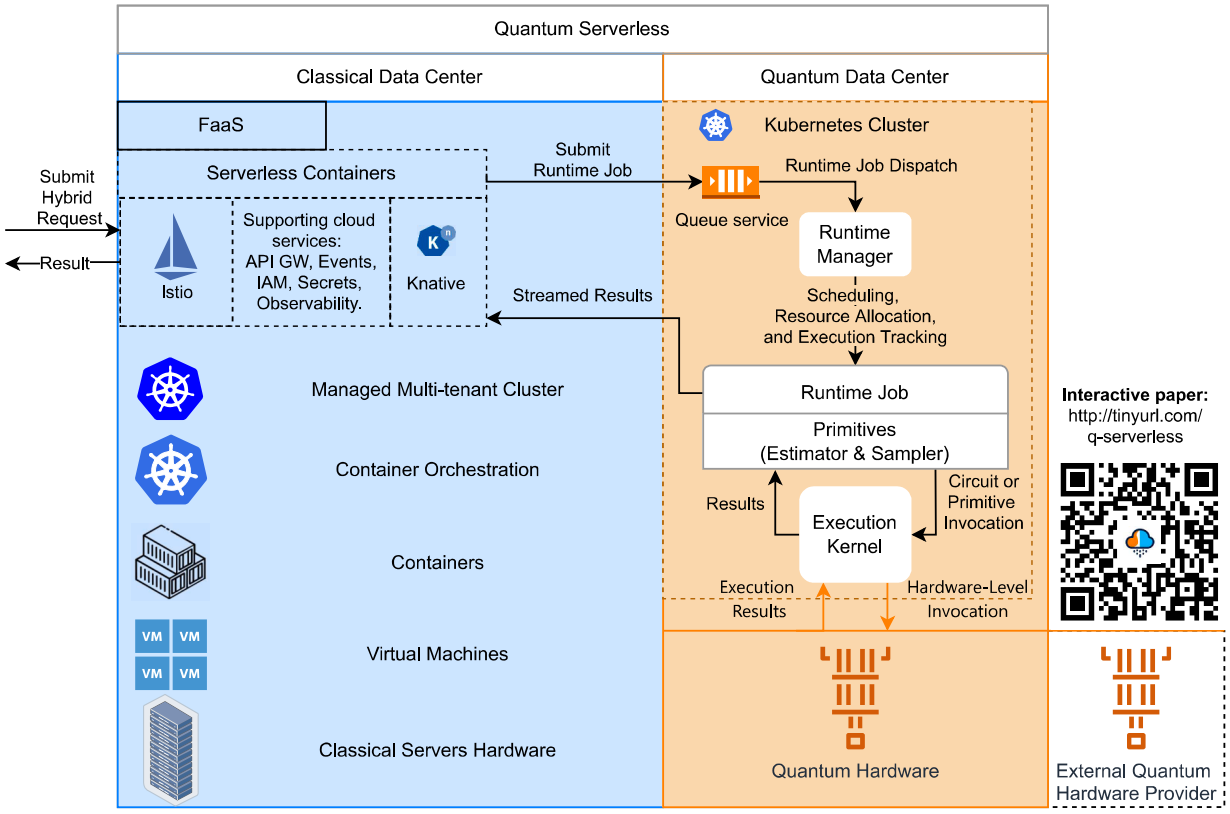


Fig. 1. Unified quantum serverless architecture.

A request initiated by a user is first received by the Classical Data Center, which provides the entry point for application logic, APIs, user interaction, and serverless execution. The user-defined workload may contain both classical and quantum parts. The classical part can be executed as functions or serverless containers, while the quantum-related part is packaged as a runtime job and forwarded to the Quantum Data Center. There, the queue service buffers requests according to hardware availability, and the Runtime Manager orchestrates scheduling, resource allocation, and execution tracking. The Runtime Job encapsulates the selected backend, input parameters, and the corresponding quantum circuit or primitive call. Primitives such as the Estimator and Sampler provide a higher-level interface between application logic and quantum execution. The Execution Kernel then translates the request into a hardware-level invocation and dispatches it either to locally integrated Quantum Hardware or to an External Quantum Hardware Provider. After execution, the results are returned to the Quantum Data Center, streamed back to the Classical Data Center for any required post-processing, and finally delivered to the user.

This dataflow clarifies the main abstraction provided by Quantum Serverless. The user still defines the hybrid workload, but the platform manages the operational path around the quantum call: submission, queue handling, backend binding, state tracking, and the coordination of surrounding classical tasks. In that sense, the architecture extends the logic of classical serverless platforms into a hybrid environment where quantum hardware remains scarce, asynchronous, and operationally constrained.

Classical federated serverless research has further shown that interoperability challenges also extend to deployment and storage layers, motivating abstractions that organize execution around functions, providers, and regions while hiding provider-specific deployment and storage details [10].

Fig. 1 therefore should be read as a layered orchestration view rather than only as a collection of components. The Classical Data Center contains the serverless entry points and cloud-side execution environment, including functions, serverless containers, orchestration support, and supporting infrastructure services. The Quantum Data Center contains the runtime path that specializes this execution model for quantum workloads, including the queue service, Runtime Manager, Runtime Job abstraction, primitives, Execution Kernel, and access to quantum hardware. The External Quantum Hardware Provider represents remote quantum backends that are reachable through the execution layer when the platform does not rely solely on in-house hardware. This separation makes explicit where classical elasticity ends, where quantum execution begins, and where the two are coordinated.

2.4. Key component descriptions

Fig. 1 illustrates the key components of the Quantum Data Center. It shows the role of the Runtime Manager in executing Runtime Jobs and monitoring progress; the function of primitives like the Estimator and Sampler in simplifying workloads; how Quantum Circuits define computations; and the role of the Execution Kernel in abstracting hardware operations, all supported by cloud-based Quantum Hardware.

2.4.1. Runtime manager

The Runtime Manager is the central orchestration component of the Quantum Serverless architecture. It receives runtime jobs from the queueing layer and coordinates their execution across the hybrid quantum-classical workflow. Its responsibilities include scheduling jobs, selecting or binding the appropriate backend, allocating execution resources, tracking job state, and managing the operational lifecycle of the workload until results are returned. In this sense, the Runtime Manager acts as the control-plane element that connects user-level workflow requests with the underlying quantum execution path.

In a Quantum Serverless setting, the role of the Runtime Manager extends beyond simply forwarding a circuit to a backend. It manages

the execution context around the quantum call, including interaction with queue services, coordination of classical pre-processing and post-processing steps, monitoring progress, and handling failures or retries when required. This is one of the key distinctions between exposing quantum hardware through an API and providing a managed Quantum Serverless execution model.

We use Qiskit Runtime [2,11] as a representative example. Qiskit Runtime provides a service and programming model for executing quantum workloads more efficiently at scale by combining quantum execution with managed runtime support and higher-level primitives such as the Estimator and Sampler. Although our examples rely on Qiskit, the same architectural role of the Runtime Manager can be generalized to other frameworks and platforms that aim to support managed hybrid quantum-classical execution.

Thus, the Runtime Manager operationalizes the serverless abstraction by turning a user-defined hybrid workload into a managed execution process.

2.4.2. Runtime job

A Runtime Job is the execution unit managed by the Quantum Serverless platform. It encapsulates the information required to run a hybrid quantum-classical workload, including the selected backend, input parameters, execution settings, and the associated quantum circuit or primitive invocation. Once submitted, the Runtime Job becomes the object through which the platform tracks execution state, monitors progress, and returns results to the higher-level application workflow.

Within the architecture, the Runtime Job serves as the bridge between user-defined workload logic and managed execution. It allows the Runtime Manager to schedule, queue, monitor, and control the lifecycle of a submitted task without exposing low-level execution details to the user. This packaging of execution context is essential for Quantum Serverless, because hybrid workloads often involve asynchronous behavior, hardware waiting times, and coordination between classical and quantum stages.

In Qiskit Runtime, the Runtime Job abstraction provides a concrete example of this role by representing an executable workload together with its metadata, execution status, and result retrieval interface. Although the terminology may vary across platforms, the same architectural idea applies more broadly: Quantum Serverless requires a managed job abstraction that persists the state and context of execution across the full hybrid workflow.

In this sense, the Runtime Job is the persistent execution context that enables serverless-style management of hybrid quantum workloads.

2.4.3. Primitives

Primitives provide the high-level interface through which hybrid quantum-classical workloads access quantum execution services. Instead of requiring the developer to manage low-level backend interaction for every task, primitives expose reusable execution patterns that capture common quantum operations in a simplified and more application-oriented form. In the Quantum Serverless architecture, they act as the interface layer between user-defined workflow logic and the managed runtime environment.

In practice, primitives reduce the coordination burden on the user by packaging recurring execution behavior into standardized calls that can be scheduled, monitored, and optimized by the platform. This is important in Quantum Serverless because the goal is not only to provide access to quantum hardware, but also to abstract part of the surrounding execution complexity, including orchestration of classical and quantum steps, result handling, and repeated invocation patterns within larger hybrid workflows. Thus, primitives help operationalize the serverless abstraction at the API level.

We use Qiskit Runtime as a representative example. In this setting, the *Sampler* [12] accepts a quantum circuit as input and returns a readout of quasi-probabilities, while the *Estimator* [12] accepts circuits together with observables and evaluates expectation values and related quantities needed by many hybrid algorithms. These primitives therefore serve as the main access points through which applications interact with the managed runtime service. Although the terminology differs across platforms, the broader architectural role remains the same: primitives provide a simplified, reusable interface for invoking quantum functionality within a managed hybrid execution model.

2.4.4. Circuit

A quantum circuit [13] is the program-level representation of the quantum part of a hybrid workload. It specifies the sequence of operations to be applied to quantum data, including quantum gates, measurements, resets, and, when supported, interactions with classical control logic. Within the Quantum Serverless architecture, the circuit is the core computational artifact that is packaged inside a Runtime Job and passed through the managed execution path toward the quantum backend.

From an architectural perspective, the circuit should be distinguished from the surrounding workflow logic. The user defines the circuit as the quantum computation to be executed, while the platform manages how that circuit is submitted, scheduled, monitored, and integrated with classical pre-processing and post-processing steps. This distinction is important because Quantum Serverless does not abstract away quantum program design itself; rather, it abstracts much of the operational complexity around executing circuits within a larger hybrid application.

In this sense, the circuit is the quantum payload of the workflow, while the Runtime Manager, Runtime Job, Primitives, and Execution Kernel together provide the managed environment in which that payload is executed. Although different platforms may support different circuit models, optimizations, or intermediate representations, the architectural role remains the same: the circuit expresses the intended quantum computation, while the Quantum Serverless platform manages its execution context.

2.4.5. Execution kernel

The Execution Kernel is the component that connects the managed runtime environment to concrete quantum backends. It receives the circuit or primitive invocation from the Runtime Job context and performs the backend-facing execution step required to run that workload on the selected quantum hardware or simulator. In architectural terms, it is the boundary layer between the logical hybrid workflow and the physical execution environment.

Its role is not only to forward circuits for execution, but also to mediate backend-specific operational details that should remain abstracted from the end user. These details may include backend invocation format, execution submission to local or external quantum hardware, result retrieval, and the coordination needed to return execution outcomes back into the managed workflow. In this sense, the Execution Kernel operationalizes the serverless abstraction at the execution boundary: the user defines the computation, while the platform manages how that computation is actually dispatched and completed.

Within the unified Quantum Serverless architecture, the Execution Kernel therefore plays a key role in separating what is to be executed from how it is executed on the target backend. This separation is especially important in hybrid quantum-classical settings, where different providers may expose different hardware technologies, interfaces, and execution constraints. Although concrete implementations vary across platforms, the architectural function remains the same: the Execution Kernel provides a controlled execution handoff from runtime-managed workflow logic to backend-specific quantum processing.

