Quantum Information Processing (2026) 25:103
https://doi.org/10.1007/s11128-026-05119-y

Layer model for gate-based modular quantum computers

Rob F. M. van den Brink[1] · Juan C. Boschero[2] · Michele Amoretti[3]

Received: 9 May 2025 / Accepted: 18 February 2026

Abstract
This paper elaborates on a consensus view from the standardization body JTC22/WG3 on a layer model that offers a high-level hierarchical description of hardware and software modules to support the development of modular quantum computers. The group of lower layers covers multiple stacks, one for each hardware architecture, while the upper layers are defined around a common software stack. A hardware abstraction layer connects the two. The addition of a separate communication unit offers a robust separation between the stack and the outside world and allows controlled access for multiple users, as well as serving hybrid computing systems.

Keywords Quantum computing · Layer model · Standardization · Consensus view · Full-stack · Modularity · Functional description · Framework

1 Introduction

A properly defined layer model is the first step toward a modular quantum computer. Modularity would enable customers, such as research institutes and system integrators, to select products (modules) from different origins and combine them into a single quantum computer. It is a proven concept that has given the market for personal computers a boost, and our aim is to realize that for future quantum computers as well.

It must be noted that modularity in this paper does not refer to “horizontal" modularity, such as modular methods to scale-up the size of a quantum computer. In those solutions, multiple hardware modules on a smaller scale are combined into a single quantum computer of much larger size. With horizontal modularity, the modules involved offer *similar* functionalities from the *same* vendor that are interconnected in parallel [1–3] The use of proprietary interfaces is often adequate for this kind of modularity.

✉ Juan C. Boschero
juan.boschero@tno.nl

[1] Delft Circuits, Delft, The Netherlands

[2] TNO, Applied Cryptography & Quantum Algorithms, Den Haag, The Netherlands

[3] CINI HPC-KTT Laboratory, University of Parma, Parma, Italy

Published online: 17 March 2026

This paper refers to something that might be called “vertical" modularity. In those solutions, the modules involved offer *complementary* solutions from *different* vendors that are hierarchically stacked. The layers of such stacks require standardized interfaces between the layers to combine products from different vendors. Vertical modularity requires a mature supply chain from which customers can acquire the best combination of products that meets their requirements. This concept can only flourish when hardware and software modules from all those suppliers can interwork with each other through well defined, commonly accepted interfaces. This also means that the functionality, requirements, and interfaces of all these modules are well specified by dedicated standards.

The biggest challenge here is not to create a working stacking solution for quantum computers [4–6], but *to build consensus* among competitive commercial parties with a common interest in a global market. This is a delicate process that cannot proceed until a consensus view has been achieved on how to slice a quantum computer into smaller chunks. It requires a top-down approach for identifying a stack of hierarchical layers and high-level descriptions of their functionalities.

This is where standardization comes into play, and the JTC22/WG3 standardization body [7] has recently published such a layer model [8] as a follow up to the roadmap proposed by the Focus Group on Quantum Technologies [9]. This layer model is an abstract description of a (computing) system through a common stack of layers and provides good guidance on how to proceed. The next step is to elaborate on functional requirements in different and more dedicated standard documents before technical details and interfaces can be specified. This effort is complementary to other standardization activities in the world. The IEC/ISO JTC3 was established in 2024 and has recently initiated work on quantum computing supply chains and benchmarking [10]. A standard on quantum computing vocabulary has been put forth; however, it does not contain any information on quantum computing architectures [11]. We expect that any future architectural work by JTC3 will take into account the documents produced by CEN/CENELEC JTC22 and consequently, this research as input for terminology describing technical functionality. Although IEEE is currently working on hardware and software standards for quantum computing, as of June 19th, 2025, the IEEE P3120, *Standard for Quantum Computing Architecture*, has been withdrawn until further notice [12].

A quantum computing layer model is not a new concept, as literature exists proposing layered architectures for quantum computers. Jones et al. [13] proposed a layer model for quantum computers using optically controlled quantum dots to address scalability challenges for quantum error correction [13]. Although research does make advances in detailing scalable layers, the architecture is specified for one form of gate-based quantum computing. Furthermore, the research does not address the practical details of users interacting with the quantum computer, whether locally or in the cloud. Geck et al. [14] proposed a complete-circuit architecture for electron spin qubits that improves scalability by physically placing the control circuits close to the qubits at cryogenic temperatures. This work proposes a generic five-layer architecture in which the physical/hardware layers are heavily discussed while higher-level software is left out of scope. Consequently, this work does not address the practical concerns of user-oriented usage of a gate-based computer. The quantum layer model described by Fu

et al. [15] is more generic toward a universal gate-based computer, with seven vertical layers proposed and additional *codes* or applications introduced horizontally. The research clearly differentiates between logical and physical qubits, even discussing the implications of using ancillae in an architecture. However, the interaction of users on the layers, whether locally or from the cloud, as well as the data transfer between the layers, is not explicitly mentioned. Bandic et al. [16], describing a simplified full-stack system in the NISQ era, proposed a generalization of that seven-layer stack.

Recently, a full stack for networked quantum devices was proposed by Delle Donne et al. [17], where the software layers (including a network stack [18]) are integrated into a novel operating system denoted as QNodeOS. To the best of our knowledge, this is the most advanced layered architecture for few-qubit quantum devices that share entangled qubit [19, 20] pairs over short, medium, or long distances. That is, the focus is on quantum communications rather than quantum computing. For this reason, the full stack by Delle Donne et al. cannot be directly compared with the one proposed in this paper, which is oriented toward quantum computing on devices equipped with many qubits. Furthermore, as illustrated in Section 2, the proposed layer model devises a Communication Unit that is separate from the main stack. This unit allows for a clean interface between the internal parts of the stack and the outside world; for instance, it can become a node within a hybrid quantum computing network.

The research conducted by Stirbu et al. [21] presents a practical framework for integrating quantum computing into cloud-native environments using Kubernetes concepts, with a focus on orchestrating and executing hybrid classical–quantum tasks. In contrast, this paper proposes an architectural framework for modular quantum computers, emphasizing vertical modularity and interoperability across hardware and software layers. This blueprint describes properly defined layers to support the integration of orchestration software, such as that described in [21], into vendor modular quantum systems.

Building on the research overview provided, there is a notable gap in describing gate-based universal computing as general modular layers that the industry can readily adopt. This is particularly evident in the lack of higher-level software frameworks and cloud-based abstractions discussed in the existing literature. For instance, the paper by Murillo et al. [22] outlines key challenges in quantum software engineering, such as modularity, platform independence, and hybrid integration. By introducing standardized vertical layers, such as the hardware abstraction layer and a communication unit, the present paper translates these strategic needs into a practical architecture and offers a concrete blueprint for scalable and interoperable quantum systems, aligned with the roadmap proposed in [22]. More specifically, the present paper proposes layered architectures for universal gate-based computing, including examples of inputs and outputs to facilitate modular standardization within the industry. The aim is not only to define the layers and their purposes but also to outline their *functionalities* and processes necessary for maximizing marketability. Therefore, instead of delving into the technical details of each functionality, which we will leave out of the scope, this paper provides an indication of the information required both within and outside of each layer.

