Teaching Quantum Design Automation with Block-Based Programming

Damian Rovara* Robert Wille*†
*Technical University of Munich, Germany
†MQSC, Garching near Munich, Germany
damian.rovara@tum.de robert.wille@tum.de
https://www.cda.cit.tum.de/research/quantum

***Abstract*—As quantum circuits grow beyond small toy examples, preparing them for execution on physical devices becomes increasingly complex. Design automation is therefore essential for scalable quantum computing: *Compilation procedures* optimize resource requirements and transform circuits to a format compatible with specific hardware; *resource estimation* evaluates execution cost; *verification methods* prove circuit correctness. However, these concepts present a steep learning curve for novices, particularly when quantum circuits are introduced through low-level textual representations. To address this, we present a block-based programming framework for quantum design automation, implemented as an extension to the *Scratch* programming platform. This system allows users to build quantum circuits as a sequence of blocks and embed them in classical control logic to perform evaluations, compare simulation results, and directly apply different design automation techniques. We evaluated the approach in a user study with computer science students, who completed guided exercises using the platform and provided structured feedback in the form of self-reports and short knowledge assessments. Results demonstrate strong understanding and confidence in quantum design automation concepts, suggesting that the block-based approach successfully lowers the entry barrier to quantum design automation. The implemented framework is open-source and available at https://github.com/munich-quantum-toolkit/scratch-quantum.**

***Index Terms*—quantum design automation, block-based programming, verification, resource estimation, languages**

I. Introduction

Quantum computing is a rapidly growing field of research, built on a fundamental basis of interdisciplinarity. Developers wanting to implement quantum algorithms require not only a strong understanding of the underlying quantum-mechanical principles. They also need substantial knowledge in software development to understand the computational concepts that are necessary to build executable quantum programs.

However, at the same time, the steady improvements of real quantum hardware, such as the continuous reduction of gate errors and increases in qubit counts, also lead to more complex quantum programs. This makes a fully manual development workflow error-prone and inefficient. Preparing larger circuits for the execution on actual hardware, optimizing their resource requirements, or verifying their correctness present just some of many computationally hard tasks that are practically infeasible for manual processing. When faced with the same problem in the classical world, circuit developers proposed *design automation* methods that take over this manual workflow [1].

Unfortunately, the interdisciplinarity of quantum computing renders the adoption of these techniques difficult: Many quantum computing experts do not have the necessary background in design automation, while many classical design automation researchers lack the required quantum-mechanical fundament.

While several educational tools have been designed to ease the access for non-experts into the quantum computing world, there exists a distinct boundary between those tools and the potential applications in quantum design automation. Graphical circuit editors, for instance, allow users to visually learn the effects of individual gates on underlying quantum states [2, 3]. However, these circuits are limited in their compatibility with other software tools and lack abstraction structures such as classical control flow, breaking the pedagogical link to actual software engineering. Textual representations for quantum programs, such as quantum programming frameworks [4–7] and intermediate representations (IRs, [8–12]), on the other hand, may address these limitation, at the cost of being more difficult to grasp for new learners.

In this work, we aim to bridge this gap by introducing a block-based framework for the development of quantum circuits as an extension to the *Scratch* [13, 14] platform. This approach leverages the proven educational value of block-based development, while providing an easy integration of quantum and classical concepts through the pre-existing classical instructions provided by *Scratch*. Additionally, through a closely coupled translation to other IRs, we not only provide a stronger compatibility with the myriad of existing tools for quantum design automation, but also allow new learners to naturally mature into the larger quantum computing ecosystem.

An exploratory evaluation with computer science and electrical engineering students indicates that the tool lowers the entry barrier to quantum design automation concepts while maintaining high usability. The proposed framework is fully open-source; its code is available at https://github.com/munich-quantum-toolkit/scratch-quantum as part of the *Munich Quantum Toolkit* (MQV, [15]), with a web UI accessible at https://munich-quantum-toolkit.github.io/scratch-quantum/.

The remainder of this work is structured as follows: Section II provides the required background in quantum design automation. Section III then presents in more detail the limitations of existing software tools to teach quantum design automation concepts and demonstrates how a block-based framework can address these limitations. Then, Section IV presents the core functionalities of the proposed framework, and Section V explores how they can be used to teach quantum design automation concepts to new learners. Section VI then performs an empirical evaluation of the proposed framework and, finally, Section VII concludes this work.

