ACADEMIA QUANTUM

Technical Note
Published: 2025-04-14
https://doi.org/10.20935/AcadQuant7627

Toward an ecosystem-agnostic standard for quantum runtime architecture

Markiian Tsymbalista[1,*], Ihor Katernyak[1]

Academic Editors: Randy Kuang, Yuhua Duan

Abstract

Research on achieving Quantum Utility (QU) with software approaches covers all layers of Quantum Computing Optimization Middleware (QCOM) and requires execution on real quantum hardware (QH). Due to the nascent nature of the technology domain and the proprietary strategies of both large and small players, popular runtimes for executing quantum workloads lack flexibility in programming models, scheduling, and hardware access patterns, including queuing, which creates roadblocks for researchers and slows innovation. These problems are further exacerbated by emerging hybrid operating models that place Graphical Processing Unit (GPU) supercomputing and Quantum Intermediate Representation (QIR) at the heart of real-time computations across quantum and distributed resources. There is a need for a widely adopted runtime platform (RP) driven by the open-source community that can be deployed to work in a distributed manner between Quantum Processing Units (QPUs), GPUs, control hardware, and external computational resources and provide required flexibility in terms of programming and configuration models. The product discovery approach of the Value Proposition Canvas (VPC) and software architecture techniques including Architecture Drivers (ADs), Quality Attributes (QAs), Attribute-Driven Design (ADD) were employed to design a blueprint architecture. The intent of the blueprint is to move beyond the concept of just software that runs quantum workloads and focus on practical challenges of existing runtimes that block efforts of QC optimization research. Once embraced by the community, it will open doors to the evaluation of new algorithmic strategies not widely used as of now.

Keywords: *quantum computing, quantum utility, quantum algorithm performance, quantum computing optimization middleware (QCOM), quantum runtime architecture, quantum runtime platform (RP), open quantum, quantum reference architecture, Quantum Computing Platform as a Service (QCPaaS)*

Citation: Tsymbalista M, Katernyak I. Toward an ecosystem-agnostic standard for quantum runtime architecture. *Academia Quantum* 2025;2. https://doi.org/10.20935/AcadQuant7627

1. Introduction

Quantum Computing Optimization Middleware (QCOM) [1], as complex optimization software layer, will continue its evolution driven by the efforts of a significant number of researchers. For effective computation optimization techniques, remote execution capability is mandatory. There are several obstacles to that, mostly driven by the fact that access to QH is scarce and is mostly exposed through proxy companies building their own runtimes which usually implies that there are a lot of limitations in how APIs could be used. They are tuned for the development of widely known quantum algorithms and lack flexibility, and they hide implementation details under the hood. By performing this, they are reserving their part of the pie. Promising research requires uncertainty and black box points to be removed to be able to find gaps in the value stream and prepare proper compensation strategies. All software requires its builders to understand its architecture, core qualities, and limitations. So, having closed ecosystems on the path to QU creates a significant bottleneck. From another perspective, the open-source community for quantum software is growing fast. The Open Quantum Hardware [2] initiative aims to put all open-source quantum tech under an umbrella. However, there is no mature project for a Runtime Platform (RP) which could be adjusted to the personal needs of researchers and deployed close to QH where maximum flexibility could be leveraged to move things forward. Smaller industry players spend a lot of time reinventing the wheel by implementing similar runtime capabilities. It is important to build value on top of this and move forward which unfortunately does not happen to quantum RPs in the present day.

There is a need for a quantum runtime reference architecture that will close common challenges for researchers on the path to QU. It should be further implemented in working software, which is outside the scope of this paper. Adoption and development by the community will play a significant role, while it will be open-source in its nature. Open-source quantum RPs should be significant milestones driving ecosystems forward.

2. Methodology

Upon the implementation of experiments as part of the QCOM scope, the authors have encountered a list of challenges. These

[1]Faculty of Electronics and Computer Technologies, Ivan Franko National University of Lviv, Lviv 79000, Ukraine.
*email: markiian.tsymbalista@lnu.edu.ua

scenarios require system capabilities that are not widely implemented in existing tools and RPs. This has been observed as part of the technology consulting effort for one of the commercial companies working in quantum optimization research.

To refine product context, the Value Proposition Canvas (VPC) [3] was leveraged. The VPC is a strategic tool used to ensure that a product aligns well with user needs. As outlined in **Figure 1**, it consists of two parts: describing user/customer needs on the right and how a product fits those needs on the left. The Customer Jobs or jobs to be performed section describes what the user is trying to achieve. Pains—what frustrates or limits them. Gains—what the user values. On the other side, the Products and Services section briefly describes what is offered. Pain Relievers—how user pains are reduced. Gain Creators—how the product enhances user gains.

Further, by using system context elaboration techniques, use cases and constraints were identified which served as an input to the software analyses and design effort. The software architecture (SA) methodology and its methods, including Quality Attribute Scenarios (non-functional requirements) [4], Attribute-Driven Design (ADD) [5], and the Architecture Tradeoff Analyses Method (ATAM) [6], were leveraged to think of, design, and evaluate a blueprint of a runtime reference architecture to cover required system capabilities. ADD is a widely used approach to designing software systems that allows advancing the implementation phase based on the Architectural Drivers (ADs) (requirements that have a profound impact on the SA of a system), Design Decisions (DDs) (architectural choices that are hard or very costly to change after initial system implementation), and other artifacts produced as a result of the ADD cycle.

As part of this study, we combine the VPC and ADD methodologies by tying ADs to the VPC (the Value proposition summarized Section, e.g., AD-1, AD-2) and describe DDs that show how ADs are addressed (in the Conceptual architecture vision for QCPaaS Section, e.g., DD-1, DD-2). This is what is basically the core of every software architecture effort: identifying and addressing the main functional and non-functional requirements of a system.

This paper also utilizes methods from qualitative research to analyze past work that has been proposed on quantum cloud platforms and runtimes.

2.1. Value proposition summarized

Products and services:

- Software RPs that allow the exposure of access to QH, mirroring the core functionality of leading cloud quantum services (not including QH itself) and exposing additional features.
- A Client Software Development Kit (SDK) that is leveraged by software clients to interact with remote runtime.
- Programming models based on adapter design to streamline common quantum performance optimization challenges (AD-1).

Jobs to be performed:

- Quantum Machine Learning (QML) execution. Efficiently run ansatz optimization loops (schedule a batch of independent jobs) (AD-2).
- Compilation. Optimize compilation flows for circuit parts that require frequent recompilation, as an example (AD-3).
- Noise mitigation. Efficiently run pre- and post-processing error mitigation (EM) schemes that require multiple modified quantum circuits (ZNE, PEC, tomography-based methods, etc.) (AD-4).
- Measurement processing, e.g., running advanced EM post-processing routines involving DNNs or tensor networks (AD-5).
- Combine QML + EM use cases in different combinations (AD-6).