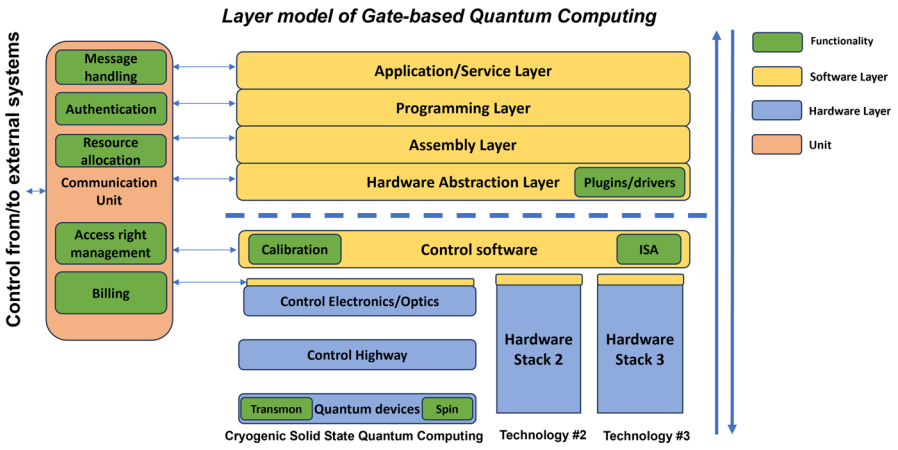

Fig. 1 Overview of the layer model of quantum computing, capable of supporting multiple hardware stacks. Figure from [8]

2 Overview of the layer model

A practical approach to managing the overall complexity of quantum computing is to conceptualize it as a stack of layers that collectively address the entire quantum computer. The proposed model is shown in Figure 1.

The stack covers hardware layers and software layers, each having dedicated functionalities. The layers are chosen in such a manner that the functionality of each layer can be described independently. This means that interworking between these layers is facilitated via well-defined interfaces at the boundaries of these layers. In principle, each layer interacts only with the one below and above it, but it is not excluded that interactions may bypass a layer to interact directly with one deeper or higher. Each layer aims to be more agnostic to the exact implementation of the layers below. The higher up in the stack, the more hardware-agnostic the layers will gradually become. By agnostic, we mean that the same system works for different quantum computing hardware platforms, such as solid state quantum computing, ion traps, neutral atoms, optical quantum computing, and topological quantum computing.

Since quantum computing is an area that encompasses many different implementations, the proposed model divides all layers into two groups. The group of upper layers addresses software, primarily at a higher level of abstraction, and is organized as a single stack. The group of lower layers mainly addresses hardware and depends on the physical platform. This group comprises multiple stacks, one for each identified architecture family, each with multiple layers. For the sake of clarity, in Figure 1, the groups of lower layers are *alternatives to one another*. In this work, we use cryogenic solid-state quantum hardware as an example to illustrate the functionalities of hardware layers. It is important to note that the layers may vary depending on the specific hardware technology.

The dashed line distinguishes the two groups, and layers above it may need to accommodate functionalities and interfaces below it, which can vary across different

hardware stacks. The aim of the hardware abstraction layer is to offer a more harmonized and common interface to the higher software layers. This approach decouples software design from hardware design to some extent, which has clear advantages, such as the reusability of algorithms for different hardware.

The addition of a communication unit next to the stack offers robust separation between the full stack and the outside world. It exchanges information directly with each layer but provides access to those layers only to users authenticated for that purpose. This addition simplifies the offering of quantum computation resources to a plurality of users and hybrid computing systems.

A module, within this context, is something that can be sold and shipped independently of other modules. It can offer the functionality of a single layer, multiple layers, or a subset of a layer. In all cases, it requires interfaces for interworking with other modules.

The boundaries between the layers are natural locations for defining standardized interfaces so that modules can take advantage of them. But when the functionality of a module spans two or more layers, there is no need to implement the interfaces between the inner layers. A module may also support different operating modes, such that it complies with the various requirements for each (member of an) architecture family.

Figure 1 shows an overview of the proposed layer model.

So far, the following quantum architecture families have been identified (in arbitrary order):

- Cryogenic solid-state based [23–30]
- Room temperature solid-state based [31, 32]
- Trapped ions [33, 34]
- Neutral atoms [35, 36]
- Photonic quantum computing [37, 38]
- Other architectures that may be identified in future

A one-size-fits all approach may not apply to all these different architectures; therefore, each may have its own stack. The use of four lower layers has been shown to be adequate for serving the needs of cryogenic solid-state technologies. Other architectures, depicted in this figure as technology #2 and #3, may need another division into layers. Their stack has, therefore, been drawn as a single box, and the details are left for further study.

Within an architecture family, multiple members may exist, such as transmons and spin qubits for cryogenic solid-state QC. Small differences in the functionalities of the lower layers may therefore occur as well.

3 Hardware and control layers

As mentioned previously, to illustrate the layer model concept, only the hardware stack of cryogenic solid-state quantum computing is detailed here. Other architectures may require different stacks and are omitted for simplicity.

The members of this architecture family have in common that they all use a cryostat, where the quantum devices in a holder are controlled from outside the fridge by room temperature electronics. The following members have been identified within this architecture family:

- Transmons [23]
- Flux qubits [24]
- Semiconductor spin qubits [25]
- Topological qubits [26]
- Artificial atoms in solids [27]
- Molecular spins [29, 30]

3.1 Quantum devices

The quantum devices in hardware layer 1 function as modules containing qubits, typically operating at cryogenic temperatures and implemented either as chips or on a PCB. Their quantum states can be manipulated and read out by sending pulses and measuring their response. These devices may also have strict requirements regarding shielding, operating temperature, magnetic conditions, and other environmental factors.

3.2 Control highway

The control highway covers all hardware needed for transporting microwave, light wave, RF, and DC signals via electrical and/or optical means between the control electronics at room temperature and the quantum devices at cryogenic temperatures. It is usually a mix of transmission lines, filtering, attenuation, amplification, (de)multiplexing, as well as means for proper thermalization.

Downstream signals require attenuation at cryogenic temperatures to keep most of the thermal noise away from the qubits. Overall loss values of 50 dB or more are not uncommon. Additional filtering up to IR frequencies can reduce unwanted out-of-band noise even further. Since attenuators heat up by dissipating attenuated signals, they produce more thermal noise than desired. Thermalization is therefore required to keep the attenuators cool and to drain away most of the heat flow from room temperature nodes through the transmission lines. Superconducting sections can offer additional thermal isolation to prevent qubits from heating up.