II. Background

This section presents the fundamental concepts of quantum design automation and previous related work in the field.

A. Design Automation in Quantum Computing

As quantum algorithms grow in size, the manual preparation and evaluation of circuits is slowly becoming unfeasible even for experts in the field. A similar trend in the classical circuit design community has previously led to the emergence of various electronic design automation methods, such as for circuit synthesis or verification [1, 16–19]. As a result, design automation methods for quantum computing have also risen in popularity over the recent years. In the following, we explore the core research areas of quantum design automation.

Compilation: Execution time on quantum hardware is expensive and longer circuit executions lead to a higher degree of noise, both due to qubit decoherence and gate errors. Therefore, it is crucial to compile quantum circuits in an efficient way. This is especially important when considering restrictive hardware properties, such as limited gate sets or qubit connectivities, that require input circuits to be transformed to fit the device characteristics.

Frameworks such as *Qiskit* [6] or *TKET* [20] are commonly employed software tools that aid in the compilation of quantum circuits. They employ pattern rewrite rules to efficiently prune down circuits and prepare them for execution on quantum hardware. At the same time, approaches based on machine learning [21–25] are also gaining popularity, taking advantage of massive datasets of quantum circuits that are available on different platforms [26, 27] to train models to produce minimal representations of quantum circuits.

As new hardware types with different constraints—such as differing coupling maps or even entirely new qubit modalities—are being developed, compilation tools must be adapted and improved to tackle the relating challenges [28–30]. Furthermore, an increasing shift towards a hybrid ecosystem in which quantum and classical operations are required to be executed in an interleaved manner requires compilation toolchains that can handle these complex workflows.

Resource Estimation: Quantum hardware of the *near-term intermediate scale quantum* (NISQ) era is limited both in accuracy and qubit count. While smaller circuits can be executed on this hardware, larger problem instances quickly reach the limits of current devices. Similarly, state-of-the-art simulators [31–35] also struggle with the execution of arbitrary quantum circuits of larger sizes. As a result, reasoning over large circuits becomes challenging.

To still be able to evaluate large quantum circuits, *resource estimation* tools are commonly employed. They analyze circuits in terms of their resource requirements and yield results that can be used to determine the performance of optimizations or compare hardware vendor roadmaps to estimate at what time we can expect advanced quantum computers to execute these circuits [36, 37]. Resource estimation tools often also investigate further metrics, such as accuracy estimates [38] or approximate costs for the fault-tolerant execution of the circuit [39].

Verification: Methods for verification play an important role in evaluating the correctness of complex quantum circuits. Quantum Hoare logic [40], for instance, provides a fundamental framework for the formal verification of circuits, extending classical verification rules to the quantum domain. Assertions are furthermore being employed for the verification and testing of quantum circuits [41–49], allowing circuit designers to specify the expected quantum state during the execution and comparing it with execution outcomes.

Finally, quantum circuit equivalence checking has been commonly employed, especially to verify the correctness of circuit transformations [50, 51]. As empirical evaluations have shown the prevalence of bugs in quantum circuits, verification and testing is becoming an increasingly important aspect of quantum design automation [52–56].

As the design and evaluation of quantum circuits is a conceptually difficult topic, educational tools require visual aids to provide an understanding of the underlying concepts. The following evaluates related software tools in the field, investigating their use cases.

B. Software Tools for Quantum Design Automation

With the growing need for design automation tools in quantum computing, several larger software tools have found wide-spread adoption. The *Qiskit* framework [6], for instance, can be used to construct quantum circuits, to optimize and analyse them, as well as to execute them through simulators or on physical devices. Similarly, *TKET* [20] is another popular software tool for the compilation of quantum circuits. The *Munich Quantum Toolkit* (MQT, [15]) is another set of design automation software tools for quantum circuits, ranging from high-level application scenarios [57, 58] to device-specific and device-agnostic compilation passes [59–61].

The *Microsoft Azure Quantum Development Kit* [62] provides a programming environment for the design of quantum circuits, as well as several tools for debugging and testing. Through its quantum resource estimator [5, 39], it also provides users with the ability to evaluate the resource requirements for the fault-tolerant execution of quantum circuits.