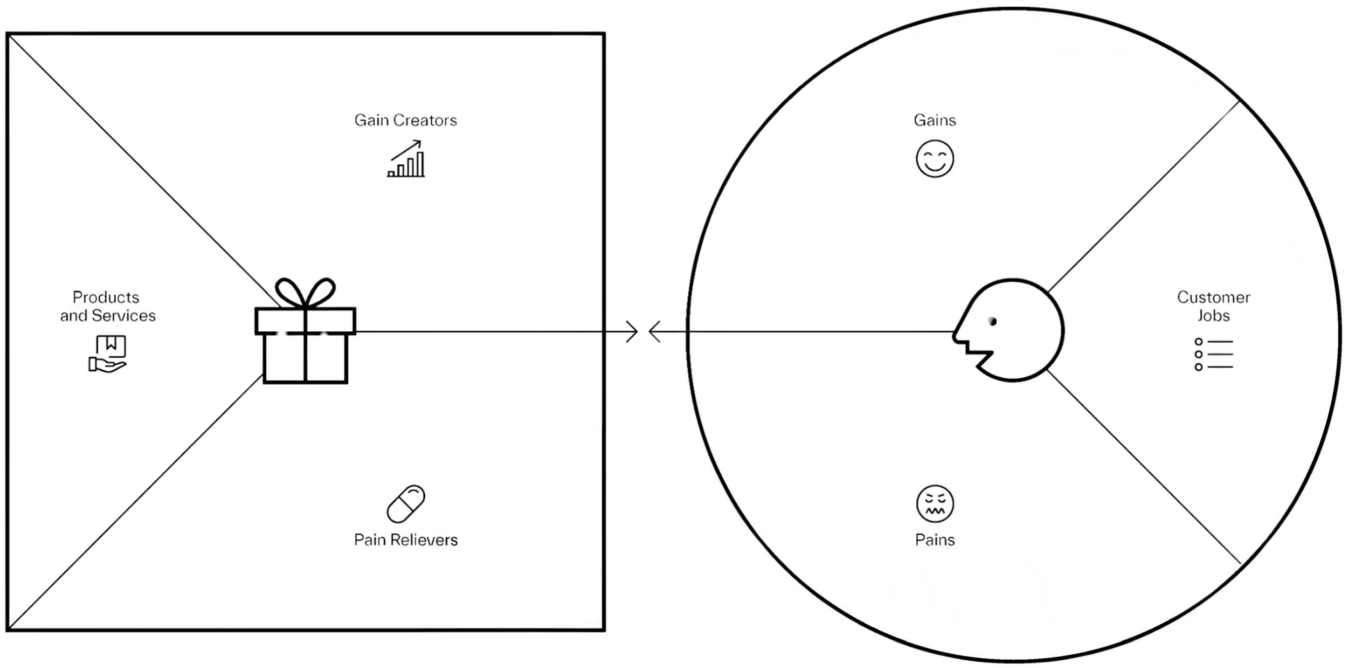

Figure 1 • Value Proposition Canvas structure [3].

- Have a baseline solution to analyze and implement support for complex distributed use cases. An example could be to schedule a circuit, part of which is going to be run on the simulator (maybe even specific GPU hardware) with another part on real QH. Further, this use case could be split into two modes: with mid-circuit measurements and without (AD-7).
- Batch jobs that include different circuits with an ability to choose a specific EM method for each (AD-8).
- Deploy a custom library on the runtime for gate calibration, EM, or other accuracy optimization tasks, so it could be used during computations (AD-9).
- Schedule an arbitrary circuit implemented with common frameworks, e.g., Qiskit, Cirq, and PennyLane, for execution on QH (AD-10).

Gains that researchers look for on a platform:

- A need to put circuit execution closer to QH (AD-11).
- A minimal impact of RPs on results' accuracy (AD-12).
- Code reuse for tasks like EM and QML (AD-13).
- Plugin resource estimation capability (AD-14).
- Quick deployment on arbitrary computational infrastructure. The full infrastructure automation of the platform, such as with the Infrastructure-as-Code (IaC) approach (AD-15).
- A familiar tech stack (AD-16).
- The usability of software interfaces (AD-17).
- Cost savings (AD-18).
- Flexibility in the programming model that allows the full customization of the execution pipeline (AD-18).

Pains that researchers have:

- For niche quantum hardware manufacturers and research institutions, there is a necessity to build runtime/cloud quantum platforms from scratch to expose their QH.
- Long execution time for circuits. QML use cases spawn thousands of calls to QH when running optimization loops for ansatz. QH could be located on another continent. Because of this, network operations could significantly increase waiting time for algorithm execution, making researchers idle like in the old days of mainframe machines (AD-19).
- The low availability of QH for running workloads (mechanisms for queue management and resource allocation do not provide the required optimal level of determinism) (AD-20).
- Dependency on vendor SDKs (with Qiskit runtime [7] dependency on the Qiskit SDK) and their limited customization, e.g., of compilation steps, hardware access approaches, and opinionated runtime primitives (with some extensibility points though) (AD-21).
- No optimal structure of runtime pricing models from different vendors (AD-22).
- The black box architecture and implementation details of most quantum platforms.
- A lack of customization of RPs (AD-23).

Gain creators:

- Open-source solutions that could be deployed to commodity computational infrastructure either publicly or privately.
- Co-location deployment models that reduce impact on results' accuracy.
- Programming models that include primitives required to solve common quantum tasks.
- An ability to plug in resource estimation modules.
- Implementation using well-known tooling, e.g., Python, Docker, and Kubernetes.
- Implementing SDKs adhering to object-oriented design practices.
- Cost savings by using open-source platforms that do not require licensing costs.
- The flexibility of programming models to construct and run complex custom pipelines.

Pain relievers:

- Efficiency when running workloads that require thousands of circuit executions as part of one algorithm.
- Queue management and resource allocation modules that are opened for extensibility.
- An ability to use arbitrary quantum circuit frameworks.
- No dependency on pricing policies for bridge services, e.g., AWS and Azure, due to the open-source nature.
- Documented architectures with open-sourced codebases.
- Flexibility to substitute or customize every component that constitutes a platform.
- For small quantum hardware fabs, an ability to leverage open-source, ready-to-use solutions to expose access to their hardware and start earning revenue.

2.2. Quick comparison with widely known and used RPs

The scope of this paper is not to extensively explore the details and comparisons of different runtimes but to draw a picture of what they do not cumulatively allow. These perspectives are outlined in the previous section on the VPC. Below are the main roadblocks of some of the RPs that are critical for the "jobs to be performed" of the VPC:

- Qiskit runtime [7]. A lack of flexibility in the programming model, complex extensibility, dependance on the IBM ecosystem, and no details on the internal design.