Upstream signals require low-noise amplification, making it essential to minimize signal loss between the qubit and the first amplifier. When TWPA's are used as the first amplification stage, the control highway should also transport pump signals.

As the number of qubits in a single quantum computer grows rapidly, managing an extensive number of channels within a single cryostat becomes increasingly complex. The size of a control highway can easily become very bulky, making it more challenging to keep crosstalk under certain thresholds. Outgassing is also an issue that must be kept to a minimum since the control highway has to operate under demanding vacuum conditions. Moreover, it should be designed so that vibrations in the cryostat do not induce unwanted signals in the qubits. Therefore, a control highway is more than

just a collection of cables; it is a carefully designed subsystem essential for efficient operation. A convenient implementation of a control highway is a module offered as a top or side loader for insertion into a cryostat, having all thermalization onboard.

3.3 Control electronics

The hardware layer 3 covers all room temperature electronics for generating, receiving, and processing microwave, RF, and DC signals. Some implementations make use of routing/switching and/or multiplexing of control signals. It receives commands from higher layers to fire baseband and modulated pulses, generate pump signals, and perform measurements of qubit responses.

If these commands are standardized, the control electronics can easily be replaced by similar electronics from other brands. This capability usually requires a simple translation of standardized commands into proprietary hardware commands for storing samples in the memory of an AWG (Arbitrary Waveform Generator) or for firing a selected pulse. It is worth noting that this translation cannot be accomplished by the control software layer, which is unable to target specific control electronics systems.

Figure 1 illustrates that the translation of standardized commands into proprietary hardware commands can be accomplished through a thin software wrapper layered directly above the hardware, which serves as an integral component of layer 3. It can be offered as firmware built into the electronics or as an external piece of (driver) software.

3.4 Control software

The control software refers to the software systems and tools designed to manage, coordinate, and optimize operations dictated by higher-level languages. It plays a crucial role in translating higher-level quantum assembly instructions into commands that can be handled by the control electronics. The interface to higher layers is somewhat comparable to how one interfaces with the CPU's of classical computers. Different hardware stacks may offer similar functionality, but the instructions they can handle may be implementation dependent.

This layer may include an instruction set architecture (ISA), low-level quantum error correction (QEC), and calibration functionalities (as shown in Figure 1).

- **ISA (Instruction Set Architecture)** refers to a lower-level method of defining operations on a quantum computer. Instead of defining specific gates, this layer defines gates (or other instructions) as operations, using pulses that are applied for a certain time on specific qubits. An example of an instruction set architecture is pulse-level programming, where a user can specify wave pulses on qubits instead of gates. This requires knowledge of the system's control equipment, as well as the topology and the nature of the qubits.
- **Low-level Quantum Error Correction** refers to techniques for the detection or simple correction of errors, partly autonomously and partly controlled from higher layers. QEC as a whole [39, 40] is a functionality distributed over various layers, with the purpose of protecting quantum information from errors caused by noise

and decoherence. A detailed presentation of these techniques goes beyond the scope of this paper.
- **Calibration** refers to low-level methods to stabilize the hardware by continuously monitoring performance to maintain optimal operation.

3.4.1 Functionality of an ISA

The aim of an instruction set architecture (ISA) is to convert a sequence of machine-specific instructions from higher layers into commands for the control electronics to manage individual qubits. As such, the ISA has full awareness of the underlying quantum hardware and its topology. Due to the ISA's knowledge of the quantum hardware, it also has the responsibility of handling the execution timings and scheduling of individual instructions, such that higher layers should only know their sequence.

On input, the ISA may receive instructions from higher layers; for instance, to change the quantum state of qubits (gate instructions), to read out qubits (measurement instructions), or any other instructions to interact with all available qubits. These instructions can be provided to the ISA in a specific format, such as binary machine instructions, ASCII human-readable instructions, or function calls. Instructions intended for controlling one or two qubits may be fed one by one to the ISA, but it is more efficient if the ISA can handle many of them in parallel as a "vector" of instructions to interact with an ensemble of qubits simultaneously.

Higher layers can either push these instructions into a buffer within the ISA whenever the ISA signals readiness, or the ISA can poll instructions from a buffer within higher layers after completing the execution of a previous instruction group. This process also includes polling requests and instructions from multiple users. In all cases, it requires a well-defined interface (API) with the above layer(s), as well as a well-defined instruction set language (such as OpenPulse [41] or Pulser [42]).

An ISA may handle gate-level instructions as well as pulse-level instructions. Both may be mixed in a single compilation pass for bypassing specific gate instructions, which can be parsed in the Software Development Kit (SDK) by the user. Gate-level instructions are considered to be any set of operations that can be parsed onto universal gate-based quantum computers, regardless of the hardware, while pulse-level instructions are operations that are heavily dependent on the system's physical architecture. The ISA will thus support instructions to specify the exact waveform of a pulse to be fired to a specific qubit, as well as an ensemble of pulses, where each pulse has its own waveform and relative delay.

The common way of sending instructions to the ISA is via higher level layers such as the assembly or programming layer. Alternatively, a user may be allowed, via the communication module, access to the control software layer directly or via the hardware abstraction layer by supplying ISA-readable instructions.

On output, the ISA issues commands to the control hardware, such as triggering pulses to qubits or reading their responses through measurement. This requires the ISA to have full control over the timing of all these commands. If a pulse is to be applied to a qubit, the ISA may calculate its characteristics in real-time, such as waveform/pulse-shape and magnitude. Alternatively, it may also read predefined characteristics from a library created by other software units, stored either in the control software layer or

in the control electronics layer. In all cases, it requires a well-defined interface (API) with the control electronics as well as a well-defined command set.

4 Software layers

4.1 Hardware Abstraction Layer (HAL)

The aim of the Hardware Abstraction Layer for gate-based quantum computers is to inform the higher layers of the capabilities and limitations supported by the underlying hardware. Layers above the HAL can use this information to hide many implementation-specific details from higher layers by offering a more unified interface. The layers above may also use this information to provide higher-level commands to programmers or programs that allow for the implementation of hardware-specific optimizations and adaptations. Not all quantum computers can make use of all instructions or gates. The HAL can thus relay information about the underlying architecture and translate high-level instructions into commands that the hardware can execute.

A gate-based quantum computer processes a sequence of instructions to change the state of a quantum register with many qubits before the resulting state is queried through measurements. A convenient graphic representation of such a sequence has the appearance of a circuit in which the elements seem to operate on one or more qubits simultaneously. Due to this convenient graphic representation, these instructions are called gates.

Moreover, the HAL facilitates task scheduling by creating a queue for tasks submitted concurrently by multiple users. Furthermore, the scheduling can be optimized for time or resource efficiency.

Another functionality is that the HAL can select between multiple quantum architectures and even partition a single task into multiple queues to perform calculations in parallel on multiple architectures.