All of these tools are compatible with a variety of low-level quantum programming languages, such as OpenQASM [63]. At the same time, other compilation frameworks are instead building on classical compilation infrastructure, predominantly based on or inspired by the *Multi-Level Intermediate Representation* (MLIR, [64]) framework [8, 9, 11, 12]. Further higher-level quantum development kits, such as *PennyLane* [5], *Qrisp* [65], or *Guppy* [4], build on top of these quantum-classical representations.

III. Motivation

This section highlights core challenges that emerge when teaching quantum design automation concepts. It then proposes a new approach to more easily and intuitively teach these concepts to students.

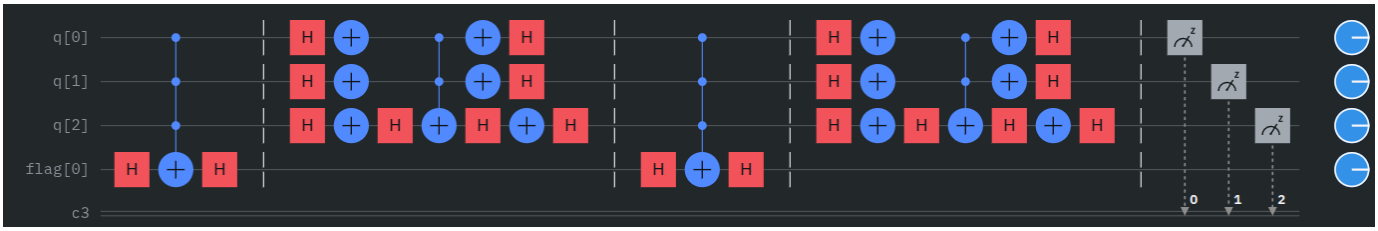


Figure 1. Implementing Grover's algorithm using IBM's *Quantum Composer* after unrolling the loop into a single circuit.

A. Challenges in Teaching Quantum Computing

Textual code representations for quantum circuits, such as *OpenQASM* [63], *QIR* [66], or *MLIR* [64] find a lot of applications in quantum design automation [12, 67]. However, reading the resulting code files and gaining an intuitive understanding of their inner workings poses a challenge even for experts with years of experience in the field. Asking new learners to start with these verbose and complex representations would greatly hinder their process, which is why visual quantum circuit editors have risen in popularity over the last years. Tools such as *Quirk* [7, 68] or IBM's *Quantum Composer* [2, 3] allow learners to easily get acquainted with quantum circuits by providing a drag and drop interface in which users can add gates to a circuit and, through integrated simulation methods, view how changes to the circuit affect the resulting state.

However, as quantum computing research has grown into a much broader interdisciplinary field—incorporating ideas such as HPC integration or fault tolerance—circuit editors have started to demonstrate several shortcomings that limit the degree to which they can cover the quantum computing ecosystem.

Notably, the introduction of *quantum-classical interactions* in quantum computing workflows slowly leads to a deviation from typical circuits. Dynamic measurements are starting to be more commonly included in the standard repertoire of supported operations [69–72]. Even further, other control flow features, such as *loops* and *function calls*, are completely incompatible with the current visual circuit representation methods.

Furthermore, while visual circuit editors might be a useful pedagogical stepping stone for circuit design methods, their integration with other design automation tools is often very loose. IBM's *Quantum Composer*, for instance, allows the export to and import from *OpenQASM* programs. However, to run and evaluate transformations on the generated circuits, users are required to move away from the visual editor. This boundary between circuit design and other design automation tasks renders the topics very remote and hinders the natural integration of the ideas behind them.

Finally, as the landscape of automated tools in the field of quantum computing grows wider, so does the required compatibility of generated quantum programs. While *OpenQASM* is commonly employed, many software frameworks also build upon *QIR*, *MLIR*, or other intermediate representations [4, 8–12]. To accelerate the education process of new learners and maximize the tools they have at their disposal, quantum program design tools require compatibility with as many different representations as possible.