2.4.6. Quantum hardware

Quantum Hardware is the execution target of the quantum part of the hybrid workflow. Within the Quantum Serverless architecture, it represents the backend on which circuits or primitive invocations are ultimately executed after passing through the managed runtime path.

This hardware may be integrated directly within the provider's environment or accessed through an External Quantum Hardware Provider, depending on the platform's deployment model.

From an architectural perspective, the key issue is not only the physical realization of the hardware, but the fact that quantum backends remain scarce, heterogeneous, and operationally constrained. Different providers expose different technologies, interfaces, and performance characteristics, while access is often subject to queueing, availability limits, and provider-specific execution policies. For this reason, quantum hardware should not be viewed as a directly accessible resource in the same way as ordinary cloud compute instances. Instead, it is reached through the managed execution path formed by the Runtime Manager, Runtime Job, Primitives, and Execution Kernel.

This distinction is essential for Quantum Serverless. The user interacts with a managed service abstraction, while the platform mediates access to one or more backend technologies and hides much of the operational complexity involved in selecting, invoking, and coordinating hardware execution. In this sense, the architecture separates hybrid application logic from the diversity and scarcity of actual quantum devices, allowing the same serverless execution model to be applied across both in-house and external quantum backends.

Together, these components define a managed execution path in which quantum circuits are embedded into a broader hybrid serverless workflow rather than submitted as isolated backend calls.

3. Related work and broader quantum context

This section positions the present survey against prior work and summarizes the broader technological context in which Quantum Serverless is emerging. We first review representative works that explicitly address Quantum Serverless or closely related hybrid quantum–classical serverless models. We then briefly discuss the current NISQ and hybrid HPC–QC landscape, which helps explain why orchestration, queueing, and managed execution are central concerns in today's Quantum Serverless ecosystem. The purpose of this section is therefore twofold: to clarify how this survey differs from earlier contributions and to situate Quantum Serverless within the broader maturity trajectory of near-term quantum computing.

3.1. Related work on quantum serverless

Research on Quantum Serverless is still emerging and spans several complementary directions, including platform-centered implementations, framework proposals, architectural integration approaches, scheduling models, and broader conceptual visions. IBM introduced the term *Quantum Serverless* in 2021 to describe a programming model that combines quantum and classical resources while abstracting infrastructure-level concerns from end users [2]. Building on this direction, Qiskit Serverless [14] was later introduced as an open-source framework for running workloads across quantum and classical resources, with support for reusable programs, long-running workloads, and parallelized pre-processing and post-processing around quantum execution.

A second line of work focuses on framework-level proposals for serverless quantum applications. Nguyen et al. [15] propose QFaaS, a serverless Function-as-a-Service framework for quantum computing that addresses challenges such as heterogeneous quantum programming languages and hardware platforms, integration of quantum and classical computation, and the lifecycle of a quantum function from development and deployment to backend selection, execution, and post-processing. Their work is important because it explicitly treats quantum functions as deployable serverless units, but it remains centered on a proposed framework and its lifecycle rather than on a broader cross-provider comparative assessment.

A third line of work addresses integration architectures between classical cloud systems and quantum backends. Grossi et al. [16] present a serverless cloud integration framework intended to improve accessibility and workload management when connecting API-exposed quantum providers to existing classical infrastructures. Their architecture uses FaaS for scalable job submission and a queue-based design to avoid bottlenecks, showing that serverless principles can help operationalize hybrid access to quantum resources. This contribution is relevant at the integration level, but it focuses on a specific architectural solution rather than on the broader maturity of Quantum Serverless across platforms.

Other works investigate optimization and performance aspects of hybrid quantum–classical serverless platforms. Cicconetti [17] studies the execution of variational quantum algorithms in hybrid infrastructures through a system model and simulator for hybrid classical–quantum serverless platforms, emphasizing scheduling, prioritization, and resource management under realistic workload assumptions. Li and Zhao propose Moirai [18], a quantum serverless function orchestration framework that uses circuit-aware representations and reinforcement learning to optimize device allocation and circuit deployment, reporting improved execution performance over baseline methods. These studies contribute valuable performance and orchestration insights, but they are primarily optimization-oriented and do not aim to provide a broader ecosystem-level survey.

Finally, some works discuss Quantum Serverless in broader conceptual settings. Gill and Singh [19] outline a vision in which quantum computing, blockchain, and edge serverless systems may be combined to support secure and efficient edge services. Such contributions are useful for illustrating possible application domains and future directions, but they remain high-level conceptual perspectives rather than systematic comparative reviews.

Compared with these prior works, the present paper takes a different role. Existing contributions are mainly platform-specific, framework-specific, optimization-oriented, or conceptual. In contrast, this survey provides a cross-provider comparative perspective grounded in a unified architecture and an explicit KPI framework. Rather than advancing a single implementation path, it synthesizes heterogeneous industry and research efforts into a structured baseline for assessing the current maturity of Quantum Serverless across hardware access, software ecosystems, queue handling, pricing flexibility, service operability, and Near Real-time readiness.

3.2. Broader NISQ and hybrid quantum context

Quantum Serverless is emerging within the broader technological context of near-term quantum computing, where practical systems remain constrained by noise, limited qubit counts, and strong dependence on classical support infrastructure. The United Nations' designation of 2025 as the International Year of Quantum Science and Technology reflects the growing global visibility of the field, but it also highlights that quantum computing is still in a phase of active technological maturation rather than broad production deployment [20].

A key reason why Quantum Serverless is relevant today is the increasing emphasis on hybrid execution environments that combine classical and quantum resources. EuroHPC and related initiatives have already outlined plans for hybrid HPC–QC infrastructures, including quantum simulators integrated into data centers, cloud-based quantum platforms, proof-of-concept applications, and hardware-agnostic interfaces [21]. Similarly, recent work on Quantum-Accelerated HPC emphasizes that benchmarking, interoperability, and co-located classical–quantum execution are becoming central concerns as these systems move closer to practical use [22].

This broader context is shaped by the current *noisy intermediate-scale quantum* (NISQ) era. Cheng et al. [23] describe NISQ computing as a stage in which medium-scale noisy processors enable important experimental demonstrations, but still fall short of large-scale fault-tolerant operation. In such an environment, quantum processors rarely operate in isolation: they depend on classical systems for orchestration, compilation, pre-processing, post-processing, and resource coordination. This

makes hybrid execution management - rather than hardware access alone - a central issue for the practical evolution of Quantum Serverless [24].

At the same time, long-term progress remains uncertain. Sankar Das Sarma cautions against overestimating the immediacy of quantum breakthroughs, emphasizing that technological progress in this field remains difficult to predict and may still be at an early stage relative to eventual large-scale adoption [25]. This uncertainty reinforces the need to interpret current Quantum Serverless offerings as transitional rather than fully mature infrastructures.

A major step toward more capable systems is the development of logical qubits and quantum error correction. Recent results have shown that error rates can be reduced by scaling encoded qubit structures, providing an important foundation for more reliable quantum computation [26]. Google's more recent progress below the surface-code threshold further strengthens the view that fault-tolerant quantum computing is advancing, although still far from large-scale production systems [27]. Earlier quantum advantage demonstrations also illustrate that specialized quantum processors can outperform classical systems on specific tasks, but such milestones do not yet translate directly into general-purpose, production-grade cloud services [28].

Taken together, these developments explain why current Quantum Serverless platforms remain strongly shaped by queueing, orchestration overhead, hybrid runtime coordination, and backend scarcity. In other words, the broader NISQ and hybrid HPC–QC landscape provides the technological setting in which Quantum Serverless must currently operate: promising, rapidly evolving, but still constrained by the realities of near-term quantum hardware.

Beyond performance, maturity, and orchestration constraints, the broader context of Quantum Serverless also includes security and reliability requirements. Because hybrid quantum–classical workflows depend on cloud-managed control planes, remote backends, and integrity of both software and hardware components, they inherit concerns from cryptography, fault tolerance, and secure system engineering. For this reason, we briefly summarize the most relevant security and reliability considerations that shape the long-term viability of Quantum Serverless.

3.3. Security and reliability considerations

Security and reliability are important background considerations for Quantum Serverless because hybrid quantum–classical workflows depend on trustworthy control planes, secure orchestration, and reliable arithmetic kernels across heterogeneous cloud and hardware environments. In practice, Quantum Serverless inherits concerns from both serverless cloud systems and emerging quantum infrastructures, including identity and access control, cryptographic protection of communication and stored data, resistance to implementation-level attacks, and dependable execution despite hardware faults or noisy environments. These aspects are not the main focus of this survey, but they are relevant for assessing whether Quantum Serverless can evolve toward production-grade deployment.

From a cryptographic readiness perspective, recent work shows that secure implementation can be achieved efficiently even in constrained environments. Side-channel-resistant Ed25519 implementations on ARM Cortex-M4 demonstrate practical secure signatures for edge and embedded contexts [29], while optimized Curve448 architectures show that strong elliptic-curve security can be engineered with competitive efficiency [30]. At the post-quantum level, SIKE implementations on ARM and fault-detection constructions for finite-field inversion further illustrate that implementation hardening and reliability mechanisms can be incorporated into arithmetic kernels relevant to cryptographic pipelines [31,32]. Together with multidisciplinary perspectives on security for safety-critical systems [33], these results indicate that Quantum Serverless platforms should not be evaluated only by functional and performance KPIs, but also by how well they can rely on secure and fault-aware supporting infrastructure.

A second concern is the growing attack surface of hybrid quantum–classical environments. Fault injection, side-channel leakage, and arithmetic bottlenecks in cryptographic components can directly affect orchestration, identity, and data protection layers. Prior work on reliable lightweight cryptographic architectures, efficient FPGA-based SIKE implementations, and high-speed NTT accelerators for post-quantum cryptography suggests that fault tolerance, side-channel awareness, and hardware-level dependability should be treated as relevant design criteria for future Quantum Serverless platforms [34–36]. At the primitive level, design-for-error-detection in nonlinear components, low-cost AES S-box designs, and Ed25519 signature accelerators further confirm that secure performance and fault awareness must be jointly evaluated [37–39].

Security risks in Quantum Serverless also extend beyond hardware noise to the learning, signature, key-exchange, and integrity-verification layers of hybrid workflows. Evidence of systematic poisoning attacks in healthcare machine-learning pipelines highlights the need for explicit data-integrity and model-trust controls when quantum–classical orchestration includes AI-driven decision components [40]. At the same time, highly optimized EdDSA implementations, FPGA-based SIKE/SIDH key-exchange architectures, and reliable hash-function designs show that cryptographic robustness must be balanced with throughput and latency considerations in practical systems [41–43].

Finally, security readiness is not only a matter of algorithms and hardware, but also of engineering practice and human capacity. Efficient ARM-based SIDH implementations show that constant-time post-quantum key-exchange designs can be adapted to constrained platforms without losing practical efficiency [44], while ParallelNTT demonstrates that high-throughput FPGA acceleration for ML-DSA and ML-KEM can significantly reduce latency bottlenecks in post-quantum arithmetic kernels [45]. At the same time, gamified hardware-security education shows that workforce preparedness is itself a security factor: secure-by-design Quantum Serverless systems will depend on engineers who can recognize implementation-level vulnerabilities, side-channel exposure, and fault-related risks during deployment and operation [46]. For this reason, security and reliability should be regarded as enabling conditions for the long-term maturation of Quantum Serverless, even if they lie outside the main comparative scope of this survey.

4. Evaluation methodology

Numerous open questions and challenges influence the successful implementation of Quantum Serverless and its seamless integration with classical computing. Our evaluation methodology compares various approaches and offers a comprehensive state-of-the-art analysis using clear and measurable key performance indicators (KPIs). These KPIs cover essential aspects of the emerging technology ecosystem and its potential as a breakthrough technology.