4.1.1 Organization of qubits

The HAL can report to higher layers how qubits are organized in one or more quantum registers. This is a system comprising multiple qubits, each with its own index. All qubits can be members of a single register or spread out over multiple (smaller) registers. The HAL supports instructions to operate on such registers for initializing, changing, and querying the state of the qubits. The HAL can report the properties of each register by means of the following parameters:

- *Width*: The HAL can specify the number of available qubits and how they are organized within these registers. It can also specify whether all qubits are part of a single quantum register or if they are allocated to multiple (smaller) registers. The use of multiple registers may occur when using modular hardware architectures.
- *Depth*: The HAL can specify the maximum depth for circuits of gates that can be executed before the calculated result becomes unreliable. This value is related to the coherence time of the implementation and other imperfections of the underlying hardware.

- *Connections*: The HAL can also provide an *adjacency matrix* for each quantum register to indicate which qubits are edge-connected. For instance, when a register has N qubits, then this adjacency matrix C has a size of $N \times N$. The default of each element in this matrix is false, but if qx and qy are the indices of two adjacent qubits, then $C(qx, qy) = C(qy, qx) =$true. Matrix C is therefore a symmetric matrix since $C(k, r) = C(r, k)$.

The HAL can provide additional information about the underlying architecture.

4.1.2 Concept of native and primitive gates

The HAL can provide a list of supported *native* and *primitive* gates, which can differ for different hardware implementations. The term *native* refers to an operation that changes the quantum state of a register by means of a "single" physical action on one or more qubits simultaneously. For instance, a rotation Rx(a) or Ry(b) in x or y direction via a single pulse. A *primitive* gate is a sequential combination of native gates that is emulated as a single operation. For instance, a Z-gate that is implemented as a sequence of an Rx and an Ry gate. In other words, if an operation is implemented by firing one or more *simultaneous* pulses, it is a native gate. If two or more *sequential* pulses are required to achieve the desired operation, it is a primitive gate. As a result, a native gate can be executed in the minimum execution time.

Knowledge about which gates are native is relevant information for compilers that try to optimize a circuit with respect to execution time. However, a compiler or interpreter does not always know how to convert well-known gates into a smart combination of native gates for any possible set of native gates. In those cases, a fall-back situation should be supported by the HAL in terms of predefined solutions for well-known gates like $\mathrm{Rx}(a), \mathrm{Ry}(b), \mathrm{Rz}(c), X, Y, Z, H, S, T, CNOT$, etc. Gates that are not implemented in the ISA can be emulated within the HAL to support commonly used gates. Therefore, the HAL should be able to specify a list of supported gates and tag those that are "primitive." Note that gates may be referred to as *compound* gates when they are emulated in layers above the HAL layer, such as a "single" QFT gate to emulate a quantum Fourier transform.

The boxed example in Table 1 illustrates, for a specific case, that the single-qubit gates X, Y, $Rx(a)$, and $Ry(b)$ are all native to that implementation, while the gates Z and $Rz(c)$ are primitive gates. A similar example can be elaborated with two-qubit gates. For a specific implementation, a gate like *CNOT* may also be considered primitive when it cannot be implemented with a single native two-qubit gate.

4.1.3 Concept of measurement

The HAL supports instructions to query the state of one or more qubits in a quantum register by means of a measurement. The answer will be returned as a binary string stored in a dedicated register. Note that the state will collapse after such a query. The HAL also supports instructions to read out the bits in this register and/or to use these bits for instructing controlled gates. If the hardware supports it, the HAL can also provide instructions to specify the basis for these measurements.

Table 1 Example of a specific hardware implementation, where Rx, Ry, X, Y are native gates and Rz, Z are compound gates

Example

The concept of native gates can be explained by the following example. Assume that a specific hardware implementation supports a mechanism to rotate a qubit via a “single" pulse composition that can be controlled with two real parameters “a" and “b". Assume that the definition of this rotation function equals:

$$RN(a,b) = \begin{bmatrix} \cos(a/2) & -i\exp(-ib)\sin(a/2) \\ -i\exp(ib)\sin(a/2) & \cos(a/2) \end{bmatrix} \tag{1}$$

Then some of the well known gates can be implemented via:

$$Rx(a) = \begin{bmatrix} \cos(a/2) & -i\sin(a/2) \\ -i\sin(a/2) & \cos(a/2) \end{bmatrix} = RN(a,0) \tag{2}$$

$$Ry(b) = \begin{bmatrix} \cos(b/2) & -\sin(b/2) \\ \sin(b/2) & \cos(b/2) \end{bmatrix} = RN(a,\pi/2) \tag{3}$$

$$Rz(c) = \begin{bmatrix} \exp(-ic/2) & 0 \\ 0 & \exp(ic/2) \end{bmatrix} = RN(\pi,0) * RN(\pi,-c/2) * \exp(i\pi) \tag{4}$$

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = Rx(\pi) * \exp(i\pi/2) = RN(\pi,0) * \exp(i\pi/2) \tag{5}$$

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} = Ry(\pi) * \exp(i\pi/2) = RN(\pi,\pi/2) * \exp(i\pi/2) \tag{6}$$

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = Rz(\pi) * \exp(i\pi/2) = RN(\pi,\pi) * RN(\pi,\pi/2) * \exp(-i\pi/2) \tag{7}$$

In this hardware implementation, $Rx(a)$, $Ry(b)$, X, Y can be considered as native gates. The gates $Rz(c)$ and Z are to be combined from two sequential native gates, so they are primitive gates. Knowledge about which gates are native is relevant for quantum algorithms that try to find an optimal circuit representation in terms of execution time.

4.1.4 Interfacing considerations

A preferred way of communicating with the HAL is by means of binary instructions, preferably common to all quantum computing implementations. Therefore, it requires a list of binary commands to allow the HAL to report the capabilities and limitations of the underlying hardware and to execute all the aforementioned instructions. Such an interface may also offer a convenient format for instructing a simulator that emulates a quantum computer with a limited set of qubits.

4.2 Assembly layer

This layer concerns quantum assembly languages, such as OpenQASM [43]. They describe quantum computations leveraging the gate-based quantum computer instruction set provided by the HAL while adding ways to define macros, data, and names (e.g., for functions). Such languages have in common that they can describe universal circuits with single- and two-qubit gates. Due to the immense diversity of gate-based quantum computing architectures, it is unlikely that a unique, widely accepted quantum assembly language will emerge and later become a standard.

4.3 Programming layer

The specification of quantum algorithms using register-level representation languages is not easy for programmers. Indeed, quantum assembly programs are usually generated by a software library from a piece of code written in a common programming language, such as Python. In general, the Programming Layer includes all the languages, libraries, and software development facilities for coding quantum algorithms or high-level applications that use predefined quantum algorithms as subroutines.