- Qiskit Dell runtime [8]. Dependent on many outdated libraries, not actively supported, dependent on Qiskit, error-prone, and makes it hard to maintain the programming model.
- Strangerworks Qiskit runtime [9]. It builds abstraction, extending Qiskit runtime primitives, which allows access to different backends (from different providers) in the Qiskit programming model. There is a lack of flexibility to use other SDKs and it is a proprietary technology.
- Pennylane Catalyst [10]. Part of the product contains a runtime. It is tied to the Pennylane ecosystem and framework. The runtime implements some logic around device management which is not explicit as the result is not extensible.
- Intel runtime [11]. A runtime is part of the SDK. There is not much visibility regarding how it is built under the hood. What is obvious is that C++ oriented.
- Multiple papers, e.g., [12, 13], on RPs address some related challenges, but they do not cover value proposition, outlined above.

2.3. Conceptual architecture vision for QCPaaS

The core blueprint of the Quantum Computing Platform as a Service (QCPaaS) architecture is outlined in **Figure 2**. This paragraph describes the architecture using the concept of DD, which is part of the ADD [5] architecture methodology, in a narrative style.

QCPaaS is assumed to be built around the microservice reference architecture (DD-1), heavily reliant on Docker as a containerization technology and on Kubernetes as a cluster orchestration platform (DD-2). The platform itself is scripted with the IaC approach using the Terraform [14] software tool (DD-3) which allows the deployment of the platform in either private or public data centers with a couple of clicks and with minimal manual configuration. The platform's concept of workers implementing the Bring-Your-Own-Container (BYOC) pattern allows the extension of the core image of the worker to include any custom library, e.g., Mitiq [15] (DD-4). It means that in a working condition, several workers with required library images are deployed to the cluster and are used by end-users of the QCPaaS by selecting an option in the SDK call. When a worker is deployed, it should auto-register itself with the runtime (DD-13). No explicit AD about Continuous Integration/Continuous Deployment (CI/CD) server is used but BYOC is deployed to the cluster by using the pipeline job of the CI/CD server by triggering it manually or using the platform's User Interface (UI) that could be additionally developed (DD-5). The usability of QCPaaS software interfaces is achieved by the implementation of a client-side SDK (DD-6). Independence from a particular programming framework is partially driven by the BYOC mechanism described above, when a framework library, e.g., Qiskit, is deployed as a container in a cluster. Another part of the solution includes logic that discovers required libraries from the client code and makes sure that the execution container has them pre-deployed (DD-7). The implementation of the Workflow Manager (DD-8) is central to the architecture as this component is primarily responsible for allocating resources and both the efficient

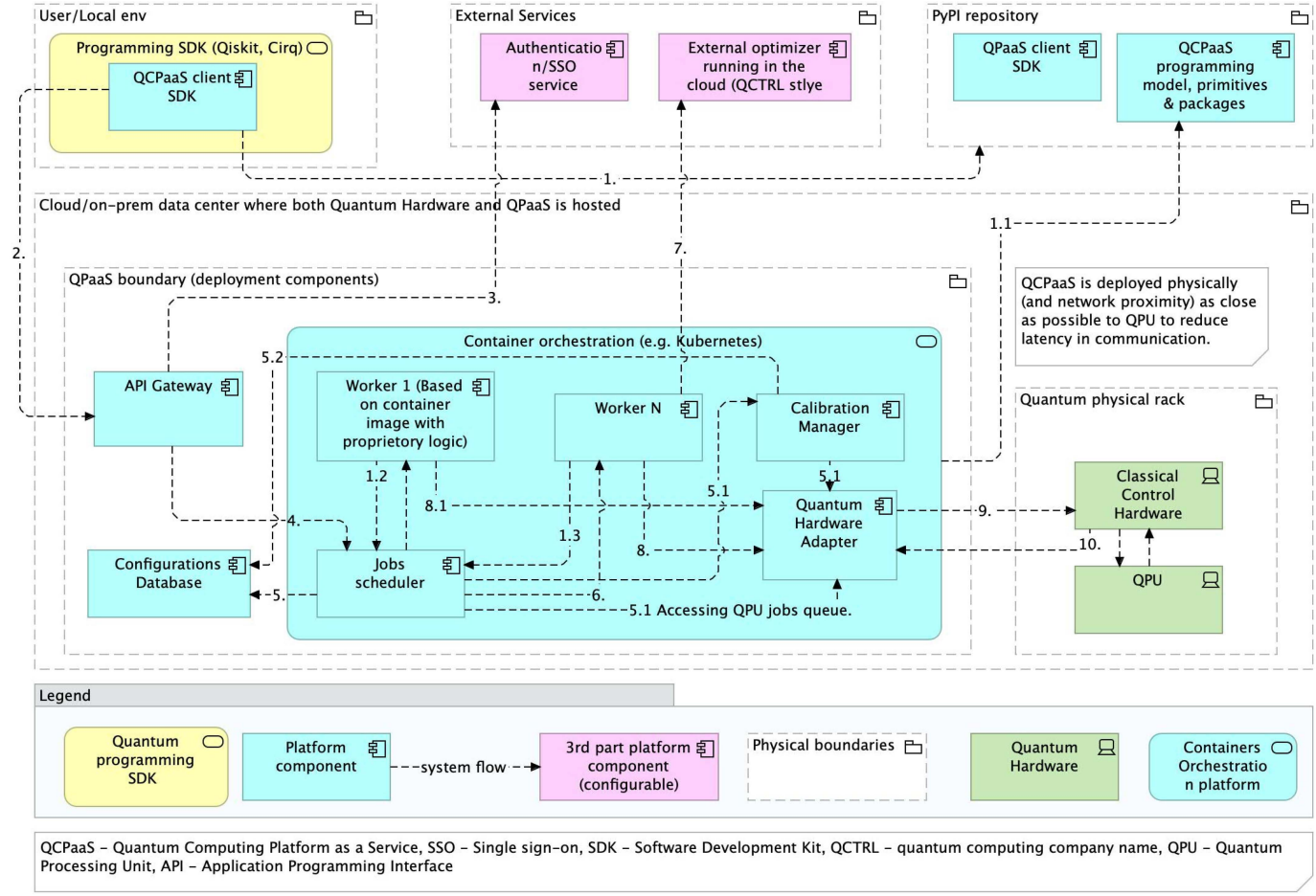

Figure 2 • Conceptual architecture of QCPaaS.

and effective scheduling of workloads. To achieve the maximal accuracy of results QCPaaS, should be physically deployed as close as possible to real hardware (DD-15). A description of the main components and their relationships is defined below.