Identifying cloud platforms and providers reveals information about Quantum Serverless availability and compatibility across different cloud services, which is crucial for widespread adoption. Exploring the variety of quantum hardware and services accessible through Quantum Serverless is essential for understanding its fundamental principles. Quantum application developers are interested in providing programming capabilities and options.

A closer examination of integrating real quantum hardware into Quantum Serverless helps assess its practicality and efficiency in solving real-world quantum problems. Evaluating the presence and capabilities of simulators is essential for testing and experimentation to understand the versatility of Quantum Serverless platforms. The cost-effectiveness of Quantum Serverless solutions is determined by analyzing pricing models and structures. Insights into the potential advantages and limitations of quantum technologies are gained by comparing Quantum Serverless with its classical counterpart.

Considering the impact on efficiency and performance, it is important to examine queueing services to understand how tasks are managed

Cloud Platform	Quantum Provider	Quantum Hardware Architecture	Quantum Hardware Systems
IBM Quantum	IBM	Superconducting Qubits	IBM System One, IBM Hummingbird, IBM Falcon, IBM Eagle, IBM Heron
Azure Quantum	Microsoft, IONQ, Quantinuum, Rigetti, Pasqal, Quantum Circuits, 1Qit, Toshiba	Trapped Ions, Superconducting Qubits, Spin Qubits	IonQ Ion Trap, Honeywell H1 Series, Rigetti Aspen, Quantum Circuits Inc., Toshiba Spin Qubits
Google AI Quantum	Google, IONQ	Superconducting Qubits, Trapped Ions	Google Sycamore, Bristlecone
Amazon Braket	AWS, D-Wave, IONQ, Oxford Quantum Circuits (OQC), QuEra, Rigetti, Xanadu	Superconducting Qubits, Trapped Ions, Quantum Annealing	Rigetti Aspen, IonQ Ion Trap, D-Wave Advantage, OQC Lucy
Alibaba Cloud Quantum Computing Cloud Platform	Alibaba Group	Superconducting Qubits, Spin Qubits	Alibaba Quantum
Quantum Cloud Services (QCS)	Rigetti	Superconducting Qubits	Ankaa-2, Ankaa-9Q-3, Rigetti Aspen
Atos Qaptiva™	IQM, PASQAL, Quandela	Superconducting Qubits, Neutral Atoms, Photonic	N/A
量易伏 QUANTUM LEAF	Baidu	Superconducting Qubits	Origin Quantum
N/A	Intel	Spin Qubits	Spin Qubits in Silicon
N/A	Honeywell	Trapped Ions	Honeywell H1 Series
IONQ Quantum Cloud	IONQ	Trapped Ions	IonQ Harmony, IonQ Aria, Forte, Forte Enterprise
D-Wave Leap	D-Wave	Quantum Annealing	D-Wave Advantage, D-Wave 2000Q
QCi Qatalyst	IONQ, Riggeti, D-Wave	Entropy Quantum Computers	Dirac-1, Dirac-2, Dirac-3
XANADU Cloud	Xanadu	Photonic Qubits	Xanadu X8 Photonic
STRANGE WORKS	AWS, Azure, D-Wave, HITACHI, Honeywell, IBM, IONQ, Rigetti	Trapped Ions, Superconducting Qubits, Spin Qubits	Eagle, IonQ Forte, Ankaa-2
QUANTINUUM	Quantinuum, Honeywell	Trapped Ions	Quantinuum H1-1, Quantinuum H1-2
QCWARE Forge	AWS Braket	Superconducting Qubits, Trapped Ions, Quantum Annealing	Rigetti Aspen, IonQ Ion Trap, D-Wave Advantage, OQC Lucy
PASQAL Cloud services	PASQAL	Neutral Atoms	PASQAL Neutral Atom
OQC private Quantum Computing-as-a-Service (QCaaS)	Oxford Quantum Circuits (OQC)	Superconducting Qubits	OQC Lucy
Q-CTRL	IBM, Rigetti	Superconducting Qubits	IBM Eagle, IBM Heron

Fig. 2. Declarative KPIs evaluating serverless computing in quantum computing hardware.

and processed within Quantum Serverless platforms. In this context, the Near Real-time KPI measures Quantum Serverless's ability to execute tasks with minimal delay, which is especially crucial for applications that need fast or real-time processing.

Quantum Computing resources in our methodology include: *Cloud Platform (CP)* specifies the existing quantum computing cloud platform. *Quantum Provider (QP)* identifies quantum computing hardware providers, including laboratories, some larger cloud providers, and private companies supported by various funds and government agencies. *Quantum Hardware Architecture (QHA)* details the architectures of quantum computing hardware. *Quantum Hardware Systems (QHS)* specifies the quantum computing hardware systems. *Programming (PRG)* lists runtimes, libraries, development frameworks, tools, automation, SDKs, APIs, etc. *Simulators (SIM)* indicates available simulators, simulation tools, and frameworks.

The selected KPIs aim to evaluate Quantum Serverless from an adoption and operability perspective, not just based on raw hardware performance. Declarative KPIs measure ecosystem coverage and practical usability, including provider availability, programming support, simulators, and pricing transparency. Quantifiable KPIs focus on how ready the serverless operation is for execution, covering FaaS/CaaS support, queuing, Near Real-time behavior, pay-as-you-go pricing, and production maturity. This combination allows for comparison across different platforms, especially when direct hardware-level benchmarking is not reported.

4.1. Declarative KPIs

Essential information about the Quantum Serverless that cannot be directly measured specifies the following declarative KPIs:

- *Specification* of *Quantum Computing resource*, including Cloud Platform (CP), Quantum Provider (QP), Quantum Hardware Architecture (QHA), Quantum Hardware Systems (QHS), Programming (PRG), and Simulators (SIM).
- *Pricing (PRC)* explains the pricing model for quantum services.
- *Classical Serverless (CS)* contains a list of offered Function as a Service (FaaS) and Containers as a Service (CaaS) serverless services.

4.2. Quantifiable KPIs

The following measurable KPIs can evaluate the present state of the Quantum Serverless solution:

- *Availability metrics* of *Quantum Computing resource*, including Cloud Platform (CP), Quantum Provider (QP), Quantum Hardware Architecture (QHA), Quantum Hardware Systems (QHS), Programming (PRG), and Simulators (SIM).
- *Function as a Service (FaaS)* reviews the availability of Serverless Function as a Service.
- *Containers as a Service (CaaS)* assesses the availability of Serverless Container as a Service.

- *Queue (Q)* evaluates the use of queuing systems for executing quantum circuits.
- *Near Real-time (Near-RT)* assesses the capability for Near Real-time processing.
- *Pay-as-you-go (PAYG)* reviews the availability of pay-as-you-go pricing models.
- *Quantum Serverless in Production (QS-P)* assesses the readiness of the quantum serverless solution for production deployment.

The scoring system for quantifiable KPIs is as follows:

- *M:* Integrates *multiple* KPI resources.
- ✓*:* presents a built-in resource integrated into the Quantum Serverless solution.
- *X:* means that the Quantum Serverless solution does not implement the KPI.
- *N/A:* specifies unavailable Information about the KPI.

4.3. Excluded KPIs:

For the Classical Data Center, we omitted KPIs for the architecture components (Fig. 1) below the containers layer (VM, Classical Server Hardware).

These classical lower-layer components are excluded because they are well-established, widely standardized infrastructure in cloud systems and do not serve as the key differentiators for Quantum Serverless adoption in this review. Instead, the methodology emphasizes KPIs that directly influence hybrid quantum-classical integration decisions and measurable service capabilities at the platform level.

In the Quantum Data Center, we only included KPIs for the Quantum Hardware Architecture and the name of the Quantum System. Quantum Serverless is primarily a software platform rather than a hardware challenge. This paper does not address the extensive research available on Quantum Hardware, as it is outside the scope of the current study. Multiple architecture components specify each KPI (Fig. 1). For example, Container Orchestration, Containers, VMs, and Classical Server Hardware specify the Queue KPI.

4.4. Performance analysis

This paper relies on the official documentation of quantum providers as an important source of information. Data was gathered manually, and every detail was verified through the official website, documentation, or paper. Besides evaluating KPIs, our methodology also analyzes availability, accessibility, and performance.

Availability pertains to the following declarative KPIs as providers of quantum computing resources: CPs, QPs, QHAs, QHSs, PRGs, and SIMs. To conduct a detailed statistical analysis of the availability of quantum computing resources, our methodology includes the following metrics:
Number of unique resources: The number of available distinct quantum computing resources.
Number of available resources: A count of available quantum computing resources.
Most frequent available resource: Identification of the most frequently offered quantum computing resource.
Trends and Insights: Analysis of the overall trends and insights about quantum computing resources.

Taken together, these KPIs are intended to assess not only the general availability of quantum cloud resources, but more specifically the maturity of *Quantum Serverless* as a managed hybrid execution model. Therefore, the following evaluation should not be read merely as a catalogue of quantum platforms. Rather, it examines the extent to which current offerings move beyond raw backend access toward serverless-like abstractions, including orchestration support, queue-aware execution, pricing flexibility, production readiness, and integration of classical and quantum workflow stages.

Table 1
Availability metrics for quantum computing CPs and QPs.

Unique Quantum Cloud Computing Platforms (CPs)	18
Available Quantum Computing resources offered by CPs	51
Total Quantum Hardware Providers (QPs)	20
In-house Quantum Hardware Providers (QPs)	10 (47.62%)
Multiple In-house Quantum Hardware Providers (QPs)	6 (28.57%)
No In-house Quantum Hardware Providers (QPs)	5 (23.81%)

5. Evaluation of quantum serverless platforms

This section evaluates current platforms through the lens of Quantum Serverless maturity. While some of the examined criteria describe the broader quantum-cloud ecosystem, such as hardware availability, simulators, and programming support, the main purpose of the evaluation is to determine how far current offerings progress from simple remote access to quantum backends toward managed hybrid service models. Accordingly, the results should be interpreted along two complementary dimensions: first, the breadth of the quantum resources and software ecosystems currently available; and second, the extent to which these platforms exhibit serverless-relevant properties such as orchestration support, queue-aware execution, production readiness, granular pricing, and integration of classical and quantum execution flows.

Figs. 2 and 3 present the evaluation of the Declarative KPIs for Hardware and Software platforms, and Fig. 4 the Quantifiable KPIs.

5.1. Analysis of quantum computing resources

5.1.1. Cloud platforms (CPs)

Our evaluation includes CPs (labs and other cloud platforms offering quantum computing), including Rigetti Quantum Cloud Services (QCS) [47] from Rigetti Quantum Provider, Atos [48], Baidu [49], IONQ [50], D-Wave [51], Xanadu [52], PasQAL [53]. IBM Quantum [54], Azure Quantum [55], Google Cloud AI [56], and AWS Braket [80] are attractive cloud providers that successfully emerged in the market. Intel [57] and Honeywell [58] do not have their cloud platforms, as they provide only Quantum Hardware.

Fig. 5 presents the availability of quantum computing cloud platforms (CPs).

5.1.2. Quantum providers (QPs)

Quantum Hardware Providers offer Quantum Hardware on a particular quantum computing cloud platform. Multiple Quantum Hardware Providers might be integrated into one quantum computing cloud platform. For example, Azure Quantum is an example of a cloud platform CP offering many Quantum Hardware Providers (QPs), including Microsoft, IONQ, Quantinuum [59], Rigetti, Pasqal, Quantum Circuits [60], 1Qbit [61], Toshiba [62].

Some quantum computing cloud platforms do not have their own quantum hardware and focus only on developing quantum software, including QCRTL [63] and Strangeworks [64].