Example 1. *Consider teaching Grover's algorithm [73] via a standard visual circuit editor. As the algorithm relies on the repeated application of* oracle *and* diffusion *steps, the use of loops would provide an intuitive understanding of this idea to students with a computer science background. However, because existing tools typically lack classical control structures, the algorithm must be manually unrolled, stripping away a crucial layer of algorithmic abstraction. This is illustrated in Fig. 1 for just two loop iterations. Most of the visualization is taken up by two identical circuit segments. An implementation with more iterations will lead to even larger redundancies in the visualization, substantially harming the readability. Furthermore, demonstrating the algorithm's flexibility for different search targets requires the student to manually rearrange the underlying circuit for every new search value. Lastly, if we then want to analyze the resource costs of these different variations, or investigate potential optimizations to the circuit, it must be repeatedly exported to external tools (e.g., via OpenQASM). This creates a disjointed workflow severely hindered by constant, repetitive manual export cycles.*

This example shows that, to fully facilitate the learning process of developers interested in quantum computing, we need to provide a wider playground that makes development easy while providing the maximum degree of compatibility with design automation methods.

B. Proposed Framework

To address the issues and limitations illustrated above, we propose adopting a proven methodology from classical computer science education. Specifically, we introduce a framework that extends *Scratch*—a widely used block-based programming platform that provides a drag-and-drop environment in which users can build programs by connecting individual instruction blocks [13, 14]. While preliminary efforts have previously explored introducing quantum concepts into Scratch [74, 75], these approaches largely focus on higher-level quantum mechanical concepts, and lack the underlying architectural infrastructure required for quantum design automation. As an extension to *Scratch*, the proposed framework can take advantage of a wide variety of classical programming concepts, provided in a beginner-friendly format to give new students easier access [76]. In particular, this also includes classical control flow such as loops, conditionals, and function calls.

The proposed extension provides a set of quantum computing instructions, such as for the allocation of qubits and application of gates and measurements. This allows users to construct quan-

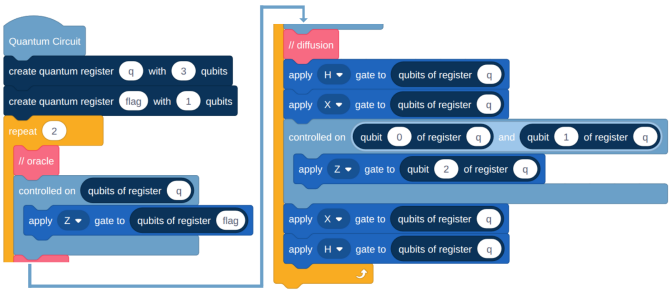


Figure 2. Implementing Grover's algorithm inside the proposed *Scratch* extension.

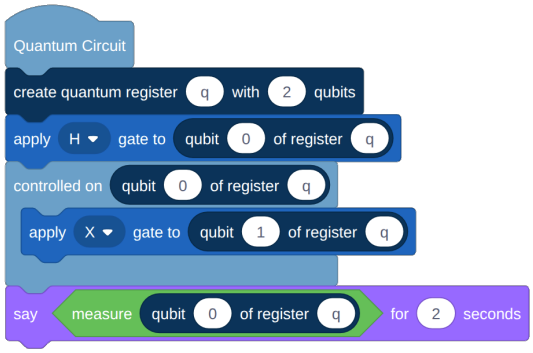


Figure 3. A Bell circuit created through the *Scratch* extension.

tum programs the same way they would using a visual circuit editor while also having full access to classical computation.

Furthermore, the extension allows the simulation of the constructed circuits with a modular simulation backend, which enables easy extensibility with different simulator types. In addition, further analysis tools, such as different *resource estimation* methods, are directly integrated in the framework, so that no interaction with external tools is strictly required. On the other hand, if interoperability with other tools and frameworks is desired, the proposed framework also supports the export of constructed circuits into an MLIR-based format which allows the translation to commonly used formats such as *OpenQASM* and *QIR*, as well as to the `jeff` exchange format [77], through which compatibility with a variety of quantum program compilation and execution pipelines is enabled [4, 8, 9, 11, 12]. This allows learners to naturally grow into the mature quantum computing ecosystem.

Example 2. *Revisiting the Grover's algorithm scenario from Example 1, the proposed block-based framework eliminates the issues. As* Scratch *supports classical control flow structures, the loop that had to be unrolled for circuit editors can instead be applied intuitively. An example implementation of the circuit is illustrated in Fig. 2. As can clearly be seen, the algorithm itself is only defined once, while the iterations themselves are covered by the "*repeat*" block. Similarly, classical instructions and conditionals can be used to execute the algorithm for various inputs, allowing users to search for different values with only minimal changes to the program. Finally, as resource estimation is directly integrated into the system, no export to other frameworks is required. Resource estimation is simply performed through the switch of a single instruction, changing between "*simulation*" and "*estimation*" mode. This way, even the evaluation of different versions of the program can always be performed with just a single press of the "*play*" button.*

IV. A Quantum Extension for Scratch

This section describes the technical details of the proposed framework, including the usage of the different features it supports, as well as how they are implemented. The implementation of the framework is open-source and a live version is accessible at https://munich-quantum-toolkit.github.io/scratch-quantum/.