1. The user installs the SDK for communication with the QCPaaS (contains programming model).
    1.1. Containers running in the cluster access QCPaaS libraries if needed.
    1.2. Worker 1 registers itself with the scheduler, sharing details about computational capability.
    1.3. Worker N registers itself with the scheduler.
2. The user creates quantum and classical program using the framework of choice and schedules it for execution on the QCPaaS (leveraging platform SDK).
3. The client is authenticated with an external service. This is a configurable element, the particular implementation of which could be substituted. There is also no explicit choice regarding API gateway implementation.
4. The Workflow Manager component receives execution payload.
5. The Workflow Manager accesses the configuration database to obtain data about the local job queue, available workers and their properties, etc.
    5.1. The Workflow Manager, via the Calibration Manager, retrieves the latest calibration data from the QPU, such as coherence times, gate error rates, and qubit frequencies, by accessing the Quantum Hardware Adapter. This could be realized either on demand or periodically.
    5.2. The latest calibration data with timestamps are stored in the configuration database.
6. A hybrid job is scheduled for execution with an available worker.
7. An external service is accessed for optimization or other logic if needed.
8. Hardware is accessed via adapter software.
    8.1. The hardware adapter is accessed as part of a separate use case using a different worker.
9. The hardware adapter accesses the QPU control stack via the native API.
10. Results are returned to the adapter from the QPU.

2.3.1. Programming model explained (DD-10)

One of the core drivers for QCPaaS development is the need for a flexible programming model that allows pre- and post-processing steps to be conducted for different parts of algorithm execution, the customization of pipeline of quantum circuit transpilation, and optimization with arbitrary custom routines or products from third-party vendors to be conducted. Considering that the development approach in the ecosystem is the classical Object-Oriented Programming (OOP) paradigm, the decision was made to use a classical design pattern from Object-Oriented Design (OOD), the name of which is Decorator [16].

It is a structural design pattern that allows attaching new behaviors to objects by placing these objects inside special wrapper objects that contain the behaviors.

Part of the QCPaaS client SDK is the RuntimeWorkerBase class, outlined in **Figure 3**. It is a base class that all custom user workloads should inherit from. It provides the generic run() method that the runtime looks for to execute a scheduled workload. QML-Worker is an example of an implementation which contains an example of an hybrid algorithm that is planned to be executed on the runtime to increase efficiency. It overrides the run() method and sets backend_name, e.g., "FakeKolkata", and executionoptions list, e.g., ['ErrorMitigatedExecutionBackend', 'option2'], which is a pipeline (could be many items) for custom optimization logic that is present on the runtime and which is discovered as part of a separate process, defined in 1.2 flow description, of conceptual architecture vision description. RuntimeProvider (**Figure 3**) is used by the client to schedule workload on the remote runtime. The module name with logic inherited from RuntimeWorkerBase is selected as a parameter (in our case qml_worker). Under the hood, it converts the Python class to a string and discovers all required modules that are required to run the workload. These data are sent to the runtime via an http call. On the runtime side, the Workflow Manager (**Figure 3**) complements this programming model by looking for a worker that could satisfy workload needs in required packages and custom pipeline optimization modules.

Decorator's paradigm fits very nicely with the logic of assigning pre- and post-processing steps for parts of the algorithm execution in a reusable way. This could be achieved by developing custom pipeline classes, e.g., ErrorMitigatedExecutionBackend, and assigning the correct order of their execution (pre-, post-) in the execution_options list which comes as part of the programming model. Reusability is achieved by deploying specific container images with the logic on the runtime using the BYOC model which could be reused by different clients. In **Figure 3**, EnrichedExecutionBackend both aggregates ExecutionBackend and inherits from it. This is a core of the Decorator pattern, which allows the attachment of custom functionality (ErrorMitigatedExecutionBackend) on the fly to different backend implementations, e.g QiskitBackend. The pattern itself does not handle the full resolution and substitution of specific modules under the hood on the runtime. This is achieved by traversing all the descendants of EnrichedExecutionBackend which are available in the context of runtime execution and setting the correct order according to the execution_options list.

2.3.2. Hardware adapter design

This part of software usually goes with programming SDKs like Qiskit or Cirq. Qiskit has adapter architectures implemented via providers [17]. With the release of BackendV2, the IBM team removed a lot of drawbacks and inflexibilities from the previous version, e.g., hard-to-discover backend features due to lack of native data structures and the handling of timing constraints. These limitations were removed in version 2 [18] along with new features being added like customizable compilation and a comprehensive view of backends by representing multiqubit instructions. Of course, being chosen and adjusted for IBM product lines, this part of the Qiskit package could still be used as a starting point