Fig. 5 shows the number of quantum hardware in quantum computing cloud providers (CPs). Table 1 displays measurable KPIs for quantum computing availability. The analysis leads to the following conclusions:

Eighteen unique quantum providers are analyzed in Fig. 2. The most common quantum hardware is IONQ, offered by eight labs and quantum cloud providers. IONQ and Rigetti hardware frequently coexist on quantum cloud platforms, including Amazon Braket and Azure Quantum. Honeywell and Quantinuum are often associated, indicating a partnership or close collaboration. IBM and Rigetti are regularly listed as available quantum computing hardware providers across multiple platforms. Quantum providers such as IBM, Rigetti, and D-Wave are major players, commonly distributed across various platforms. Several providers, including Hitachi, Microsoft, and Alibaba, are less frequently featured, suggesting a more niche or specialized market presence. 52.38% of

Cloud Platform	Programming	Simulators	Pricing	Classical Serverless
IBM Quantum	Qiskit	Clifford simulator, Matrix Product State, Extended Clifford (e.g. Clifford+T), General, context-aware, Schrödinger wavefunction	Pay-as-you-go, Premium Plan, Open Plan	IBM Cloud Code Engine, IBM Cloud Functions
Azure Quantum	Cirq, Qiskit, Q#	Full state simulator, Sparse simulator, Simple resources estimator, Trace-based resource estimator, Toffoli simulator, Noise simulator	Pay-as-you-go, Custom plans	Azure Container Apps, Azure Functions
Google AI Quantum	Cirq, OpenFermion, TensorFlow Quantum	Quantum Virtual Machine (QVM), qsim - Optimized quantum circuit simulators, GPU-based quantum simulation on Google Cloud, FQE	Pay-as-you-go, Custom plans	Google Cloud Run, Goolge Cloud Functions
Amazon Braket	Amazon Braket, Qiskit provider for Amazon Braket	Local state vector simulator (braket_sv) ('Default Simulator'), Local density matrix simulator (braket_dm), State vector simulator (SV1), Density matrix simulator (DM1), Tensor network simulator (TN1), PennyLane's Lightning Simulators	Free-Tier, Per-task price, Per-shot price, Per-minute rate, Price per Minute	AWS Fargate, AWS Elastic Compute Service ECS, AWS Lambda
Alibaba Cloud Quantum Computing Cloud Platform	Alibaba Cloud Quantum Development Platform (ACQDP)	general-purpose, tensor-contraction based quantum circuit simulator, error correction simulations	×	Elastic Container Instance, Function Compute, Alibaba Cloud Container Service for Kubernetes (ACK)
Quantum Cloud Services (QCS)	Cirq, Qiskit, Q#, pyQuil, quilc, QVM	WavefunctionSimulator, ReferenceWavefunctionSimulator, ReferenceDensitySimulator, NumpyWavefunctionSimulator	Pay-as-you-go, Custom plans	×
Atos Qaptiva™	myQLM, Qiskit, Cirq, ProjectQ or Forest	Atos Quantum Learning Machine (Atos QLM), Atos QLM Enhanced (Atos QLM E)	N/A	×
量易伏 QUANTUM LEAF	Qcompute	QCompute OpenSimulator	×	×
N/A	Quantum SDK (C++)	Intel Quantum Simulator (Intel-QS), also known as qHiPSTER (The Quantum High Performance Software Testing Environment)	×	×
N/A	Cirq, Qiskit, Q#	Syntax Validation tool, H1-1 Emulator, H1-2 Emulator	×	×
IONQ Quantum Cloud	Cirq, Qiskit, Q#, Amazon Braket, blqs	IONQ Quantum simulator	Volume Based Pricing, Request Custom Pricing	×
D-Wave Leap	D-Wave SDK, Ocean Tools, Qiskit	Simulated annealing	Free-Trail, Custom plans, Pay-as-you-go	×
Qatalyst	Qatalyst API (Q Api)	braket_simulator, braket_simulator_tn1, braket_simulator_dm1, and qonductor. Braket's simulators: SV1 (state vector simulator), TN1 (based on tensor networks), and DM1 (density matrix simulator)	Pay-as-you-go, AWS Braket (Free-Tier, Per-task price, Per-shot price, Per-minute rate, Price per Minute)	×
XANADU Cloud	PennyLane, Strawberry Fields	Lightning quantum simulator, Strawberry Fields Simulator, Jet	Free Tier, Flexible Tier, Partner Tier	×
STRANGE WORKS	Strangeworks QC	Strangeworks Quantum Simulator	Subscription-based, Per-month, Annually	×
QUANTINUUM	TKET, Qermit, lambeq	H1-1 Emulator, H1-2 Emulator	Pay-as-you-go (H-System Quantum Credits (HQCs))	×
QCWARE Forge	AWS Braket, Quasar library	Quantum Approximate Optimization Algorithm (QAOA) QAOA qcware/cpu_simulator, QAOA qcware/gpu_simulator	Free (Developer Plan), Enterprise (Pay-As-You-Go, Small, Medium, Large)	×
PASQAL Cloud services	Qadence, Pulser	Fresnel quantum simulator from PASQAL	Per-shot price, Per-hour price	×
OQC private Quantum Computing-as-a-Service (QCaaS)	Amazon Braket Python SDK	N/A	Custom plans	×
Q-CTRL	Q-CTRL Python package, Open Controls, IBM Qiskit, Quantum Machines, Rigetti Quil, QuTiP	coherent_simulation, noise_simulation, Simulate quantum dynamics for noiseless systems using graphs, Simulate multi-qubit circuits in quantum computing, Simulate open system dynamics, Simulate large open system dynamics	Pay monthly (Basic plan - free, Pro, Enterprise), Pay annually (Basic plan - free, Pro, Enterprise)	×

Fig. 3. Declarative KPIs evaluating serverless computing in quantum computing software.

quantum cloud platforms explicitly list "Available" without detailing multiple options for users. 47.62% offer multiple quantum hardware options. 76.19% of platforms provide in-house quantum hardware. Market trends indicate ongoing development and investment in proprietary quantum technologies, leading to an increase in-house quantum hardware. Platforms with in-house hardware may offer more direct control, better performance, or easier integration for researchers and developers. Additionally, the unspecified status highlights areas where more transparency or detail may be needed.

5.1.3. Quantum hardware architecture (QHA)

We identify nine unique quantum hardware architectures in quantum computing platforms and other specialized architectures in research labs. Fig. 6 presents the count of quantum hardware architectures in quantum computing platforms.

Table 2 matches the quantum computing cloud platforms (CPs) versus the offered Quantum Hardware Architectures (QHAs).

We conclude the following: The most common QHA is Superconducting Qubits offered by 12 CPs. Trends indicate the dominance of

Cloud Platform	Quantum Provider	Available Quantum Hardware	In-house Quantum Hardware	Programming	Simulators	FaaS	CaaS	Queue	Near RT	Pricing	Pay-as-you-go	QS in Production
IBM Quantum	✓	M	M	M	M	✓	✓	✓	✗	M	✓	✓
Azure Quantum	M	M	✓	M	M	✓	✓	✓	✗	M	✓	✗
Google AI Quantum	M	M	✓	M	M	✓	✓	✓	✗	M	✓	✗
Amazon Braket	M	M	✓	M	M	✓	✓	✓	✗	M	✓	✗
Alibaba Cloud Quantum Computing Cloud Platform	✓	✓	✓	✓	M	✓	✓	✓	✗	✗	✗	✗
Quantum Cloud Services (QCS)	✓	M	M	M	M	✗	✗	✓	✗	M	✓	✗
Atos Qaptiva™	M	M	✗	M	M	✗	✓	✓	✗	N/A	N/A	✗
量易优 QUANTUM LEAF	✓	✓	✓	✓	M	✓	✓	✓	✗	✗	✗	✗
Intel N/A	✓	✓	✓	✓	M	✗	✓	✓	✗	✗	✗	✗
Honeywell N/A	✓	✓	✓	M	M	✗	✗	✓	✗	✗	✗	✗
IONQ Quantum Cloud	✓	M	M	M	M	✗	✗	✓	✗	M	✓	✗
D-Wave Leap	✓	M	M	M	M	✗	✗	✓	✓	M	✓	✗
QCi Qatalyst	M	M	M	M	M	✗	✗	✓	✗	M	✓	✗
XANADU Cloud	✓	✓	✓	M	M	✗	✗	✓	✗	M	✗	✗
STRANGE WORKS	M	M	✗	M	M	✗	✗	✓	✗	M	✗	✗
QUANTINUUM	M	M	M	M	M	✗	✗	✓	✗	M	✓	✗
QCWARE Forge	M	M	✗	M	M	✗	✗	✓	✗	M	✓	✗
PASQAL Cloud services	✓	✓	✓	M	M	✗	✗	✓	✗	M	✓	✗
OQC private Quantum Computing-as-a-Service (QCaaS)	✓	✓	✓	M	M	✗	✗	✓	✗	✓	✗	✗
Q-CTRL	M	M	✗	M	M	✗	✗	✓	✗	M	✗	✗

Fig. 4. Quantifiable KPIs evaluating serverless computing in quantum computing platforms and hardware.

Superconducting Qubits, accounting for 38.71% of all architectures, suggesting that many platforms and research efforts focus on superconducting technology, likely due to its maturity and scalability. The notable presence of Trapped Ions at 22.58% highlights their importance in the current quantum computing landscape, owing to their high-fidelity gate operations and long coherence times, which are well-suited to various quantum applications. Emerging and specialized architectures include Quantum Annealing at 9.68%, primarily in specialized applications rather than general-purpose quantum computing; Photonic Qubits at 6.45%, used in quantum communication and certain computational models that benefit from light-based operations; and Neutral Atoms at 6.45%, valued for their scalability potential and unique approach to quantum gate operations. The wide variety of quantum hardware indicates that no single architecture is universally considered the best and that various approaches are used to solve quantum problems, from general-purpose computing to specialized optimization tasks.

Table 3 displays the Quantum Hardware Systems (QHAs) available on each Cloud Platform (CP). It is important to note that some Quantum Systems are accessible through multiple Cloud Platforms. IBM Quantum stands out with the number of different QHAs it offers, totaling five, while the Rigetti Aspen system is notable for being available on four different platforms.

5.2. Programming

Writing quantum programs (circuits) requires knowledge of a programming language (Python, C++, etc.) and the theoretical foundations of quantum programming. Developing tools, frameworks, SDKs, APIs, and other software utilities helps create quantum serverless programs, and the ease of use of new technologies makes quantum programming accessible to a broader audience of scientists, developers, and the research community in general.

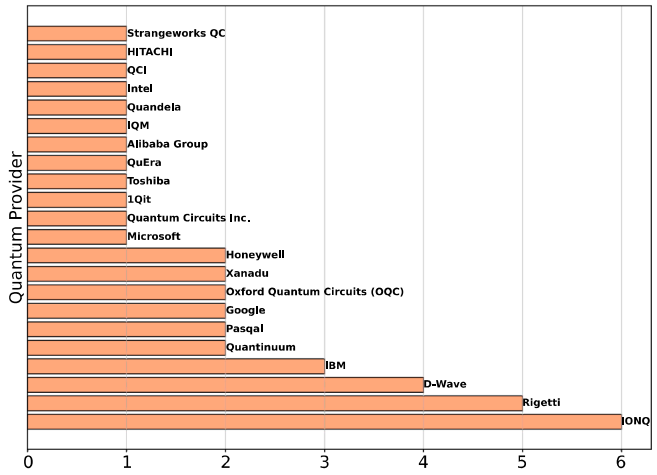


Fig. 5. The number of cloud platforms where the quantum provider is available.

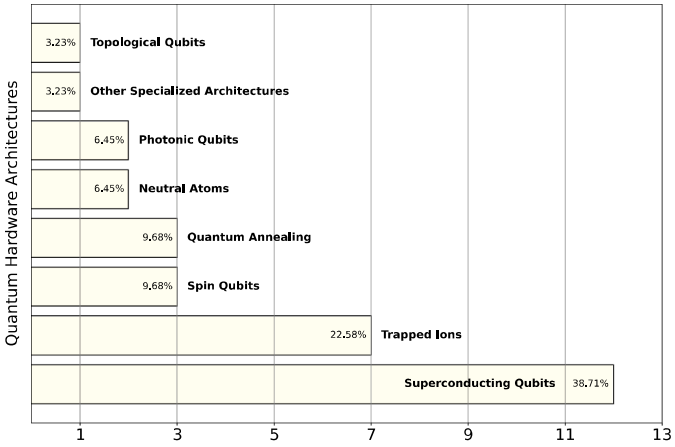


Fig. 6. Quantum hardware architectures in quantum computing cloud platforms (CPs).