4.3.1 Programming languages and libraries

In the quantum computing domain, Python is the most widely used high-level programming language. It is a general-purpose imperative language, as it allows developers to write code that specifies the steps the computer must take to accomplish the goal. Other imperative languages have been specifically designed for quantum computing, such as Q# [44] and [45]. An alternative to imperative programming is functional programming, where programs are constructed by applying and composing functions. In the quantum computing domain, there are a few functional programming languages, one of which being Quipper [46]. Writing a program in a high-level language implies using software development kits (SDKs) that include application programming interfaces (APIs) for coding quantum algorithms from scratch, as well as collections of ready-to-use quantum algorithms. The APIs may vary significantly depending on the quantum computational model (solely digital quantum computers based on gates, digital–analog quantum computers, etc.) and the specific application domain (quantum optimization, quantum machine learning, etc.). For Python programmers, there are several advanced SDKs. The most advanced SDKs support device architectures from multiple providers, provided that the quantum computational model is the same. Examples are Qiskit [47] and Cirq [48], concerning the quantum circuit model.

4.3.2 Quantum compilation

Being high-level programs that are hardware-agnostic, quantum compilers [49–52] are necessary to translate abstract quantum algorithms into the most efficient equivalents of themselves, considering the constraints and features exposed by the register-level representation layer. The input to the quantum compiler is a quantum circuit that includes single- or multi-qubit gates. Usually, the input circuit is the simplest (and most elegant) representation of a quantum algorithm (e.g., the Quantum Fourier Transform). Such a representation does not consider the constraints that may characterize the target quantum computer, such as the available gate set and the connectivity constraints regarding which qubits a two-qubit gate is natively allowed. The quantum compiler leverages information provided by the register-level representation layer to translate the input circuit into an equivalent circuit that fits the target device. An example is provided in Figure 2, in which a quantum circuit is compiled into another quantum circuit while considering the connectivity constraints of the target quantum computer.

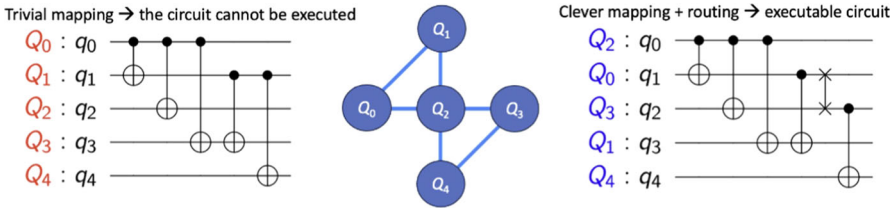

Fig. 2 The circuit on the left does not fit the connectivity constraints of the target device, which are described by the graph in the middle of the figure. The circuit on the right is the compiled version of the circuit on the left, i.e., functionally equivalent but fitting the target device. To produce the output circuit, the compiler chose a different mapping for the input circuit's qubits to the device qubits and inserted a SWAP gate before the last CNOT gate. Figure from [8]

The description format of the output circuit may differ from the description format of the input circuit. If the input and output circuits have the same description format, the compiler is often referred to as a 'transpiler'.

4.4 Service layer

This layer contains the user-side, where a task, or subset of a task, exists that requires execution. Quantum computers can help execute this task, and the user can then start programming algorithms to obtain the sought-after answer. Depending on the service used, users may perform tasks locally on a quantum computer. An alternative is to run tasks mainly remotely on a classical computer and use quantum computing as a service (QCaaS) to run specific tasks on a dedicated quantum computer.

5 Communication unit

Currently, commercial quantum computers are built for cloud-based computing or at least offer access to various end-users. This means that users who want to execute algorithms on a gate-based quantum computer from outside the stack must place a request to gain access to one or more (software) layers inside the stack. For this purpose, it is crucial that each software layer can be accessed by the communication unit. The communication unit can exchange messages with client applications that run outside the quantum stack, for instance, on a nearby computer or on a remote server somewhere in the cloud. It can handle all messages that are needed for starting a quantum computing session, including:

- Handshaking - a protocol for initiating communication between remote nodes.
- Message handling - refers to the means of exchanging information between two nodes.
- Authentication - is the user allowed to access the system?
- Access right management - which layers does this user have access to?
- Resource allocation - reserving memory, time slots, priorities, etc.
- Billing - counting how many resources have been used.

A quantum computing session offers an application the experience as if it has its own resources and is fully protected from other applications.

The communication unit can also communicate directly with the lower layers of the quantum stack, provided that the client application is allowed to do so according to allocated usage rights. For instance, to send low-level commands directly to the control electronics for firing a specific pulse to a qubit. Detected results from the control electronics can also be passed back to the communication layer, which, in turn, can send messages with those results to the client application outside the quantum stack.

5.1 Example information flow

Given the communications unit's role in linking the software layers, several scenarios will be presented to illustrate its purpose and workflow.

5.1.1 Single user accessing the full quantum stack

Assume a session has been initiated by an external user or client application. The communication unit will then start handling incoming *messages* and communicate them, for instance, with the service layer to load and run a task. The associated program will be compiled or interpreted into a sequence of *instructions* that are then sent to the HAL. Most of these instructions are passed over with minimal conversion effort to the ISA within the control software layer. The ISA generates, in turn, low-level *commands* to the control electronics to allow it to produce a variety of (analog) *signals*, such as baseband or modulated pulses. These signals are transported through the control highway to reach the qubits.

Once an instruction is executed to measure a qubit response, the measured result is reported back to the calling program in the higher layers. The results from this program are communicated back to the communication unit, which, in turn, sends them as *messages* to the external user or client application.

5.1.2 Multiple users accessing the full quantum stack

Assume multiple users with similar privileges have initialized their sessions simultaneously, and each of them has access to the stack. The communication unit is responsible for processing incoming *messages*, verifying access permissions, checking user privileges, allocating resources for each session, and keeping all sessions separate from one another. The users submit their respective tasks, and the communication unit may ask a compiler to compile each task to store the results in an allocated thread within the HAL. The HAL offers scheduling and priority for each thread based on user-specific information flagged by the communication unit. The scheduled instructions in the output queue are sequentially fed to the control software layer for further processing, as previously described.

Once the circuit is executed, the measured result is communicated back to the HAL, which in turn communicates it to the calling program.

5.1.3 User accessing lower layer

Assume a super-user with adequate access rights wants to upgrade low-level software and submit it directly to a lower layer. For instance, an upgrade of the ISA functionality within the control software layer. The super-user sends this upgraded software to the communication unit, including information about the target layer. The communication unit sends “wait” requests to all layers, pauses until full confirmation, upgrades the low-level software, sends “success” or “failure” messages back to the super-user, and reports “done” to all layers. After that, the full stack can proceed as usual.

6 Discussion

The boundaries in the proposed layer model are designed to allow each layer to be developed as a standalone product, independent of the others. To ensure seamless integration, information exchange between layers must occur through well-defined interfaces. Since this layer model is the result of a consensus view among stakeholders within JTC22/WG3, it has already proven its value in defining interactions with functionalities outside the quantum stack. A working item on hybrid quantum computing, which is currently under development within JTC22/WG3, takes advantage of the "communication unit" to describe how to interface from the cloud with a quantum node.