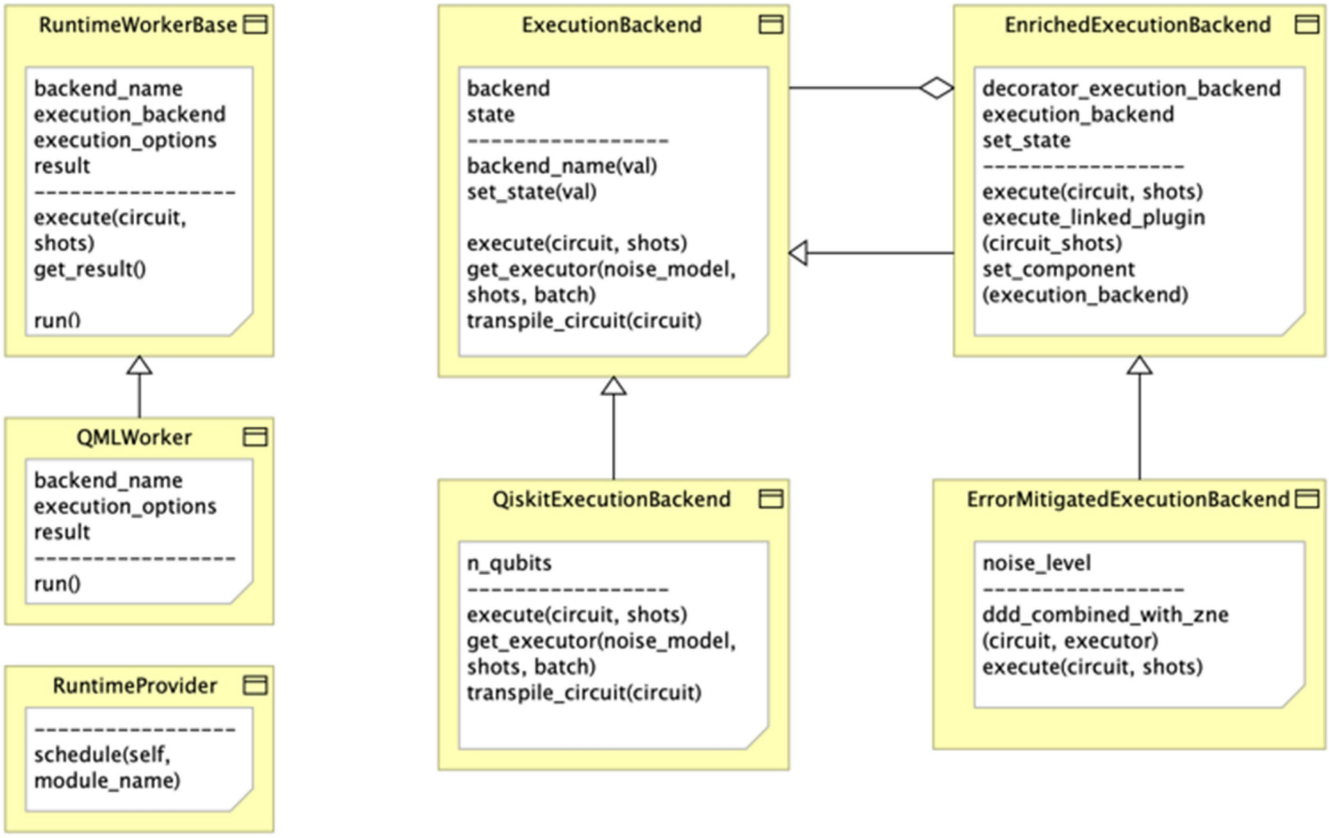

Figure 3 • A classes diagram for a core part of the programming model.

when building new proxies for open hardware QPU backends. Abstraction should be created using Qiskit backend implementation, so substitutes could be plugged in (DD-14) in case of licensing changes or computational optimization use-case pop-ups that stop optimization objectives from being reached (there were a couple in the previous version, e.g., problems with mapping backend operations and compilation customizability). Because of Qiskit's widespread use, there are providers for main quantum hardware manufacturers, e.g., IQM [19].

2.3.3. Scheduling and availability mechanism

The scheduling and availability mechanism works as a complementary part of the Workflow Manager (**Figure 2**) and allows the scheduling of workloads for execution in the most efficient way. For utility, it requires careful consideration of the unique characteristics of quantum computing (QC), such as qubit coherence times, error rates, and the hybrid nature of quantum–classical computation. This paper does not go in details of the optimization of these properties but tries to define important software modules that are required in QCPaaS as a baseline for optimization effort (DD-12):

a. Resource estimation. Based on it availability time is planned, as it allows users to see how long algorithm execution could take. Microsoft built an open-source version [20] which they use as part of their Azure quantum offering [21] and which could be incorporated in the implementation of this reference architecture. As explained in [22], it is quite a nontrivial task to equally distribute work across quantum machines. Their characteristics varies after day-to-day calibrations. Thus, different time-sensitive properties, e.g., fidelity, should be taken into consideration during scheduling.

b. Queue management. This involves the prioritization and scheduling of jobs based on various factors (size, estimated duration, user priority, SLAs, etc.). In case a batch job is scheduled (execution of 1000 and more iterations of optimization procedure), all of them should be executed within one scope to avoid long queue times when multiple users schedule their workloads.

c. A reservation approach that allows the full allocation of QH for a particular period.

d. Policies that may include limitations on the number of concurrent jobs a user can submit.

e. Feedback on the status of quantum jobs, including information on job progress, estimated completion times, and any issues encountered during execution, so jobs could be monitored and adjustments made based on feedback.

Most of the points mentioned above should be considered optional but important when trying to build enterprise-level QCPaaS actively used by hundreds of users simultaneously. In the minimal implementation, the Job Scheduler should communicate with individual hardware adapters that share queue/availability information. Also, runtime should have simple queue management that allows scheduling jobs on hardware, performing retries in cases of errors and handling priority access. This information could be used to build the minimal functionality of "job will start in x time". Scheduling should be resilient to failures. There should be a

mechanism that handles cases when the provider session reaches timeout and continuation should be built to not lose results for long-running jobs, e.g., those 1 day and longer, which are expensive to re-run.

What information could be retrieved from QH providers that could help with scheduling?

- IQM REST API [23] (used in Qiskit backend). IQM has a very limited interface in terms of job visibility. They only provide information about the status of a job, e.g., running, failing, and waiting. There are no endpoints to control at least some aspects of the queue.
- Rigetti API [24]. This has reservation, which allows the scheduling of 15 min of priority access at a specified time. Cost optimization could be implemented with the parallelization feature. The on-demand access feature allows a job to be run without reservation, but it is not possible to estimate wait time because reserved jobs could take priority. For some reason, there is no mechanism that updates wait time based on new reservation jobs, which are fixed in time.

The examples above are provided with the purpose of confirming that even established vendors do not standardize their core hardware APIs. The open QCPaaS initiative could also drive a standard for hardware API design, so emerging QH vendors and researchers could move faster with their new hardware platforms. In general, these differences in features and API designs create a lot of complexity and low-quality queue management/availability mechanisms that are common on the market right now.

2.3.4. Parametric compilation with calibration data (DD-11)

Calibration data refer to the set of parameters and measurements (coherence times, qubit frequency, gate fidelities, error rates, etc.) that describe the current physical state and performance characteristics of QPUs. These data are essential for the effective use of QH, enabling more precise control over qubit behavior, reducing errors, and ultimately leading to more reliable quantum computations. Depending on the type of QH and many other external factors like temperature fluctuations, electromagnetic interference and many other noise sources calibration data could change many times per day. For long-running quantum workloads, it is important to consider this. Quantum SDKs like Amazon Braket offer feature called “parametric compilation” [25–27] as part of hybrid jobs. Primarily, it is a mechanism to optimize computation by removing the need to compile circuits in every iteration of a hybrid job which is quite time consuming. Along with that, it also incorporates the latest quantum device calibration data to ensure that results are produced with higher quality.

The intent of this section is an attempt to reverse-engineer Amazon’s implementation (considering that it is proprietary module hidden under the AWS Braket platform) and outline architectural concerns that should be considering when implementing this feature in QCPaaS.