Table 2
Matching of cloud platforms (CPs) versus offered Quantum Hardware Architecture (QHA).

Platform	SQ	TI	QA	PQ	NA	SpQ	TQ	OSA
IBM Quantum	✓							
MS Azure Quantum	✓	✓				✓		
Google AI Quantum	✓	✓						
Amazon Braket	✓	✓	✓					
Alibaba Cloud	✓					✓		
Xanadu Cloud				✓				
D-Wave Leap			✓					
Atos Qaptiva	✓			✓	✓			
Rigetti QCS	✓							
IONQ Quantum Cloud		✓						
QCI Qatalyst							✓	✓
Origin Quantum (Baidu)	✓							
Strangeworks	✓	✓				✓		
Quantinuum		✓						
QC Ware Forge	✓	✓	✓					
PASQAL Cloud					✓			
OQC QCaaS	✓							
Q-CTRL	✓							

SQ = Superconducting Qubits, TI = Trapped Ions, QA = Quantum Annealing, PQ = Photonic Qubits, NA = Neutral Atoms, SpQ = Spin Qubits, TQ = Topological Qubits OSA = Other Specialized Architectures

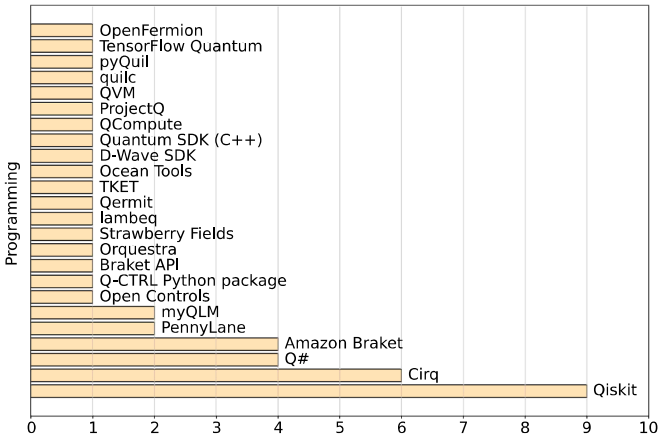


Fig. 7. Programming tools, SDKs, and APIs used in quantum computing platforms.

Fig. 7 presents the frequency of programming tools, SDKs, and platform-specific APIs used across different quantum platforms.

Table 4 presents the matching of the quantum cloud platform versus the offered programming tools.

The in-depth analysis yields the following conclusion: Qiskit is the most frequently used programming language, with widespread adoption across multiple platforms. Cirq is also widely used, especially by Google and its partners. Q# is prominent in Microsoft's ecosystem and some IONQ integrations. Amazon Braket is specific to AWS environments, Xanadu primarily uses PennyLane and Strawberry Fields. Quantum SDK (C++) is specific to Intel's quantum initiatives.

5.3. Simulators

Simulators are used to ease the development and testing of quantum programs. They simulate a real quantum computer, and a classical computer executes the simulation. The simulators do not match the speed of a quantum computer, but they simulate the results that would be obtained on one. Different quantum computers use their underlying physical principles and offer various simulators (Fig. 3).

An example is a simulation of noise that could be produced with current technology on actual quantum hardware. Some of these simulators can run on a personal computer, while others are offered as a service by cloud providers because high-performance computing (HPC) machines, such as multi-core or many-core (GPU-based) systems, are used for the simulation. HPC machines are used for simulations that would take a very long time to run on a personal computer, but we prefer to avoid using real quantum hardware due to unresolved scientific challenges or financial considerations. The following conclusions resulted from our in-depth analysis:

Major simulators

State Vector Simulator (SV1), Density Matrix Simulator (DM1), and Tensor Network Simulator (TN1) are predominantly used in multiple quantum platforms for general quantum circuit simulation tasks. *Atos Quantum Learning Machine (QLM)* is widely used by platforms that rely on Atos quantum software. *H1-1 and H1-2 Emulators* are specific to Quantinuum's platforms for emulating their trapped ion quantum hardware.

Specialized simulators

Qiskit Simulators (Clifford, Matrix Product State, etc.) focus on IBM Quantum platforms. *Quantum Virtual Machine (QVM)* and *qsim* are used by Google and its partners. *Simulated Annealing (D-Wave)* is specific to

Table 3
Cloud platforms and their quantum hardware systems.

Cloud Platform	IBM System One	IBM Hummingbird	IBM Falcon	IBM Eagle	IBM Heron	IonQ Ion Trap	IonQ Harmony	IonQ Aria	IonQ Forte	Honeywell H1 Series	Quantinuum H1-1	Quantinuum H1-2	Rigetti Aspen	Ankaa-2	Ankaa-9Q-3	D-Wave Advantage	D-Wave 2000Q	OQC Lucy	Others
IBM Quantum	✓	✓	✓	✓	✓														
Azure Quantum						✓				✓			✓						✓
Google AI Quantum																			✓
Amazon Braket						✓							✓			✓		✓	
Alibaba Cloud																			✓
Rigetti QCS													✓	✓	✓				
Atos Qaptiva																			✓
Baidu																			✓
N/A (Intel)																			✓
N/A (Honeywell)										✓									
IONQ Quantum Cloud							✓	✓	✓										✓
D-Wave Leap																✓	✓		
QCI Qatalyst																			✓
Xanadu Cloud																			✓
Strangeworks				✓					✓					✓					
Quantinuum											✓	✓							
QC Ware Forge						✓							✓			✓		✓	
PASQAL Cloud																			✓
OQC QCaaS																		✓	
Q-CTRL				✓	✓														

Table 4
Matching of quantum cloud platforms versus programming tools.

Platform	QS	Cirq	Q#	AB	OF	LM	TF	PL	SF	DW	OC	MQ	PQ	QT	blqs
IBM Quantum	✓														
Google AI Quantum		✓			✓		✓								
MS Azure Quantum	✓	✓	✓												
Amazon Braket	✓			✓											
Xanadu Cloud								✓	✓						
D-Wave Leap	✓									✓					
Atos Qaptiva	✓	✓									✓	✓			
Rigetti QCS	✓	✓	✓												
IONQ Quantum Cloud	✓	✓	✓	✓											✓
QCI Qatalyst														✓	
Alibaba Quantum		✓	✓												
Baidu Quantum Leaf												✓			
Quantinuum						✓									
QC Ware Forge				✓											
OQC private QCaaS	✓			✓											
Q-CTRL	✓												✓		

QS = Qiskit, AB = Amazon Braket API, OF = OpenFermion, LM = Lambeq, TF = TensorFlow Quantum, PL = PennyLane, SF = Strawberry Fields, DW = D-Wave, OC = Ocean, MQ = myQLM, PQ = ProjectQ, QT = Qatalyst, blqs = blqs (IONQ)

D-Wave's quantum annealing hardware. *Strawberry Fields Simulator* focuses on Xanadu Cloud photonic quantum computing. *Fresnel Quantum Simulator* is specific to PASQAL's neutral atom quantum platform.

Platform-specific simulator usage

IBM Quantum utilizes Qiskit simulators (Clifford, Matrix Product State, etc.) for various quantum circuit simulations. *Google AI Quantum* uses Quantum Virtual Machine (QVM), qsim, and FQE for different levels of quantum simulation. *Amazon Braket* provides simulators like State Vector Simulator (SV1), Density Matrix Simulator (DM1), and Tensor Network Simulator (TN1). *Xanadu Cloud* uses Strawberry Fields and PennyLane simulators for photonic quantum simulations. *Quantinuum* employs H1-1 and H1-2 emulators specific to their trapped ion hardware. *Rigetti* provides WavefunctionSimulator, ReferenceWavefunctionSimulator, and ReferenceDensitySimulator for different quantum computations. *D-Wave Quantum Cloud* uses Simulated Annealing for quantum annealing tasks. *PASQAL Cloud Services* provides the Fresnel Quantum Simulator for neutral atom simulations.

5.4. Quantum serverless

5.4.1. Classical serverless (CS)

- IBM Cloud [65], Azure [66], Google Cloud [67], AWS [68], Alibaba Cloud [69] offer classical serverless services. The rest focus solely on quantum computing and cloud computing. The analysis shows that most quantum computing platforms either do not specify or do not use specific classical serverless platforms. Among those that do specify, various options are employed, including proprietary solutions from cloud providers like AWS, Azure, Google Cloud, IBM, and Alibaba. The diversity of classical serverless platforms reflects how quantum computing services are integrated with different serverless solutions, depending on the provider's ecosystem. This variety suggests that while some

providers offer direct integration with specific serverless platforms, others either use their own solutions or do not specify any particular platform.

5.4.2. Function as a service (FaaS)

- Platforms that offer FaaS are: AWS Lambda [70], Google Cloud Functions [71], Azure Functions [70], IBM Functions [70], Alibaba Function Compute [70]. Total platforms: 20, platforms with FaaS (✓): 6 $\frac{6}{20} \times 100 \approx 30\%$ and platforms without FaaS (**X**): 14 $\frac{1}{20} \times 100 \approx 70\%$.

5.4.3. Container as a service (CaaS)

Serverless Containers are available by IBM Cloud Code Engine [72], Azure Container Apps [73], Google Cloud Run [74], AWS Fargate [75] and AWS Elastic Container Service [76], Alibaba Elastic Container Instance [77]. Total platforms: 20, platforms with CaaS (✓): 8 $\frac{8}{20} \times 100 \approx 40\%$ and platforms without CaaS (**X**): 12 $\frac{12}{20} \times 100 \approx 60\%$.

5.4.4. Quantum serverless in production (QS-P)

Our evaluation of Quantum Serverless in Production (QS-P) (Fig. 4) showed that IBM Quantum stands out as the only provider that meets all key performance indicators (KPIs) except for Near Real-time responses.

Substantial challenges persist from Quantum Hardware to the application level, constraining the realization of a fully functional Quantum Serverless implementation. IBM Quantum Serverless is available exclusively to researchers with premium access to IBM's quantum processing units (QPUs) via Qiskit Serverless. Future updates can evolve Qiskit Serverless into the foundation for Qiskit Serverless Functions.

5.4.5. Queue (Q)

Fig. 4 presents that all platforms (100.00%) offer Queue. This indicates that Queue support is universally available across all platforms, reflecting its essential role in task management and orchestration. The universal availability of Queue support suggests it has become a standard feature across quantum cloud platforms, underscoring its importance in modern cloud computing environments.

5.4.6. Near real-time (near-RT)

- We adopt a 3-second response time threshold (≤ 3s) as a pragmatic benchmark for "Near Real-time" quantum serverless execution, grounded in established human-computer interaction (HCI) standards which dictate that delays exceeding this limit disrupt the cognitive flow of interactive cloud services [78]. While this threshold is several orders of magnitude slower than the microsecond-scale latencies required for hardware-level operations, such as those necessitated by rapid qubit decoherence in quantum error correction [23]. It serves as a critical operational boundary for the API and workflow layers. By targeting this window, we ensure that hybrid iterative workflows (e.g., VQE and QAOA) maintain the responsiveness required for interactive development loops, distinguishing them from traditional batch- or queued-execution models. Consequently, our analysis reveals that current platforms fail to achieve this ≤ 3s benchmark, largely due to the physical scarcity of quantum processing units and the significant orchestration overheads inherent in the current state of quantum hardware integration.