A. Quantum Circuits in the Scratch Extension

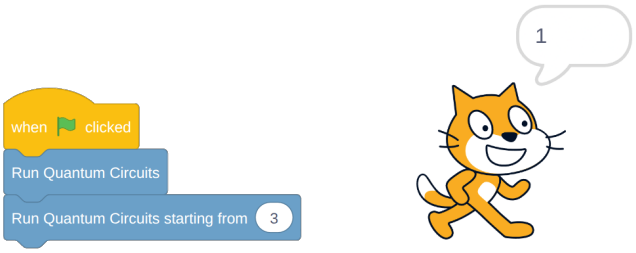
Many of the issues listed above, such as the limited classical methods available in circuit simulators, are already addressed by the strong foundation provided by the *Scratch* framework. It provides instructions to create and modify variables, control the execution flow, or even interact with the user. Therefore, the proposed extension only needs to implement instructions for the actual creation of quantum circuits. This includes:

- An instruction to *allocate named registers* of qubits with arbitrary sizes.
- Instructions to *apply different gates* to a provided circuit (including parametrized rotational gates).
- *Control modifier blocks* that can take on child instructions that will be controlled by the given qubit, similarly to how other higher-level languages implement control modifiers [12, 63, 65]. Using further "*negated control*" and "*multi-control*" blocks also allows more complex controlled gates.
- A "*measurement*" block that returns a classical value which can be easily integrated into the classical execution environment of the standard *Scratch* instructions.

Quantum operations can only be placed below a "*Quantum Circuit*" parent block. This allows users to implement several independent quantum circuits and compare them more easily. All quantum operations allow variable inputs to be specified with dynamic attributes, such as variables or computation results. This allows a simple and direct interaction between classical computations (e.g., to decide target qubits for operations) and the quantum circuit.

Example 3. *Fig. 3 illustrates the preparation of a Bell state using the proposed framework. It starts with the allocation of a qubit register consisting of two qubits. It then applies a Hadamard gate to the qubit at index 0 (q_0), followed by a* `CX` *gate controlled on q_0. Finally, it performs a measurement of q_0 and passes the output to the standard* Scratch *block "*say [text] for [time] seconds*" to display the measurement result.*

With these quantum computing concepts integrated into the *Scratch* development platform, the next important step is to provide support for quantum design automation methods. This is presented next, more precisely through simulation, resource estimation, and further compatibility with more advanced methods.



(a) (b)

Figure 4. Starting the simulation of the circuit in Fig. 3 from the *Scratch* framework and the output of the final simulation.

B. Modular Simulator Backend

The proposed framework allows for the modular application and exchange of different simulation backends to execute the provided circuits. To this end, it defines an interface specifying a set of functions that have to be implemented by a compatible simulator, such as register allocation, gate application, and state measurement. During the execution of the *Scratch* program, the quantum-specific blocks will only access these interface methods of the simulator. By relying on this abstraction, the architecture natively supports the runtime exchange of simulation backends. While the current implementation features only a single state vector simulator, this modular design provides a foundation for future extensions where learners could seamlessly investigate and compare the performance of different simulation methodologies.

To start the simulation process, the "*Run Quantum Circuit*" block can be used. This instruction causes all circuits contained in the current program to be executed in parallel. Simulations can be performed repeatedly, for instance to emulate the evaluation of several shots of a circuit. Furthermore, the "*Run Quantum Circuit starting from [state]*" block can also be used as an alternative. This instruction starts all simulations, but initializes the quantum state with the given value. This allows users to modularly execute quantum circuits from different starting states.