The main responsibility of the Calibration Manager component in **Figure 2** (DD-9) is the retrieval and storage of the latest calibration data from the quantum hardware. It could retrieve these data periodically or on demand. The assumption is that hardware adapters for QPUs allow access to that data to be obtained. The Workflow Manager should ensure that execution is aligned with the real-time state of the hardware by using data from the Calibration Manager. It should be knowledgeable enough to ensure minimal delay between compilation and execution to reduce the risk of hardware state changes. These two components rely heavily on the compiler that supports the parametric compilation feature. As this reference architecture does not tie implementation to a specific programming SDK which usually comes with compiler, this component is assumed to be chosen arbitrarily from the list of SDKs that support this feature, e.g., Qiskit [27]. It requires some customization though. Functionality should be extended so that during the parameter binding phase, the retrieved calibration data are used to determine the optimal values for the circuit parameters. This could involve updating gate parameters like rotation angles based on qubit frequency or adjusting gate timings based on the latest coherence times. Also, there should be a change at the transpiler level to delay final compilation until the binding step is complete, ensuring that the calibration data are applied as close to execution time as possible. The transpiler would optimize the circuit based on these real-time data, potentially adjusting gate sequences or error mitigation strategies. Qiskit’s execution pipeline should be modified to include a feedback loop where post-execution data (e.g., fidelity metrics) are used to inform future parameter bindings or calibration data updates. This could be quite a complex endeavor and depends on main programming framework implementation. A more detailed design could be planned once the Calibration Management module of Qiskit [28] is fully established.

Some of the architectural concerns (ACs) (fundamental considerations about technical challenges) that should be covered during implementation include the following:

- AC-1. Ensuring that the entire process—from retrieving calibration data to binding parameters and executing the circuit—happens swiftly is crucial to maintain accuracy.
- AC-2. Quantum hardware is highly sensitive, and its properties can change rapidly. Ensuring that the compilation remains accurate in such an environment is a major challenge.
- AC-3. Real-time data integration: Integrating real-time calibration data into the compilation process without introducing significant delays requires efficient data management and processing pipelines.
- AC-3. The complexity of compilation: the parametric compiler must be sophisticated enough to optimize the circuit based on detailed and potentially complex calibration data, which can involve intricate quantum mechanical considerations. As an alternative to Qiskit compiler, QIR-based implementation [29] and its benefits will be reviewed in the next sections.
- AC-3. The system must be able to scale with the number of users and circuits being processed. This requires the efficient management of calibration data and a robust scheduling system to handle potentially high volumes of parametric compilations.
- AC-4. If the calibration data change significantly between compilation and execution, or if the hardware state is not as expected, the system should either recompile the circuit or provide a mechanism for flagging or retrying the job.

A good candidate to incorporate into this design for the implementation of an open-source framework to perform quantum calibration and characterization is Qibocal [30]. This paper does not explicitly cover all the architectural concerns listed above but provides quite solid ground for moving forward with this important feature. A fresh look at the complexities of incorporating calibration data into error-aware compilation for NISQ devices is provided in [31]. The paper follows an interesting direction of analyzing historical calibration data and applying noise-aware compilation techniques based on them.

2.4. Architecture vision summary

An understanding of how different parts are implemented could be achieved by looking into the prototype code [32]. The readme file has an explanation and general guide. The prototype does not include all features designed as part of this architecture, e.g., scheduling based on availability, the calibration of data, and parametric compilation.

To reason about different architectural decisions and drivers that they fulfill, **Table 1** provides a quick summary with a mapping. Some drivers are partially covered and require more design work, while some are fully left for future ADD iterations, e.g., AD-18 and AD-22.

2.5. Hybrid quantum–classical computations: QCPaaS and outlook beyond

QCPaaS optimization gains that are coming as part of the design include the following:

- Reduced latency in circuit execution and in communication between classical computation and QPUs.
- Parametric compilation that considers calibration data.
- A basic scheduling mechanism.
- Batching iterations of long-running jobs together (as part of the session mechanism).
- A flexible pipeline mechanism that allows the inclusion of custom optimization steps, e.g., EM.

They will provide benefits for algorithms like VQE and QAOA and offer what IBM calls [33] near-time execution benefit. Circuit initialization and communication between classical and quantum computation still takes significant time and prevents achieving the real-time execution of classical code when maintaining QPU qubit coherence. To move closer to this goal, other areas of the software stack should be covered for improvement in quantum hardware design.

2.5.1. Needed improvements from the hardware side

As outlined in [34], classical computing has always been used in quantum computing for measuring results, control configurations, etc. The Microsoft quantum research group [35] puts emphasis on specialized hardware like arbitrary waveform generators (AWGs) and field programmable gate arrays (FPGAs) which are used in a common QPUs to handle precise pulses of microwaves or lasers as part of the control stack. This amplifies that this specialized hardware comes with a loss of general-purpose computational

Table 1 • Summary of Design Decisions and which Architecture Drivers they address.

Design decision	Covered or partially covered Architecture Drivers
DD-1	AD-7, AD-11, AD-13
DD-2	AD-7, AD-16
DD-3	AD-15
DD-4	AD-4, AD-5, AD-6, AD-9, AD-10, AD-13
DD-5	AD-15
DD-6	AD-17
DD-7	AD-21
DD-8	AD-7, AD-8, AD-12, AD-20, AD-8, AD-2
DD-9	AD-12, AD-2, AD-3
DD-10	AD-18, AD-17, AD-8, AD-6, AD-5, AD-4, AD-3, AD-1
DD-11	AD-12, AD-2, AD-3
DD-12	AD-7, AD-8, AD-12
DD-13	AD-4, AD-5, AD-6, AD-9, AD-10, AD-13
DD-14	AD-3, AD-5, AD-7, AD-8, AD-10
DD-15	AD-19, AD-11, AD-12

features that are required for popular hybrid algorithms. Newer generations of quantum hardware are actively trying to address these challenges by introducing all required classical capabilities on a single chip. One prominent example is ultra-low-latency chip-to-chip links between quantum computers, GPUs, and CPUs that are in active development by Seeqc [36]. The difficulties in effectively coordinating a dedicated accelerator with a central processing unit are not specific to quantum computing. Modern computing practices, especially the use of GPUs, have inspired approaches for managing data transfer between processors and improving code portability, as well as integration with existing tools and technologies. Little by little, quantum computing is moving in that direction. Effort is being made in collaboration with NVIDIA and their DGX Quantum platform [37]. DGX Quantum allows the delivery of submicrosecond latency between GPUs and QPUs, accelerating hybrid workloads. Having an ability to deploy QCPaaS on DGX Quantum and accessing QPUs of choice from it (which should be physically close enough) would be very interesting to explore new QU possibilities. However, it does not look like NVIDIA is going that way, at least not for now. But, it also does not mean that Seeqc will not scale partnerships in the broader community.