5.5. Cost effectiveness

5.5.1. Pricing

Our evaluation (Fig. 3) presents several models for charging for the execution of quantum programs on simulators and real quantum hardware.

Most common pricing models

Pay-as-you-go: The most prevalent pricing model used by 14 quantum platforms. This model allows users to pay based on their actual usage, which offers flexibility and scalability. *Custom Plans*: Adopted by 10 platforms. This model provides tailored pricing based on the user's specific needs and usage patterns, often negotiated on a case-by-case basis. *Free-Tier*: Available for 5 platforms. This model offers free access, which can be useful for users to try out services before committing to paid plans.

Less common pricing models

Subscription-based: Used by 2 platforms. This model involves paying a regular fee for access to quantum computing resources, typically monthly or annually. *Volume-Based Pricing*: Applied by 1 platform. This model charges based on usage volume, such as the number of quantum operations performed. *Per-task price, Per-shot price, Per-minute rate, Price per Minute, Per-hour price*: Each used by one platform. These models charge based on specific usage units, such as per task, per shot, per minute, or per hour.

Detailed Analysis

Pay-as-you-go model

Most platforms favor the *Pay-as-you-go* model because it aligns costs with usage, enabling greater flexibility and cost control. This model is ideal for users with variable or unpredictable usage patterns.

Custom plans

The *Custom plans* model offers a more personalized approach, often involving negotiation with the service provider to tailor the pricing structure according to the user's specific needs and expected usage. This model is standard among enterprise users with significant or specialized requirements.

Free-tier

The *Free-Tier* model provides basic or limited access at no cost, helping users explore the platform's capabilities before making a financial commitment. It also serves as an entry point for new users and developers.

Subscription-based model

The *Subscription-based* model offers a fixed price for a set period, which can benefit users with predictable usage patterns. It provides a stable cost structure and often includes access to premium features or higher usage limits.

Volume-based and per-unit pricing

Volume-Based Pricing and various *Per-unit pricing* models (such as per-task, per-shot, per-minute, or per-hour) offer specific cost structures based on usage metrics. These models can benefit users with well-defined usage patterns or those needing detailed billing granularity.

5.5.2. Pay-as-you-go

Total platforms: 20, platforms with pay-as-you-go pricing (✓): 11 $\frac{11}{20} \times 100 \approx 55\%$, platforms without pay-as-you-go pricing (**X**): 8 $\frac{8}{20} \times 100 \approx 40\%$ and N/A (**N/A**): 1 $\frac{1}{20} \times 100 \approx 5\%$. For users, pay-as-you-go pricing enables better cost management and scalability. Platforms without this option may need to consider adopting more flexible pricing models to meet the needs of users who prefer or require such arrangements.

Overall, the evaluation shows that the current ecosystem combines growing quantum-cloud availability with only partial realization of Quantum Serverless as a managed execution model. Support for hardware access, SDKs, simulators, and queueing is already widespread, indicating that the basic ingredients for hybrid quantum–classical workflows are increasingly available. However, stronger serverless characteristics remain unevenly distributed: only a minority of platforms provide Function-as-a-Service, Containers-as-a-Service, production-grade maturity, or Near Real-time responsiveness. This confirms that today's landscape is better understood as an early, queue-driven hybrid orchestration ecosystem than as a fully mature serverless computing layer for quantum applications.

6. Discussion

The evaluation results suggest that the main challenge for Quantum Serverless is no longer only the availability of quantum backends, but the maturity of the surrounding software and service ecosystem. In particular, the gap between widespread queue-based access and limited support for stronger serverless characteristics indicates that most current platforms remain at an intermediate stage: they expose hybrid execution paths, but only partially abstract the orchestration, deployment, and operational complexity around them. The following discussion therefore focuses on the main bottlenecks that currently limit Quantum Serverless maturity and on the conditions under which the field may evolve toward production-grade hybrid cloud services.

6.1. Quantum hardware layer challenges

As quantum serverless addresses many challenges for wider implementation, we examine its limitations and shortcomings.

Error correction: Maintaining qubit stability is paramount, as qubits are extremely sensitive to environmental factors, like electromagnetic interference and temperature. Therefore, error correction is crucial in quantum hardware due to decoherence and other Quantum phenomena.

Working environment: Quantum hardware cryogenic requirements to operate near absolute zero introduce costly and intricate temperature control systems.

Qubit connectivity: Achieving efficient qubit connectivity for quantum gate operations and establishing long-distance quantum communication channels are hot topics in research.

Scalability: Scalability is another concern, as scaling up quantum hardware while preserving qubit coherence and connectivity is a complex engineering problem.

Manufacturing: Developing manufacturing processes and addressing scalable resource constraints pose significant challenges. In addition, there is an increasing need to establish industry standards to specify future compatibility and reduce the costs of intended solution providers.

Integration: Moreover, integrating diverse quantum technologies, such as superconducting qubits and trapped ions, within a single quantum processor adds complexity.

Software interface: The challenges in the quantum hardware layer related to software interfaces encompass several aspects. Standardizing interfaces is essential to promoting compatibility, simplifying software development, and attracting a broader community to use novel technologies.

Resource Management: Managing the complexity of quantum hardware effectively requires the right balance between abstraction and control. Efficient resource management, error handling, and performance optimization are ongoing challenges.

Benchmarking: Developing end-to-end quantum benchmarking for Quantum Serverless is challenging due to the complexity of measuring performance across diverse cloud-based quantum resources, varying execution environments, and the need for consistent, scalable metrics that account for classical and quantum computation dynamics.

Xavier Geoffret outlines five distinct levels of benchmarking [79]: Component Level: Focuses on fundamental physics and error rates. Processor Level: Evaluates qubit interactions and crosstalk. Fundamental Physics Level: Measures quantum capabilities such as entangled state generation. Application Level: Assesses task-specific performance metrics using tools like Q-Score. Speed Level: Analyzes execution speed with metrics such as CLOPS (Circuit Layer Operations Per Second). Furthermore, Quantum Serverless benchmarking requires a comprehensive framework that integrates existing benchmarking levels while accounting for dynamic resource allocation, latency, data transfer, interoperability, hybrid workload orchestration, network overhead, and algorithm scalability in a serverless environment.

6.2. User perspectives

Quantum Software: Development Tools, Frameworks, and Libraries play pivotal roles in Quantum Serverless. They specify the environment that will allow new high-level programming tools to use the infrastructure efficiently. Quantum compilation and visualization/debugging tools require development to facilitate quantum programming. Building a comprehensive quantum software ecosystem and ensuring user-friendliness are also vital considerations. Overcoming these challenges is essential for advancing quantum computing and making it accessible to a broader range of researchers and developers.

Compatibility: It is a balancing act to harness the power of quantum hardware efficiently while abstracting its complexity for developers. Cross-platform compatibility is another challenge, as there are no associated standards. Vendor-specific solutions will emerge soon, and only the most successful will promote their specifications as open platforms so that others may develop new solutions.

Optimization: Optimizing the efficient use of quantum infrastructure and simplifying high-level programming present another challenge: making the interface easier to use. Additionally, accessible, high-level programming tools and frameworks help facilitate the development of quantum applications.

Integration: The step-by-step introduction of Quantum Serverless into existing computing ecosystems presents integration challenges, especially in achieving seamless integration between quantum programming tools and quantum hardware. It is complex to integrate quantum capabilities incrementally while maintaining compatibility with classical systems. Ensuring the seamless coexistence of Quantum Serverless architectures with classical serverless architectures and gradually replacing or augmenting classical components with quantum counterparts without disruption requires careful planning and execution. The integration process must be well-managed to maximize the benefits of quantum computing while minimizing disruptions to existing workflows and systems.

6.3. KPI - Challenges

Quantum Provider and Available Quantum Hardware: Not all cloud platforms offer multiple quantum providers, which reduces flexibility for users requiring diverse architectures. Platforms featuring multiple quantum providers include Azure Quantum, Google Quantum AI, Amazon Braket, Qaptiva, Qatalyst, Strangeworks, Quantinuum, QC Forge, and Q-CTRL.

In-House Quantum Hardware: Major cloud platforms that stand out for having multiple in-house quantum hardware include IBM Quantum, Rigetti, IONQ, D-Wave, Qatalyst, and Quantinuum.

Programming and Simulators: Most platforms offer programming environments, but the depth of support varies. Simulators are widely available, though not universally present.

Function-as-a-Service (FaaS) and Container-as-a-Service (CaaS): IBM Quantum, Azure Quantum, Google Quantum AI, Amazon Braket, Alibaba Quantum Cloud Computing, and Baidu support classical FaaS. None of the others offer classical FaaS on the cloud platform. The same applies to CaaS, with only two more providers supporting it: Qaptiva and Intel.

Queue and Near Real-time (RT) Processing: A queue system is available on all platforms, but Near Real-time quantum computation remains challenging. No cloud platform supports Near Real-time requests for quantum circuit processing.

Pricing and Pay-as-You-Go Models: Most cloud providers offer multiple pricing models, but Alibaba, Baidu, Intel, and Honeywell lack transparent pricing structures. More than half of the cloud platforms support pay-as-you-go in some models.

Quantum Systems in Production: IBM is the only company actively using quantum systems in production. Most platforms are still in the experimental or research phases.

Most KPIs for Quantum Serverless are met by IBM Quantum (only one KPI not met), Azure Quantum (two KPIs not met), Google Quantum AI (two KPIs not met), and Amazon Braket (two KPIs not met). All other platforms have three or more KPIs that are not met.

Uncovered gaps include the lack of standardized quantum infrastructure, with platforms relying on third-party quantum hardware, leading to fragmentation in quantum services. Additionally, no platform has achieved Near Real-time quantum execution, limiting real-time capabilities. Some platforms also lack flexible pricing models, such as pay-as-you-go options. There is a gap in Function-as-a-Service (FaaS) and Container-as-a-Service (CaaS) in specialized quantum platforms, as traditional cloud providers like IBM, Azure, and AWS lead in cloud-native quantum services, while hardware-centric companies like Rigetti, Honeywell, and IONQ lack these features. Finally, although some platforms offer production-ready systems, most remain experimental, delaying mainstream adoption.

6.4. Raising awareness

Realizing a breakthrough application will help spread the idea of quantum computing more broadly. Popularizing Quantum Serverless computing and making it accessible to a wide range of users can lead to groundbreaking advances in quantum computing, algorithm development, and problem-solving. The dissemination of quantum concepts is crucial to achieving widespread adoption of Quantum Serverless. Challenges include the need to ensure a broader understanding of quantum computing principles, clearly explain the specific problem areas that quantum technology can effectively address, and bridge the gap by making quantum concepts more accessible to a larger audience. Overcoming misconceptions, demystifying quantum computing, and fostering familiarity with quantum concepts are essential efforts.

Additionally, the ongoing challenge is precisely defining practical problems that can benefit greatly from quantum solutions and making these complex concepts relatable and accessible to people without an extensive background in quantum physics. This effort to popularize Quantum Serverless has the potential to unlock breakthroughs in quantum computing, transforming algorithm development and solving problems that have long seemed insurmountable with classical computing methods.

6.5. Quantum serverless ecosystem limitations

From a technology-diversity perspective, the Quantum Serverless ecosystem is dominated by superconducting and trapped-ion systems. Superconducting qubits account for 38.71% of observed architectures, and trapped ions for 22.58%, while quantum annealing (9.68%), photonic (6.45%), and neutral-atom approaches (6.45%) remain less prevalent. The concentration around two architecture families suggests that current Quantum Serverless interfaces are being optimized around the most operationally mature hardware stacks, while alternative architectures remain important but less represented in production-like cloud offerings.