Example 4. *Fig. 4a shows the main entry point of a* Scratch *program, prepared to execute the circuit defined in Fig. 3. The circuit is first executed starting implicitly from the* $|00\rangle$ *state. In a second step, the circuit is instead executed starting from the state* 3*, which is equivalent to the* $|11\rangle$ *state. The corresponding output, which is displayed using the "*say [text] for [time] seconds*" printing interface, is shown in Fig. 4b. In both cases, the measured qubit is located with equal probability in both computational basis states. Accordingly, executing the program multiple times, yields both potential results equally as often.*

At the moment, the default simulator implemented for the proposed framework allows the allocation of up to 10 qubits. Attempting to allocate further qubits safely halts execution to prevent memory exhaustion on the host machine. Analyzing larger algorithms therefore requires alternative methods, which we address through the framework's integrated *resource estimation*.

C. Resource Estimation

By substituting the "*Run Quantum Circuit*" block with a "*Perform Quantum Resource Estimation*" block, provided circuits are instead analyzed for their resource requirements. As the provided circuits may depend on dynamic runtime values and can use control flow instructions such as dynamic loops, the required resources cannot be analyzed statically. Instead, the proposed framework employs a stochastic tracing method [10, 78]. Rather than tracking a full quantum state vector, it only stores an approximate state of individual qubits. Whenever a measurement is encountered, an outcome is chosen randomly based on the information tracked by this method. While this does not always lead to correct measurement outcomes, it generates an approximate execution trace that is often indicative for the normal execution of the provided circuit.

This trace is then used to compute the required number of resources, such as qubit count or gate counts. The estimated resources are then stored in classical variables that can be accessed outside of the circuit to investigate analysis results. This allows learners to immediately access resource estimation results without requiring them to move to a different framework. Furthermore, they can just as easily compare the impact of changes to circuits by modifying them and immediately rerunning the estimation. As the stochastic tracing method does not suffer from the same exponential runtime explosion required for full simulation, this method can be used to reason over larger programs that cannot be simulated.

To provide an overview of different potential cost models for quantum circuits to learners, the proposed framework also allows different *decomposition settings* to be specified. For instance, by enabling *`SWAP` decomposition*, `SWAP` gates are internally replaced by three `CX` gates, as is often required for the compilation of quantum circuits for physical devices. This allows learners to more easily visualize the increased cost of more expensive gates. Furthermore, *Toffoli decomposition* can be enabled to force all multi-controlled gates to instead be decomposed into several single and two-qubit gates. This decomposition employs additional ancilla qubits to reduce the number of operations required for decomposition [79]. This way, learners can experience not just an increase in gate count but also qubit count as they implement circuits with multi-controlled gates. Overall, these decomposition settings allow users to explore the bottlenecks of larger quantum circuits, as well as potential targets for circuit optimizations.

Example 5. *Consider a quantum circuit using 20 qubits that applies a Hadamard gate to 19 of these qubits using a simple loop, and then performs a Toffoli gate using these qubits as controls and targeting the final qubit. While this circuit is too large for full simulation, resource estimation correctly determines that this circuit requires 20 total qubits and uses 19 single-qubit Hadamard gates and one Toffoli gate on 20 qubits, correctly counting the resources even in the presence of a loop structure. If we instead enable Toffoli decomposition and perform the resource estimation again, the estimation yields 38 total qubits with 379 single-qubit gates and 217*

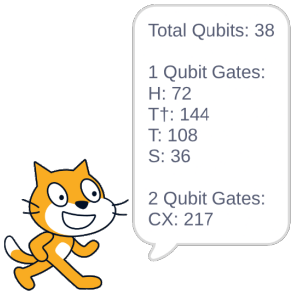


Figure 5. The resource estimates for the *Toffoli decomposition* of a 20-qubit Toffoli gate.

two-qubit gates. The exact output resulting from displaying the resource estimates are shown in Fig. 5. This outcome appropriately highlights the resource costs of Toffoli gates with several control qubits.

D. Compatibility with other Methods

To simplify interoperability with other frameworks, the block "*Export Circuits to MLIR*" can be used to translate a given circuit into an MLIR-based format provided by the *mqt-cc* compilation framework [12]. This framework provides translation methods to many other commonly employed quantum computing tools. Fig. 6 illustrates the different representations that quantum circuits constructed in the proposed framework are compatible with.