2.5.2. Next steps at software stack level

Ref. [1] defines OpenQASM as a main instrument for quantum optimization research. It continues to be an important player, especially with the OpenQASM 3.0 update which provides many features for real-time hybrid computations. It is a good choice for programming modern quantum–classical interactions. From another perspective, QIR [29] recommends itself to be more advantageous for optimizing and deploying quantum programs across platforms. QIR provides a powerful intermediate representation that is useful in large-scale quantum computing projects where performance optimization and cross-platform compatibility are critical. It typically requires a higher-level language to manage the real-time aspects of hybrid computation. Its adoption has been accelerated by Microsoft, NVIDIA (which uses it as part of a programming model for CUDA-Q [38] and DGX Quantum, respectively), and several other companies. From these observations, it feels more "native" and convenient for expressing logic for data exchange and processing, while qubits remain in use. QIR's foundation on LLVM IR [39] allows it to leverage the extensive LLVM toolchain for optimization and cross-compilation, including sophisticated analyses and transformations that are not directly applicable in OpenQASM. The seamless integration between quantum and classical code components that QIR aims to provide is more about the compilation and execution pipeline, including optimizations across the quantum–classical boundary. OpenQASM focuses on specifying the quantum circuit and its immediate classical control logic but does not inherently provide the same level of support for optimization and cross-compilation as QIR does. The study [35] focuses on innovative software components that leverage QIR to remove limitations of existing established computational models that raise the efficiency of quantum algorithms. Results look promising, so there is a very solid ground to continue further research based on QIR.

With all the innovation expected in both hardware and software stacks, QCPaaS will continue to serve a role as a mandatary platform in the overall ecosystem, adopting its internal design along with programming model to the needs of programming stacks based on QIR and OpenQASM. As of now, they could be fully integrated and are first-class components in the QCPaaS architecture vision. Containers with programming SDKs based on QIR, e.g., NVIDIA-Q, are deployed on runtimes. End-user designs and algorithms use the same SDK and schedule workloads for execution, for which a runtime identifies containers with NVIDIA-Q. Physically, classical–quantum computation is co-located via hypothetical low-link connections to bring the full potential for hybrid algorithm execution. If the assumption on the ecosystem's development is correct, QUs' potential looks close enough.

2.6. Potential experimental protocols

1. Running a single circuit job with custom error mitigation not fully available in mitiq. This could be part of an individual library or custom method (zne+ddd) API which is not directly available via mitiq. This case foresees the creation of a custom function/module that runs the job. The prototype has the implementation of zne+ddd. The path in the code is qc-paas/prototype_chunks/custom_em_flow_transpile_zne_ddd.

2. Running a batch job that allows the configuration of separate EM methods for a particular circuit, including custom zne+ddd for some of them.

3. Basic QML with custom error mitigation (zne+ddd). The path in the code is qc-paas/full_flow/client_example/qml_example_worker.py

4. Running a batch job with error mitigation from the mitiq library (zne, pce, cdr, classical shadows, ddd, readout-error, qse, twirling). Batch-to-batch processing is possible under the hood (batch job with zne error mitigation (EM)). A competitive advantage could be proper scheduling to estimate all batches of jobs together to execute them as fast as possible. The Workflow Manager is responsible for handling scheduling and execution time reservation.

3. Discussion

As mentioned in [1], the industry effort is focused primarily on cloud-based quantum hardware exposure and utilization along with tooling that helps to design algorithmic advancements. This reflects profit-driven capitalism, which is normal considering that significant investments in the QC capabilities of these companies should be paid off in at least mid-term with minimal risks. In the center of this value chain, there are major inefficiencies caused by closed-ecosystem implementations of runtime environments and inflexible programming models. There is a lot of room for improvement in QC execution quality which could be performed using the available tooling, but our hypothesis is that by opening the design of QCPaaS, allowing everyone to contribute and customize it, could have a profound effect and major push towards reaching QU and will be actively adopted as the standard case. We want to believe that it will also stimulate major vendors to make their quantum runtimes open-source, so the best standardized runtime platform could emerge in the upcoming years and one more industry-wide milestone could be reached.

This paper complements the QCOM architecture [1] by providing required tooling for the implementation of optimization steps that require remote/co-located execution.

The QCPaaS reference architecture defined as part of this research plays a confident and irreplaceable role in the near-term future of QU. While more in-depth analysis and prototyping activities are required to cover complexities and in scheduling, resource allocation, hardware adapter mechanisms, etc., most of the architectural decisions will apply to most remote execution use cases.

Next steps in research should be taking a prototype and deploying it physically close to real QPU hardware implementation to measure quantitative metrics, e.g., fidelity. Scheduling mechanisms and parametric compilation implementation along with a proposal for hardware adapter mechanisms could be good supplements and improvements for the QCPaaS in version 2.0. Benchmarking different algorithmic approaches, the path to which is opened by a programming model being designed, should be the most interesting and potentially the most influential output.

4. Conclusions

This study demonstrates the importance of the standardization and adoption of the ecosystem-agnostic QCPaaS architecture in competitive QC optimization research. Value proposition driven by industry needs is transitioned into the proposed QCPaaS architecture (design of its core components and programming model) and prototype. Unlike previous studies, our work concentrates on practical challenges and needs, providing a perspective on new experimental strategies that are not possible with other widely known tools. This could lead to the discovery of new methods that positively impact QC performance.

Although this study provides strong evidence for the benefit of the QCPaaS, practical experiments and validation have not been conducted yet due to the challenges of establishing partnerships with quantum hardware vendors. This work is foundational, and experimental validation is left for future work.

As QC continues to evolve, the development of ecosystem-agnostic runtime architectures will be crucial in unlocking its full potential. Our work represents an essential step toward that goal.

Funding

This research received no external funding.

Author contributions

Conceptualization, M.T., I.K.; methodology, M.T., I.K.; software, M.T.; validation, M.T., K.I; formal analysis, M.T.; investigation, M.T.; resources, M.T.; data curation, M.T.; writing—original draft preparation, M.T.; writing—review and editing, M.T.; visualization, M.T.; supervision, I.K.; project administration, I.K.; funding acquisition, I.K. All authors have read and agreed to the published version of the manuscript.

Conflict of interest

The authors declare no conflict of interest.

Data availability statement

This study does not involve empirical data. However, a prototype of the proposed architecture is available on GitHub at https://github.com/markiiantsymbalista/qcpaas.

Institutional review board statement

Not applicable.

Informed consent statement

Not applicable.