Establishing a consensus among both vendors and customers on these interfaces and the functionality of each layer will be highly beneficial to all stakeholders in the quantum industry. This is where standardization comes into view, a process that is already ongoing in JTC22/WG3 [7]. Such a consensus view will open up a global market and a mature supply chain of modules from a variety of vendors. This approach will benefit vendors by enabling them to sell their products in larger volumes to a broader audience, as standardized interfaces ensure compatibility across a wider range of designs. It is also beneficial for customers, such as research teams and system integrators, since they will not suffer from vendor lock-in. If a new product outperforms an older one from another vendor, upgrading the system will be much simpler, as it can be done without requiring the disinvestment of the remaining components. In general, the approach adopted in this paper aims to avoid monolithic solutions, as they often hinder modular approaches, enabling flexibility and interoperability for vendors.

As this work is formalized within JTC22/WG3 [7], quantum networks are considered out of scope. Although not explicitly mentioned, our current implementation inherently supports the application of quantum computers interconnected via quantum networks. A functional quantum network requires both a physical link, facilitated by communication qubits, and a classical network to manage overhead communication [53]. The classical network, responsible for handling request overhead in distributed quantum computing, would interact with the stack through the communication unit as part of its functionality.

In the context of a cryogenic solid-state quantum computer, the quantum device layer, positioned at the bottom of the hardware stack, handles communication qubits as an integrated functionality. This means that a quantum network interface could be

achieved through additional functionalities within the existing layers without requiring modifications to the overall layers.

7 Conclusion

Modularity played a key role in the growth of the personal computer market. Similarly, organizing quantum computing functionalities into distinct layers, as proposed in this model, will streamline interoperability between products from different vendors. This approach will not only accelerate the advancement of quantum computers but also foster a global market for quantum technologies. Further research should focus on functionalities within each layer, the information flow between layers, and the inclusion of more hardware architectures. Once this is sorted out, dedicated standards for single or multiple layers and their interfacing should be developed within standardization bodies such as JTC22/WG3 [7]. A work item dedicated to cryogenic solid-state quantum computers has already been started within [7]. Future quantum computers will provide access to individual end-users via the cloud, as well as integration with hybrid computing systems. Different users and systems will require access to specific layers; some will primarily need access at lower levels, while others will only require access at higher levels. This makes the communication unit essential, as it must ensure authorized access to the appropriate layers while delivering a seamless experience, making users feel as though they have dedicated resources.

Acknowledgements The development of this layer model is related to part of the activities the authors are carrying out within CEN-CENELEC's JTC22/WG3 [7]. The result is based on comments and contributions from NSB-s/NCs, industrial partners, experts, and academia. Written contributions to JTC22/WG3 on this work item have been submitted by Rob van den Brink (Delft Circuits, project leader), Michele Amoretti (CINI, University of Parma), Juan Boschero (TNO), Michael Fellner (Parity QC), Christian Ertler (Parity QC), Andreas Spoerl (DLR), Oskar van Deventer (TNO), Niels Neumann (TNO), Angie Qarry (QDeepTech), Jean-Baptiste Latre, Blaise Vignon (Alice&Bob), and JL Henry.

Author Contributions All authors have equally contributed to the content and revision of the document.

Data Availability No datasets were generated or analyzed during the current study.

Declarations

Conflict of interest The authors declare no Conflict of interest.

References

1. Rad, H.A., Ainsworth, T., Alexander, R.N., Altieri, B., Askarani, M.F., Baby, R., Banchi, L., Baragiola, B.Q., Bourassa, J.E., Chadwick, R.S., Charania, I., Chen, H., Collins, M.J., Contu, P., D'Arcy, N., Dauphinais, G., Prins, R.D., Deschenes, D., Luch, I.D., Duque, S., Edke, P., Fayer, S.E., Ferracin, S., Ferretti, H., Zhang, Y.: Scaling and networking a modular photonic quantum computer. Nature **638**, 912–919 (2025). https://doi.org/10.1038/s41586-024-08406-9
2. Wright, K., Beck, K.M., Debnath, S., Amini, J.M., Nam, Y., Grzesiak, N., Chen, J.-S., Pisenti, N.C., Chmielewski, M., Collins, C., Hudek, K.M., Mizrahi, J., Wong-Campos, J.D., Allen, S., Apisdorf, J., Solomon, P., Williams, M., Ducore, A.M., Blinov, A., Kreikemeier, S.M., Chaplin, V., Keesan, M., Monroe, C., Kim, J.: Benchmarking an 11-qubit quantum computer. Nat. Commun. **10**, 5464 (2019). https://doi.org/10.1038/s41467-019-13534-2
3. Gold, A., Paquette, J.P., Stockklauser, A., Reagor, M.J., Alam, M.S., Bestwick, A., Didier, N., Nersisyan, A., Oruc, F., Razavi, A., Scharmann, B., Sete, E.A., Sur, B., Venturelli, D., Winkleblack, C.J., Wudarski, F., Harburn, M., Rigetti, C.: Entanglement across separate silicon dies in a modular superconducting qubit device. npj Quantum Information. **7**, 142 (2021). https://doi.org/10.1038/s41534-021-00484-1
4. P. Murali, N. M. Linke, M. Martonosi, A. J. Abhari, N. H. Nguyen, and C. H. Alderete, 2019 Full-stack, real-system quantum computer studies: architectural comparisons and design insights, in Proceedings of the 46th International Symposium on Computer Architecture, ser. ISCA '19. New York, NY, USA: Association for Computing Machinery. p. 527–540. [Online]. Available: doi: https://doi.org/10.1145/3307650.3322273
5. Ball, H., Biercuk, M.J., Hush, M.R.: Quantum firmware and the quantum computing stack. Phys. Today **74**(3), 28–34 (2021)
6. Shehata, A., Groszkowski, P., Naughton, T., Gopalakrishnan Meena, M., Wong, E., Claudino, D., Ferreira da Silva, R., Beck, T.: Bridging paradigms: Designing for hpc-quantum convergence. Futur. Gener. Comput. Syst. **174**, 107980 (2025). https://doi.org/10.1016/j.future.2025.107980
7. CEN-CENELEC, JTC 22 Quantum Technologies, https://www.cencenelec.eu/areas-of-work/cen-cenelec-topics/quantum-technologies/, 2025
8. CEN-CENELEC, CEN/CLC/TR 18202:2025, "Layer model of Quantum Computing", project JT022004, published on 2025-09-03.
9. van Deventer, O., Spethmann, N., Loeffler, M., Amoretti, M., van den Brink, R., Bruno, N., Comi, P., Farrugia, N., Gramegna, M., Jenet, A., Kassenberg, B., Kozlowski, W., Länger, T., Lindstrom, T., Martin, V., Neumann, N., Papadopoulos, H., Pascazio, S., Peev, M., Pitwon, R., Rol, M.A., Traina, P., Venderbosch, P., Wilhelm-Mauch, F.K.: Towards european standards for quantum technologies. EPJ Quantum Technol. **9**(1), 33 (2022)
10. International Organization for Standardization, Catalogue of iso/tc 307 blockchain and distributed ledger technologies standards, https://www.iso.org/committee/10138914/x/catalogue/p/0/u/1/w/0/d/0, 2025, accessed: 2025–10-21
11. ISO, Information technology — quantum computing — vocabulary, International Organization for Standardization and International Electrotechnical Commission, Geneva, Switzerland, Standard ISO/IEC 4879:2024, 2024, accessed: 2025–10-21. [Online]. Available: https://www.iso.org/standard/80432.html
12. IEEE Standards Association, Ieee sa standards board approvals – 19 June 2025, https://standards.ieee.org/about/sasb/sba/19jun2025/, 2025, accessed: 2025–10-21. [Online]. Available: https://standards.ieee.org/about/sasb/sba/19jun2025/
13. N. C. Jones, R. Van Meter, A. G. Fowler, P. L. McMahon, J. Kim, T. D. Ladd, and Y. Yamamoto, Layered architecture for quantum computing, Physical Review X, vol. 2, no. 3, Jul. 2012. [Online]. Available: doi: 10.1103/PhysRevX.2.031007
14. Geck, L., Kruth, A., Bluhm, H., Waasen, S.V., Heinen, S.: Control electronics for semiconductor spin qubits. Quantum Sci. Technol. **5**(1), 015004 (2019). https://doi.org/10.1088/2058-9565/ab5e07
15. X. Fu, L. Riesebos, L. Lao, C. G. Almudever, F. Sebastiano, R. Versluis, E. Charbon, and K. Bertels, A heterogeneous quantum computer architecture, in Proceedings of the ACM International Conference on Computing Frontiers, ser. CF '16, 2016, p. 323–330
16. M. Bandic, S. Feld, and C. G. Almudever, Full-stack quantum computing systems in the nisq era: algorithm-driven and hardware-aware compilation techniques, 2022