The serverless-readiness KPIs highlight the largest gap between platform availability and operability. Queue support is universal (100%), confirming that asynchronous execution is already a basic requirement for quantum workloads. However, only 30% of platforms offer FaaS, 40% offer CaaS, and 55% provide pay-as-you-go pricing (40% do not support it and 5% do not report it). Most notably, 0% meet the Near-RT KPI (response time <= 3 seconds), and only one provider (IBM) is considered production-ready according to Quantum Serverless KPIs. Overall, infrastructure-level access has grown faster than cloud-native developer experience, economic flexibility, and interactive latency.

A cross-platform comparison also explains why a small group of providers consistently scores better. IBM, Azure, Google, and Amazon Braket perform strongly because they combine broader provider/hardware access with stronger software ecosystems, integrated APIs/SDKs, and clearer cloud-service pathways from experimentation to deployment. In contrast, platforms focused primarily on hardware exposure or specialized tooling often satisfy fewer end-to-end service KPIs (especially FaaS/CaaS, production readiness, and pricing transparency). Therefore, the current competitive advantage is less about a single hardware breakthrough and more about the maturity of orchestration across software, operations, and business model layers.

The KPI ranking pattern further supports this interpretation: IBM is the only platform with one unsatisfied KPI, Azure/Google/Amazon Braket each have two unsatisfied KPIs, and all remaining platforms have three or more unsatisfied KPIs. This distribution indicates a steep maturity gradient rather than a smooth continuum across providers, with most platforms still in pre-production or partially integrated operating modes.

6.6. Availability, cost dimensions, and adoption

In programming support, Qiskit appears to be the most prevalent ecosystem, while Cirq and Q# form secondary clusters aligned with Google- and Microsoft-related environments. In simulation support, general-purpose simulators (e.g., state-vector, density-matrix, tensor-network) coexist with provider-specific simulators and emulators, reflecting both functional breadth and ecosystem fragmentation. On pricing, pay-as-you-go is the most common model (14 platforms), followed by custom plans (10) and free tiers (5), while subscription and volume-based models remain limited; this confirms that economic access is improving but still lacks full transparency and standardization across providers.

At the hardware-system level, availability is also concentrated: IBM stands out with multiple distinct in-house systems, and specific systems such as Rigetti Aspen appear across several cloud platforms. This recurring multi-platform exposure suggests that interoperability today is often achieved through provider federation and marketplace-style access, rather than through a fully standardized serverless abstraction layer.

These findings support three analytical conclusions. First, the current Quantum Serverless should be interpreted as a "queue-first" model rather than a true Near Real-time elastic model. Second, practical adoption is currently constrained more by integration and service-design bottlenecks than by the mere existence of quantum backends. Third, short-term progress is likely to come from improvements in hybrid orchestration, standardized service abstractions, and pricing transparency, while long-term progress depends on reducing end-to-end latency and improving production reliability.

7. Conclusion

Quantum Serverless is emerging as an important abstraction for hybrid quantum-classical computing, but our survey shows that its current maturity lies more in orchestration potential than in fully realized cloud-native execution. By consolidating the architectural components of Quantum Serverless and evaluating current platforms through a unified KPI framework, this paper provides a cross-provider view of how the field is progressing from raw quantum hardware access toward managed hybrid services.

The quantitative findings indicate that the Quantum Serverless ecosystem is growing but remains unevenly mature. Among 18 cloud platforms, we identified 51 available quantum resources and 20 quantum hardware providers. Of these platforms, 47.62% offer in-house quantum hardware, 28.57% provide multiple in-house providers, and 23.81% depend entirely on external providers. This distribution points to partial vertical integration, but also to continued reliance on federated and partner-based hardware access models. At the same time, the serverless-readiness KPIs reveal a clear maturity gap: queue support is universal, but only 30% of platforms offer Function-as-a-Service, 40% offer Containers-as-a-Service, and 55% provide pay-as-you-go pricing.

Most notably, none of the analyzed platforms satisfies the Near-RT criterion, and only one provider is currently production-ready according to the Quantum Serverless KPI set.

Our analysis also shows that the current leaders are not simply the providers with access to quantum hardware, but those that combine hardware availability with stronger software ecosystems, integrated APIs and SDKs, and clearer pathways from experimentation to deployment. In particular, IBM Quantum meets all but one of the analyzed Quantum Serverless KPIs, followed by Azure Quantum, Google Quantum AI, and Amazon Braket, each with two unmet KPIs, while all remaining platforms fail to satisfy at least three. This ranking suggests that competitive advantage is shifting away from hardware exposure alone and toward the maturity of orchestration across software, operations, and business-model layers.

Compared with prior work, which has mainly focused on individual platforms, specific frameworks, or proposal-oriented architectures, this paper offers a broader comparative perspective. Rather than describing a single implementation path, it synthesizes heterogeneous evidence into a unified architecture and KPI-based assessment that makes maturity gaps visible across providers. In this sense, the contribution of this survey is not only to catalogue existing solutions, but also to clarify what currently counts as Quantum Serverless and what still prevents it from becoming a practical, production-grade computing model.

The main message of this paper is therefore that today's Quantum Serverless should be interpreted as a queue-first hybrid orchestration model, not yet as a fully elastic serverless computing model in the classical cloud sense. Short-term progress is likely to come from better workflow orchestration, standardized service abstractions, pricing transparency, and stronger integration between quantum and classical execution environments. Longer-term progress depends on reducing end-to-end latency, improving reliability, and moving from experimental access models toward production-ready managed services. If this transition succeeds, Quantum Serverless may play for hybrid quantum computing a role similar to that played by higher-level cloud abstractions in the evolution of classical cloud adoption: transforming complex infrastructure into accessible, programmable, and scalable services.

Answering the research questions. For RQ_1, the paper identifies Quantum Serverless as a layered hybrid architecture centered on the interaction between a Classical Data Center, a Quantum Data Center, and, when applicable, an External Quantum Hardware Provider. Within this architecture, the key operational components are the Runtime Manager, Runtime Job, Primitives, Circuit, Execution Kernel, and Quantum Hardware, together with queue-based coordination and serverless classical execution support. These components collectively define how hybrid quantum–classical workloads are submitted, orchestrated, executed, and returned to the user.

For RQ_2, the KPI-based evaluation shows that quantum and cloud stakeholders are progressing unevenly toward practical Quantum Serverless computing. Basic ingredients such as hardware access, SDKs, simulators, and queue-based execution are already broadly available, but stronger serverless characteristics remain limited. In particular, Function-as-a-Service, Containers-as-a-Service, production readiness, pricing transparency, and Near Real-time responsiveness are still missing in large parts of the ecosystem. This indicates that current stakeholders are advancing from raw backend exposure toward managed hybrid services, but that Quantum Serverless is still at an intermediate, queue-driven stage of maturity rather than a fully elastic cloud-native model in the classical serverless sense.

CRediT authorship contribution statement

Dimitar Mileski: Writing – original draft, Visualization, Validation, Methodology, Investigation, Formal analysis, Data curation, Conceptualization; **Marjan Gusev:** Writing – review & editing, Supervision, Conceptualization; **Sashko Ristov:** Writing – review & editing, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was funded by CETPartnership, the Clean Energy Transition Partnership under the 2024 joint call for research proposals, co-funded by the European Commission (GA N°101069750) and with the funding organizations FFG Austrian Research Promotion Agency (Austria), NWO Dutch Research Council (the Netherlands), Swedish Energy Agency (Sweden) and GSRI (Greece); the Key Digital Technologies (KDT) Joint Undertaking through the project MATISSE, GA No. 101140216; and *Land Tirol, Austria*, under contract F.35499.

Data availability

No data was used for the research described in the article.

References

[1] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar, J.E. Gonzalez, R.A. Popa, I. Stoica, D.A. Patterson, Cloud programming simplified: a Berkeley view on serverless computing, 2019, arXiv:1902.03383 [cs.OS]

[2] B. Johnson, I. Faro, M. Behrendt, J. Gambetta, Introducing quantum serverless, a new programming model for leveraging quantum and classical resources, 2021, Last accessed 12 April 2026, https://research.ibm.com/blog/quantum-serverless-programming.

[3] M.A. Serrano, J.A. Cruz-Lemus, R. Perez-Castillo, M. Piattini, Quantum software components and platforms: overview and quality assessment, ACM Comput. Surv. 55 (8) (2022). https://doi.org/10.1145/3548679

[4] P. Singh, R. Dasgupta, A. Singh, H. Pandey, V. Hassija, V. Chamola, B. Sikdar, A survey on available tools and technologies enabling quantum computing, IEEE Access 12 (2024) 57974–57991. https://doi.org/10.1109/ACCESS.2024.3388005

[5] P.B. Upama, M.J.H. Faruk, M. Nazim, M. Masum, H. Shahriar, G. Uddin, S. Barzanjeh, S.I. Ahamed, A. Rahman, Evolution of quantum computing: a systematic survey on the use of quantum computing tools, in: 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC), 2022, pp. 520–529. https://doi.org/10.1109/COMPSAC54236.2022.00096

[6] J. Faj, I. Peng, J. Wahlgren, S. Markidis, Quantum computer simulations at warp speed: assessing the impact of GPU acceleration: a case study with IBM qiskit aer, nvidia thrust & cuquantum, in: 2023 IEEE 19th International Conference on e-Science (e-Science), IEEE Computer Society, Los Alamitos, CA, USA, 2023, pp. 1–10. https://doi.org/10.1109/e-Science58273.2023.10254803

[7] S. Ristov, S. Pedratscher, T. Fahringer, AFCL: An abstract function choreography language for serverless workflow specification, Fut. Gen. Comp. Syst. 114 (2021) 368–382.

[8] P. Gritsch, M. Cotrotzo, S. Ristov, Scale interoperable composite backend services with AFCL workflows in serverless sky computing, IEEE Trans. Netw. Serv. Manag. 22 (5) (2025) 4762–4774. https://doi.org/10.1109/TNSM.2025.3592700

[9] S. Ristov, S. Pedratscher, T. Fahringer, XAFCL: run scalable function choreographies across multiple faas systems, IEEE Trans. Serv. Comput. 16 (1) (2023) 711–723. https://doi.org/10.1109/TSC.2021.3128137

[10] S. Ristov, S. Brandacher, M. Hautz, M. Felderer, R. Breu, CODE: Code once, deploy everywhere serverless functions in federated FaaS, Future Gener. Comput. Syst. 160 (2024) 442–456. https://doi.org/10.1016/j.future.2024.06.017

[11] Qiskit: An Open-source Framework for Quantum Computing, 2021, https://doi.org/10.5281/zenodo.2573505

[12] Qiskit runtime overview, 2022, Last accessed 12 April 2026, https://cloud.ibm.com/docs/quantum-computing?topic=quantum-computing-overview.

[13] IBM, IBM Quantum Docs, 2025, Last accessed 12 April 2026, https://docs.quantum.ibm.com/.

[14] K. Ferris, S. Panda, R. Davis, Qiskit Serverless sets the stage for Qiskit Functions | IBM Quantum Computing Blog — ibm.com, 2024, [Last accessed 12 April 2026]. https://www.ibm.com/quantum/blog/qiskit-serverless.

[15] H.T. Nguyen, M. Usman, R. Buyya, QFaaS: a serverless function-as-a-Service framework for quantum computing, (2022). arXiv preprint arXiv:2205.14845

[16] M. Grossi, L. Crippa, A. Aita, G. Bartoli, V. Sammarco, E. Picca, N. Said, F. Tramonto, F. Mattei, A serverless cloud integration for quantum computing, (2021). arXiv preprint arXiv:2107.02007

[17] C. Cicconetti, Modeling and performance evaluation of hybrid classical-quantum serverless computing platforms, IEEE Trans. Quantum Eng. (2025).

[18] T. Li, Z. Zhao, Moirai: optimizing quantum serverless function orchestration via device allocation and circuit deployment, in: 2024 IEEE International Conference on Web Services (ICWS), IEEE, 2024, pp. 707–717.

[19] S.S. Gill, Quantum and blockchain based serverless edge computing: a vision, model, new trends and future directions, Internet Technol. Lett. (2021) e275.