Notably, circuits that have been implemented using only quantum instructions can easily be translated to *Qiskit* [6] `QuantumCircuit` objects. This allows the integration with the mature python-based quantum circuit framework. More complex programs, particularly if they involve control flow structures, can instead be translated to the adaptive QIR profile [66]. Furthermore, by going through the `jeff` exchange format [77], they can also be used within various other compilation frameworks, such as *Catalyst* [9] or *TKET* [20]. Through newer models of quantum hardware, such as the *Helios* quantum computer [80], such programs with dynamic structured control flow can even be executed on physical quantum hardware. This wide range of interoperability is core to providing learners with a launching pad to get more closely involved with the quantum computing ecosystem.

V. Exploring Quantum Design Automation with Scratch

Having established the proposed framework's technical architecture, this section demonstrates its pedagogical utility. To illustrate how the extension facilitates the understanding of quantum design automation, we present a guided workflow centered around a single running example illustrated in Fig. 7. This workflow mirrors a realistic engineering lifecycle: moving from initial circuit design and resource estimation to optimization and verification. By stepping through this process, we highlight how the integrated environment allows learners to directly observe the hardware-level consequences of their design choices without relying on disconnected external tools.

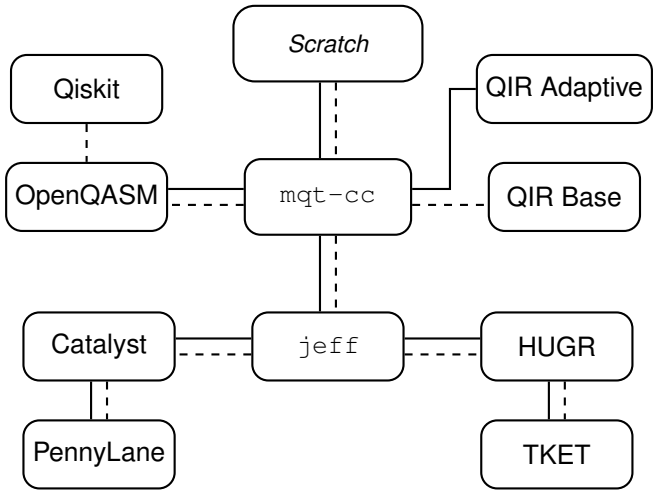


Figure 6. Interactivity of *Scratch-quantum* programs with other frameworks. Dashed lines indicate the translation of purely quantum programs while solid lines also include quantum-classical programs.

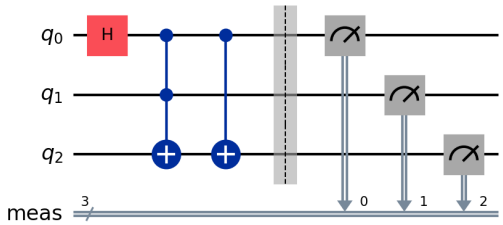
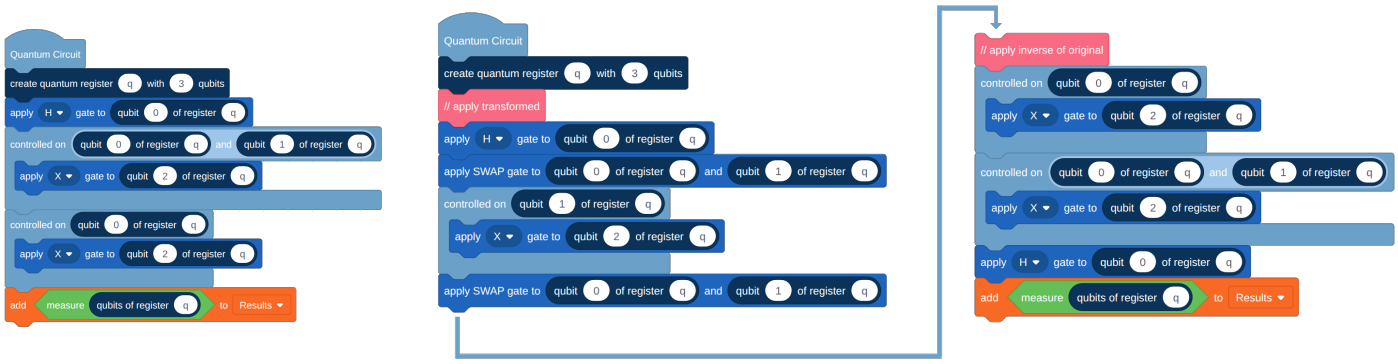