17. C. D. Donne, M. Iuliano, B. van der Vecht, G. M. Ferreira, H. Jirovská, T. van der Steenhoven, A. Dahlberg, M. Skrzypczyk, D. Fioretto, M. Teller, P. Filippov, A. R.-P. Montblanch, J. Fischer, B. van Ommen, N. Demetriou, D. Leichtle, L. Music, H. Ollivier, I. te Raa, W. Kozlowski, T. Taminiau, P. Pawe?czak, T. Northup, R. Hanson, and S. Wehner, Design and demonstration of an operating system for executing applications on quantum network nodes, 2024. [Online]. Available: arxiv:2407.18306
18. A. Dahlberg, M. Skrzypczyk, T. Coopmans, L. Wubben, F. Rozpundefineddek, M. Pompili, A. Stolk, P. Pawełczak, R. Knegjens, J. de Oliveira Filho, R. Hanson, and S. Wehner, A link layer protocol for quantum networks, in Proceedings of the ACM Special Interest Group on Data Communication, ser. SIGCOMM '19, 2019, p. 159–173
19. Brunner, N., Cavalcanti, D., Pironio, S., Scarani, V., Wehner, S.: Bell nonlocality. Rev. Mod. Phys. **86**, 419–478 (2014). https://doi.org/10.1103/RevModPhys.86.419
20. Amoretti, M., Carretta, S.: Entanglement verification in quantum networks with tampered nodes. IEEE J. Sel. Areas Commun. **38**(3), 598–604 (2020)
21. Stirbu, V., Kinanen, O., Haghparast, M., Mikkonen, T., Qubernetes,: Towards a unified cloud-native execution platform for hybrid classic-quantum computing. Inf. Softw. Technol. **175**, 107529 . https://doi.org/10.1016/j.infsof.2024.107529
22. J. M. Murillo, J. Garcia-Alonso, E. Moguel, J. Barzen, F. Leymann, S. Ali, T. Yue, P. Arcaini, R. Pérez-Castillo, I. García-Rodríguez de Guzmán, M. Piattini, A. Ruiz-Cortés, A. Brogi, J. Zhao, A. Miranskyy, and M. Wimmer, Quantum software engineering: Roadmap and challenges ahead, ACM Trans. Softw. Eng. Methodol., vol. 34, no. 5, May 2025. [Online]. Available: doi: 10.1145/3712002
23. Kjaergaard, M., Schwartz, M.E., Braumüller, J., Krantz, P., Wang, J.I.-J., Gustavsson, S., Oliver, W.D.: Superconducting qubits: Current state of play. Ann. Rev. Condens. Matter Phys. **11**, 369–395 (2020)
24. Kim, S., Abdurakhimov, L.V., Pham, D., Qiu, W., Terai, H., Ashhab, S., Saito, S., Yamashita, T., Semba, K.: Superconducting flux qubit with ferromagnetic josephson π-junction operating at zero magnetic field. Commun. Mater. **5**(1), 216 (2024)
25. Burkard, G., Ladd, T.D., Pan, A., Nichol, J.M., Petta, J.R.: Semiconductor spin qubits. Rev. Mod. Phys. **95**, 025003 (2023)
26. M. Iqbal, A. Lyons, C. F. B. Lo, N. Tantivasadakarn, J. Dreiling, C. Foltz, T. M. Gatterman, D. Gresh, N. Hewitt, C. A. Holliman, J. Johansen, B. Neyenhuis, Y. Matsuoka, M. Mills, S. A. Moses, P. Siegfried, A. Vishwanath, R. Verresen, and H. Dreyer, Qutrit toric code and parafermions in trapped ions, 2024. [Online]. Available: arxiv:2411.04185
27. van Dam, J., Avis, G., Propp, T., Ferreira da Silva, F., Slater, J., Northup, T., Wehner, S.: Hardware requirements for trapped-ion-based verifiable blind quantum computing with a measurement-only client. Quantum Sci. Technol. **9**, 045031 (2024)
28. Stolk, A.J., van der Enden, K.L., Slater, M.-C., te Raa-Derckx, I., Botma, P., van Rantwijk, J., Biemond, J.J.B., Hagen, R.A.J., Herfst, R.W., Koek, W.D., Meskers, A.J.H., Vollmer, R., van Zwet, E.J., Markham, M., Edmonds, A.M., Geus, J.F., Elsen, F., Jungbluth, B., Haefner, C., Tresp, C., Stuhler, J., Ritter, S., Hanson, R.: Metropolitan-scale heralded entanglement of solid-state qubits. Sci. Adv. **10**(44), eadp6442 (2024)
29. Gaita-Ariño, A., Luis, F., Hill, S., Coronado, E.: Molecular spins for quantum computation. Nat. Chem. **11**(4), 301–309 (2019)
30. Chiesa, A., Santini, P., Garlatti, E., Luis, F., Carretta, S.: Molecular nanomagnets: a viable path toward quantum information processing? Rep. Prog. Phys. **87**, 034501 (2024)
31. Gulka, M., Wirtitsch, D., Ivády, V., Vodnik, J., Hruby, J., Magchiels, G., Bourgeois, E., Gali, A., Trupke, M., Nesladek, M.: Room-temperature control and electrical readout of individual nitrogen-vacancy nuclear spins. Nat. Commun. **12**(1), 4421 (2021)
32. Vallabhapurapu, H.H., Hansen, I., Adambukulam, C., Stöhr, R., Denisenko, A., Yang, C.H., Laucht, A.: High-fidelity control of a nitrogen-vacancy-center spin qubit at room temperature using the sinusoidally modulated, always rotating, and tailored protocol. Phys. Rev. A **108**, 022606 (2023)
33. Bruzewicz, C.D., Chiaverini, J., McConnell, R., Sage, J.M.: Trapped-ion quantum computing: Progress and challenges. Appl. Phys. Rev. **6**(2), 021314 (2019)
34. Ringbauer, M., Hinsche, M., Feldker, T., Faehrmann, P.K., Bermejo-Vega, J., Edmunds, C.L., Postler, L., Stricker, R., Marciniak, C.D., Meth, M., Pogorelov, I., Blatt, R., Schindler, P., Eisert, J., Monz, T., Hangleiter, D.: Verifiable measurement-based quantum random sampling with trapped ions. Nat. Commun. **16**(1), 106 (2025). https://doi.org/10.1038/s41467-024-55342-3