[20] International year of quantum science and technology — quantum2025.org, [Last accessed 12 April 2026]. https://quantum2025.org/en/.

[21] The European High Performance Computing Joint Undertaking EuroHPC - Hybrid HPC-QC from a policy maker perspective, [Last accessed 12 April 2026]. https://teratec.eu/media/wp-content/uploads/2024/06/2nd_TQCI_Reims_2024_Chatwell.pdf.

[22] L. Schulz, Quantum-HPC Benchmarking: The Arrival, Applicability and Assessment of Quantum to HPC, [Last accessed 12 April 2026]. https://teratec.eu/media/wp-content/uploads/2024/06/Laura_Schulz_2024-06-04-LS-Reims-TQCI-HPCQC-Benchmarks.pdf.

[23] B. Cheng, X.-H. Deng, X. Gu, Y. He, G. Hu, P. Huang, J. Li, B.-C. Lin, D. Lu, Y. Lu, et al., Noisy intermediate-scale quantum computers, Front. Phys. 18 (2) (2023) 21308.

[24] Quantum Computers, Coming to a Datacenter Near You — spectrum.ieee.org, [Last accessed 12 April 2026]. https://www.facebook.com/48576411181, https://spectrum.ieee.org/quantum-data-center.

[25] Quantum computing has a hype problem — technologyreview.com, [Last accessed 12 April 2026]. https://www.technologyreview.com/2022/03/28/1048355/quantum-computing-has-a-hype-problem/.

[26] Suppressing quantum errors by scaling a surface code logical qubit, Nature 614 (7949) (2023) 676–681.

[27] R. Acharya, L. Aghababaie-Beni, I. Aleiner, T.I. Andersen, M. Ansmann, F. Arute, K. Arya, A. Asfaw, N. Astrakhantsev, J. Atalaya, et al., Quantum error correction below the surface code threshold, Nature (2024). https://doi.org/10.1038/s41586-024-08449-y

[28] F. Arute, K. Arya, R. Babbush, D. Bacon, J.C. Bardin, R. Barends, R. Biswas, S. Boixo, F.G. Brandao, D.A. Buell, et al., Quantum supremacy using a programmable superconducting processor, Nature 574 (7779) (2019) 505–510.

[29] D. Owens, R. El Khatib, M. Bisheh-Niasar, R. Azarderakhsh, M.M. Kermani, Efficient and side-channel resistant ed25519 on ARM cortex-M4, IEEE Trans. Circuits Syst. I Regul. Pap. 71 (6) (2024) 2674–2686.

[30] M.B. Niasar, R. Azarderakhsh, M.M. Kermani, Optimized architectures for elliptic curve cryptography over curve448, Cryptol. ePrint Arch. (2020).

[31] M. Anastasova, R. Azarderakhsh, M.M. Kermani, Fast strategies for the implementation of SIKE round 3 on ARM Cortex-M4, IEEE Trans. Circuits Syst. I: Regul. Pap. 68 (10) (2021) 4129–4141.

[32] A. Cintas-Canto, M.M. Kermani, R. Azarderakhsh, Error detection constructions for ITA finite field inversions over $GF(2^m)$ on FPGA using CRC and Hamming codes, IEEE Trans. Reliab. 72 (2) (2022) 651–661.

[33] M.M. Kermani, R. Azarderakhsh, M. Mirakhorli, Education and research integration of emerging multidisciplinary medical devices security (2016).

[34] S. Subramanian, M. Mozaffari-Kermani, R. Azarderakhsh, M. Nojoumian, Reliable hardware architectures for cryptographic block ciphers LED and HIGHT, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 36 (10) (2017) 1750–1763. https://doi.org/10.1109/TCAD.2017.2661811

[35] M. Mozaffari-Kermani, R. Azarderakhsh, M. Bisheh-Niasar, Cryptographic engineering: a fast and efficient SIKE implementation in FPGA, IEEE Trans. Circuits Syst. I Regul. Pap. 71 (2) (2024) 760–773. https://doi.org/10.1109/TCSI.2023.3337374

[36] M. Bisheh-Niasar, R. Azarderakhsh, M. Mozaffari-Kermani, High-Speed NTT-Based Polynomial Multiplication Accelerator for CRYSTALS-Kyber Post-Quantum Cryptography, 2021, (IACR Cryptology ePrint Archive). https://eprint.iacr.org/2021/563.

[37] A. Aghaie, M. Mozaffari-Kermani, R. Azarderakhsh, Design-for-error-detection in implementations of cryptographic nonlinear substitution boxes benchmarked on ASIC, in: IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS), 2018, pp. 574–577. https://doi.org/10.1109/MWSCAS.2018.8624008

[38] M. Mozaffari-Kermani, A. Reyhani-Masoleh, A low-Cost S-Box for the advanced encryption standard using normal basis, in: IEEE International Conference on Electro/Information Technology, 2009, pp. 52–55. https://doi.org/10.1109/EIT.2009.5189583

[39] M. Bisheh-Niasar, R. Azarderakhsh, M. Mozaffari-Kermani, Cryptographic accelerators for digital signature based on ed25519, IEEE Trans. Very Large Scale Integr. Syst. 29 (8) (2021) 1402–1414. https://doi.org/10.1109/TVLSI.2021.3077885

[40] M. Mozaffari-Kermani, S. Sur-Kolay, A. Raghunathan, N.K. Jha, Systematic poisoning attacks on and defenses for machine learning in healthcare, IEEE J. Biomed. Health Inform. 19 (6) (2015) 1893–1905. https://doi.org/10.1109/JBHI.2014.2344095

[41] M. Hutter, et al., EdDSA for hardware implementation: highly optimized Ed25519 and Ed448 signatures, IEEE Trans. Circuits Syst. II Express Br. (2020).

[42] B. Koziel, R. Azarderakhsh, M. Mozaffari-Kermani, Fast hardware architectures for supersingular isogeny Diffie-Hellman key exchange on FPGA, in: Progress in Cryptology – INDOCRYPT 2016, Springer, 2016, pp. 191–206. https://doi.org/10.1007/978-3-319-49890-4_11

[43] M. Mozaffari-Kermani, R. Azarderakhsh, S. Bayat-Sarmadi, Reliable hardware architectures for efficient secure hash functions ECHO and fugue, in: ACM International Conference on Computing Frontiers (CF), 2018, pp. 204–207.

[44] B. Koziel, A. Jalali, R. Azarderakhsh, M.M. Kermani, D. Jao, NEON-SIDH: Efficient Implementation of Supersingular Isogeny Diffie-Hellman Key-Exchange Protocol on ARM, 2016, (Cryptology ePrint Archive, Paper 2016/669). https://eprint.iacr.org/2016/669.

[45] B. Taghavi, R. Azarderakhsh, M. Mozaffari Kermani, ParallelNTT: maximizing performance of forward and inverse NTT on FPGA for ML-DSA and ML-KEM, in: Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI) 2025, 2025, pp. 372–378.

[46] R. Karam, S. Katkoori, M. Mozaffari-Kermani, Engaged student learning with gamified labs: a new approach for hardware security education, in: Proceedings of the 2023 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE), IEEE, 2023, pp. 1–4.

[47] Welcome to Quantum Cloud Services, Last accessed 12 April 2026, https://docs.rigetti.com/qcs/.

[48] Atos, 2022, Last accessed 12 April 2026, https://atos.net/en/.

[49] Baidu, Last accessed 12 April 2026, https://research.baidu.com/Research_Areas/index-view?id=75.

[50] IONQ, Last accessed 12 April 2026, https://ionq.com/.

[51] Wave systems: the practical quantum computing company, Last accessed 12 April 2026, https://www.dwavesys.com/.

[52] Xanadu, Last accessed 12 April 2026, https://www.xanadu.ai/.

[53] H. Silvério, S. Grijalva, C. Dalyac, L. Leclerc, P.J. Karalekas, N. Shammah, M. Beji, L.-P. Henry, L. Henriet, Pulser: an open-source package for the design of pulse sequences in programmable neutral-atom arrays, Quantum 6 (2022) 629.

[54] IBM, IBM Quantum, 2025, Last accessed 12 April 2026, https://quantum-computing.ibm.com/.

[55] Bradben, Azure Quantum Documentation, Last accessed 12 April 2026, https://learn.microsoft.com/en-us/azure/quantum/.

[56] Google quantum ai, Last accessed 12 April 2026, https://quantumai.google/.

[57] Intel Quantum computing, Last accessed 12 April 2026, https://www.intel.com/content/www/us/en/research/quantum-computing.{html}.

[58] Honeywell Quantum, Last accessed 12 April 2026, https://www.honeywell.com/us/en/company/quantum.

[59] Quantinuum, Last accessed 12 April 2026, https://www.quantinuum.com/.

[60] Quantum Circuits Inc, Last accessed 12 April 2026, https://quantumcircuits.com/.

[61] 1Qbit, 2022, Last accessed 12 April 2026, https://1qbit.com/.

[62] Toshiba Quantum Information Group, Last accessed 12 April 2026, https://www.toshiba.eu/pages/eu/Cambridge-Research-Laboratory/quantum-information.

[63] H. Ball, M.J. Biercuk, A.R.R. Carvalho, J. Chen, M. Hush, L.A. De Castro, L. Li, P.J. Liebermann, H.J. Slatyer, C. Edmunds, V. Frey, C. Hempel, A. Milne, Software tools for quantum control: improving quantum computer performance through noise and error suppression, Quantum Sci. Technol. 6 (4) (2021) 044011. https://doi.org/10.1088/2058-9565/abdca6

[64] Strangeworks, Strangeworks Quantum Computing Platform, Last accessed 12 April 2026, https://strangeworks.com/platform.

[65] IBM Cloud, Last accessed 12 April 2026, https://www.ibm.com/cloud.

[66] Microsoft Azure, Last accessed 12 April 2026, https://azure.microsoft.com/en-us/.

[67] Google Cloud, Last accessed 12 April 2026, https://cloud.google.com/docs.

[68] AWS Amazon Web Services, Last accessed 12 April 2026, https://aws.amazon.com/.

[69] Alibaba cloud, Last accessed 12 April 2026, https://eu.alibabacloud.com/en.

[70] J. Sampé, P. Garcia-Lopez, M. Sanchez-Artigas, G. Vernik, P. Roca-Llaberia, A. Arjona, Toward multicloud access transparency in serverless computing, IEEE Softw. 38 (1) (2020) 68–74.

[71] D. Mileski, M. Gusev, Serverless FaaS scalability evaluation: an ECG signal processing use case, in: 2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO), IEEE, 2022, pp. 853–858.

[72] IBM Cloud Code Engine Docs, Last accessed 12 April 2026, https://cloud.ibm.com/docs/codeengine?topic=codeengine-getting-started.

[73] Azure container apps, Last accessed 12 April 2026, https://azure.microsoft.com/en-us/products/container-apps/.

[74] Cloud run: Container to production in seconds, Last accessed 12 April 2026, https://cloud.google.com/run.

[75] B. Denise, S. Enggist, AWS Fargate, Last accessed 12 April 2026, https://aws.amazon.com/fargate/.

[76] ECS, 2013, Last accessed 12 April 2026, https://aws.amazon.com/ecs/.

[77] Elastic container instance (ECI) - alibaba cloud, Last accessed 12 April 2026, https://www.alibabacloud.com/product/elastic-container-instance.

[78] J. Nielsen, Response Times: The 3 Important Limits, 1993, Last accessed 12 April 2026, https://www.nngroup.com/articles/response-times-3-important-limits/.

[79] X. Geoffret, Benchmarking at IQM, 2024, [Last accessed 12 April 2026]. https://teratec.eu/media/wp-content/uploads/2024/06/Xavier_Geoffret.pdf.

[80] Amazon Web Services, Amazon Braket, 2020, [Last accessed 12 April 2026]. https://aws.amazon.com/braket/.