Additional information

Received: 2024-12-31

Accepted: 2025-03-18

Published: 2025-04-14

Academia Quantum papers should be cited as *Academia Quantum 2025*, ISSN 3064-979X, https://doi.org/10.20935/AcadQuant7627. The journal's official abbreviation is *Acad. Quant.*

Publisher's note

Copyright

References

1. Tsymbalista M, Maksymenko M, Katernyak I. Approaching Quantum Utility by leveraging quantum software stack. Proceedings of the 2023 IEEE 13th International Conference on Electronics and Information Technologies (ELIT); 2023 September 26–28; Lviv, Ukraine. 2023. doi: 10.1109/ELIT61488.2023.10310743

2. Shammah N, Roy AS, Almudever CG, Bourdeauducq S, Butko A, Cancelo G, et al. Open Hardware Solutions in Quantum Technology. APL Quantum. 2023;1:011501. doi: 10.1063/5.0180987

3. Value Proposition Canvas. [cited 2025 Mar 9]. Available from: https://www.strategyzer.com/library/the-value-proposition-canvas
4. Quality Attribute Scenarios in Practice. [cited 2025 Mar 9]. Available from: https://people.ece.ubc.ca/matei/EECE417/BASS/ch04lev1sec4.html
5. Cervantes H, Kazman R. Designing software architectures: a practical approach (SEI series in software engineering). Boston (MA): Addison-Wesley Professional; 2016.
6. Gordon P, Clements P, Kazman R, Klein M. Evaluating software architectures: methods and case studies. Boston (MA): Addison-Wesley Professional; 2001.
7. IBM Qiskit Runtime. [cited 2025 Mar 9]. Available from: https://cloud.ibm.com/quantum
8. Qiskit Dell Runtime. [cited 2025 Mar 9]. Available from: https://github.com/qiskit-community/qiskit-dell-runtime
9. Strangerworks qiskit runtime. [cited 2025 Mar 9]. Available from: https://docs.strangeworks.com/quantum/qiskit-runtime#strangeworks-qiskit-runtime
10. Pennylane catalyst. [cited 2025 Mar 9]. Available from: https://docs.pennylane.ai/projects/catalyst/en/latest/dev/architecture.html
11. Intel runtime. [cited 2025 Mar 9]. Available from: https://www.intel.com/content/www/us/en/developer/tools/quantum-sdk/overview.html
12. Grossi M, Crippa L, Aita A, Bartoli G, Sammarco V, Picca E, et al. A serverless cloud integration for quantum computing. arXiv. 2021. doi: 10.48550/arXiv.2107.02007
13. Nguyen HT, Usman M, Buyya R. QFaaS: A Serverless Function-as-a-Service Framework for Quantum Computing. arXiv. 2022. doi: 10.48550/arXiv.2205.14845
14. Terraform. [cited 2025 Mar 9]. Available from: https://www.terraform.io/
15. Mitiq. [cited 2025 Mar 9]. Available from: https://unitary.fund/research/mitiq/
16. Decorator OOD pattern. [cited 2025 Mar 9]. Available from: https://refactoring.guru/design-patterns/decorator
17. Qiskit Providers Interface. [cited 2025 Mar 9]. Available from: https://docs.quantum.ibm.com/api/qiskit/providers
18. Qiskit BackendV2. [cited 2025 Mar 9]. Available from: https://docs.quantum.ibm.com/api/qiskit/qiskit.providers.BackendV2
19. Qiskit IQM adapter. [cited 2025 Mar 9]. Available from: https://github.com/iqm-finland/qiskit-on-iqm
20. van Dam W, Mykhailova M, Soeken M. Using azure quantum resource estimator for assessing performance of fault tolerant quantum computation. Proceedings of the SC '23 workshops of the international conference on high performance computing, network, storage, and analysis; 2023 November 12–17; Denver, CO, USA. 2023. doi: 10.1145/3624062.3624211
21. Azure Resource Estimator. [cited 2025 Mar 9]. Available from: https://learn.microsoft.com/en-us/azure/quantum/intro-to-resource-estimation
22. Ravi GS, Smith KN, Murali P, Chong FT. Adaptive job and resource management for the growing quantum cloud. Proceedings of the 2021 IEEE International Conference on Quantum Computing and Engineering (QCE); 2021 October 17–21; Broomfield, CO, USA. 2022. doi: 10.1109/QCE52317.2021.00047
23. IQM REST API. [cited 2025 Mar 9]. Available from: https://iqm-finland.github.io/qiskit-on-iqm/api/iqm.qiskit_iqm.html#module-iqm.qiskit_iqm
24. Rigetti API. [cited 2025 Mar 9]. Available from: https://docs.rigetti.com/qcs/guides/access-a-qpu
25. Using parametric compilation to speed up Hybrid Jobs. [cited 2025 Mar 9]. Available from: https://docs.aws.amazon.com/braket/latest/developerguide/braket-jobs-parametric-compilation.html
26. Speeding up hybrid quantum algorithms with parametric circuits on Amazon Braket. [cited 2025 Mar 9]. Available from: https://aws.amazon.com/blogs/quantum-computing/speeding-up-hybrid-quantum-algorithms-with-parametric-circuits-on-amazon-braket/
27. Qiskit parameterized circuits. [cited 2025 Mar 9]. Available from: https://docs.quantum.ibm.com/guides/construct-circuits#parameterized-circuits
28. Qiskit Calibration Management. [cited 2025 Mar 9]. Available from: https://qiskit-community.github.io/qiskit-experiments/stable/0.5/apidocs/calibration_management.html
29. QIR alliance. [cited 2025 Mar 9]. Available from: https://www.qir-alliance.org/
30. Pasquale A, Efthymiou S, Ramos-Calderer S, Wilkens J, Roth I, Carrazza S. Towards an open-source framework to perform quantum calibration and characterization. arXiv. 2024. doi: 10.48550/arXiv.2303.10397
31. Kurniawan H, Rodríguez-Soriano L, Cuomo D, Almudever CG, Herrero FG. On the use of calibration data in error-aware compilation techniques for NISQ devices. arXiv. 2024. doi: 10.48550/arXiv.2407.21462
32. QCPaaS prototype. [cited 2025 Mar 9]. Available from: https://github.com/markiiantsymbalista/qcpaas
33. Rethinking quantum systems for faster, more efficient computation. [cited 2025 Mar 9]. Available from: https://www.ibm.com/quantum/blog/near-real-time-quantum-compute
34. Adoption of quantum computing operating model. [cited 2025 Mar 9]. Available from: https://www.linkedin.com/pulse/adoption-quantum-computing-operating-model-markiian-tsymbalista-qjtif/?trackingId=EBv9kmHqTo2ZxFvsdg07Kw%3D%3D
35. Lubinski T, Granade C, Anderson A, Geller A, Roetteler M, Petrenko A, et al. Advancing Hybrid Quantum-Classical Computation with Real-Time Execution. Front Phys. 2022;10:940293. doi: 10.3389/fphy.2022.940293

36. Seeqc. [cited 2025 Mar 9]. Available from: https://seeqc.com/overview

37. Nvidia DGX Quantum. [cited 2025 Mar 9]. Available from: https://www.nvidia.com/en-us/data-center/dgx-quantum/

38. CUDA-Q. [cited 2025 Mar 9]. Available from: https://nvidia.github.io/cuda-quantum/latest/index.html

39. LLVM IR. [cited 2025 Mar 9]. Available from: https://llvm.org/