35. Ritter, S., Nölleke, C., Hahn, C., Reiserer, A., Neuzner, A., Uphoff, M., Mücke, M., Figueroa, E., Bochmann, J., Rempe, G.: An elementary quantum network of single atoms in optical cavities. Nature **484**(7393), 195–200 (2012)
36. Hartung, L., Seubert, M., Welte, S., Distante, E., Rempe, G.: A quantum-network register assembled with optical tweezers in an optical cavity. Science **385**(6705), 179–183 (2024)
37. Wang, J., Sciarrino, F., Laing, A., Thompson, M.G.: Integrated photonic quantum technologies. Nat. Photonics **14**(5), 273–284 (2020)
38. Maring, N., Fyrillas, A., Pont, M., Ivanov, E., Stepanov, P., Margaria, N., Hease, W., Pishchagin, A., Lemaître, A., Sagnes, I., Au, T.H., Boissier, S., Bertasi, E., Baert, A., Valdivia, M., Billard, M., Acar, O., Brieussel, A., Mezher, R., Wein, S.C., Salavrakos, A., Sinnott, P., Fioretto, D.A., Emeriau, P.-E., Belabas, N., Mansfield, S., Senellart, P., Senellart, J., Somaschi, N.: A versatile single-photon-based quantum computing platform. Nat. Photonics **18**(6), 603–609 (2024)
39. Terhal, B.M.: Quantum error correction for quantum memories. Rev. Mod. Phys. **87**, 307–346 (2015)
40. Campbell, E.: A series of fast-paced advances in quantum error correction. Nature Rev. Phys. **6**(3), 160–161 (2024)
41. D. C. McKay, T. Alexander, L. Bello, M. J. Biercuk, L. Bishop, J. Chen, J. M. Chow, A. D. Córcoles, D. Egger, S. Filipp, J. Gomez, M. Hush, A. Javadi-Abhari, D. Moreda, P. Nation, B. Paulovicks, E. Winston, C. J. Wood, J. Wootton, and J. M. Gambetta, Qiskit backend specifications for openqasm and openpulse experiments, 2018. [Online]. Available: arxiv:1809.03452
42. Silvério, H., Grijalva, S., Dalyac, C., Leclerc, L., Karalekas, P.J., Shammah, N., Beji, M., Henry, L.-P., Henriet, L.: Pulser: An open-source package for the design of pulse sequences in programmable neutral-atom arrays. Quantum **6**, 629 (2022). https://doi.org/10.22331/q-2022-01-24-629
43. Cross, A., Javadi-Abhari, A., Alexander, T., De Beaudrap, N., Bishop, L.S., Heidel, S., Ryan, C.A., Sivarajah, P., Smolin, J., Gambetta, J.M., Johnson, B.R.: Openqasm 3: A broader and deeper quantum assembly language. ACM Trans. Quantum Comput. **3**(3), 1–50 (2022). https://doi.org/10.1145/3505636
44. Haileytap, Introduction to the quantum programming language q# - azure quantum, 2025. [Online]. Available: https://learn.microsoft.com/en-us/azure/quantum/qsharp-overview
45. V. Bezganovic, M. Lewis, S. Soudjani, P. Zuliani, High-level quantum algorithm programming using silq, 2024. [Online]. Available: https://arxiv.org/abs/2409.10231
46. Green, A.S., Lumsdaine, P.L., Ross, N.J., Selinger, P., Valiron, B.: Quipper: a scalable quantum programming language. ACM SIGPLAN Notices. **48**(6), 333–342 (2013). https://doi.org/10.1145/2499370.2462177
47. A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson, J. M. Gambetta, Quantum computing with qiskit, 2024. [Online]. Available: https://arxiv.org/abs/2405.08810
48. C. Developers, Cirq. Zenodo, May 2024. [Online]. Available: https://zenodo.org/doi/10.5281/zenodo.4062499
49. Ferrari, D., Tavernelli, I., Amoretti, M.: Deterministic algorithms for compiling quantum circuits with recurrent patterns. Quantum Inf. Process. **20**(6), 213 (2021). https://doi.org/10.1007/s11128-021-03150-9
50. D. Ferrari and M. Amoretti, Noise-adaptive quantum compilation strategies evaluated with application-motivated benchmarks, in Proceedings of the 19th ACM International Conference on Computing Frontiers, 2022, p. 237–243
51. M. G. Pozzi, S. J. Herbert, A. Sengupta, and R. D. Mullins, Using Reinforcement Learning to Perform Qubit Routing in Quantum Compilers, ACM Transactions on Quantum Computing, no. 2, May 2022
52. M. B. Healy, R. Jokar, S. Thomas, V. R. Pascuzzi, K. Barton, T. A. Alexander, R. Elkabetz, B. C. Donovan, H. Horii, and M. Hillenbrand, Design and architecture of the ibm quantum engine compiler, in 2024 IEEE International Conference on Quantum Computing and Engineering (QCE), vol. 01, 2024, pp. 866–872
53. J. C. Boschero, N. M. P. Neumann, W. van der Schoot, T. Sijpesteijn, and R. Wezeman, Distributed quantum computing: Applications and challenges, 2024. [Online]. Available: https://arxiv.org/abs/2410.00609