Figure 7. A simple quantum circuit with three qubits, used as a running example for this section

A. Circuit Design

A key advantage of using the proposed framework is its integration with classical computation concepts. This allows learners to not just build quantum circuits, but instead implement classical programs that *contain* quantum circuits. To easily distinguish between the types of instructions, *Scratch* employs a color-coding system. By assigning different shades of blue colors to quantum instructions, we can create a strong visual boundary to classical control flow and variable storage instructions, which instead largely use orange and yellow tones. This enables learners to translate standard circuit diagrams or textual algorithm descriptions into Scratch programs intuitively, easily distinguishing quantum operations from classical logic at a glance.

Example 6. *Fig. 7 illustrates a quantum circuit using 3 qubits. Students that have already been introduced to the concept of quantum circuits can easily be asked to translate this circuit into a* Scratch *program by dragging in the corresponding blocks. The resulting* Scratch *program is shown in Fig. 8a. This example illustrates concepts such as controlled gates and measurements. The shift from the "quantum world" to the "classical world" through measurements, with outcomes stored in a classical list of bits, is clearly visualized with the change in the hue of the blocks, ranging from the blue quantum-related blocks to the orange blocks for data storage, bridged together by the green*



(a) Initial circuit implementation.

(b) Equivalence checking.

Figure 8. A quantum circuit with three qubits, implemented and verified in *Scratch*. (a) Implementing the baseline circuit defined in Fig. 7. All qubits are measured and the outcomes are stored in the classical list Results. (b) Checking the equivalence of the transformed circuit by concatenating it with the inverse of the original. If the Results list contains at least one entry with the value 1, the circuits are not equivalent for the starting state |000>.

measurement bloks. Learners can now be asked to simulate the circuit and view the measurement outcomes directly through the built-in variable inspection system or to work with the measurement outcomes in further classical instructions.

B. Evaluation with Resource Estimation

After constructing the quantum circuit, we next pose the question of how its performance can be evaluated. By giving learners the freedom to modify the circuit, including the addition of further qubits, they may already experience a decrease in responsiveness as circuits grow larger. However, once the limits of the underlying simulator are crossed, this indirect evaluation process will no longer work. At this point, we introduce the concept of resource estimation to address the limitations of simulation-based analysis:

- As simulating the circuit is not required, it can efficiently process circuits of various sizes with hundreds of qubits.
- It focuses on specific metrics that are more meaningful for actual execution that is impacted by differences in errors and execution times of different gates.
- It can specifically address hardware constraints such as limited gate compatibilities.

Resource estimation allows learners to investigate the metrics most relevant to physical hardware constraints. Furthermore, by covering different gate decomposition types, learners can intuitively grasp the cost differences individual gates carry.

Example 7. *Returning to the example program shown in Fig. 8, we ask students to perform resource estimation by replacing the* "Run simulation" *block with a* "Perform Quantum Resource Estimation" *instruction. This process reports a total qubit count and gate count of 3. After enabling* Toffoli decomposition *for this program, the number of required qubits rises to 4 and the number of gates rises to 35. In particular, this includes a total of 14 T and T^{-1} gates, as well as 14 CX gates. Finally, we ask students to add additional control qubits to the Toffoli gate and investigate the changes in resource requirements. This way, learners can independently discover the relation between control counts in Toffoli gates and total qubit, as well as gate counts.*

C. Compilation

While certain quantum circuit compilation tasks, such as mapping operations to a native gate level, were already covered in the previous step on resource estimation, *circuit optimization* still represents an important aspect of compilation. After being given performance evaluation methods in the form of resource metrics, learners can now attempt to improve those metrics by applying changes to the circuit. Furthermore, the concept of qubit routing can be introduced at this stage, using the "*apply `SWAP` gate*" instruction and asking learners to prepare circuits for a given coupling map. By then reevaluating the resource requirements, students can determine how their proposed optimizations, as well as the overhead introduced by inserting the `SWAP` gates, impacted the circuit's performance metrics.

Example 8. *We highlight how the q_1 in Fig. 8 will stay in the $|0\rangle$ state throughout the entire execution of the circuit. As a result, the first Toffoli gate in the circuit effectively has no effect on the overall quantum state, as q_1 is one of its control qubits, and can therefore safely be removed. Further, we introduce a coupling map with nearest neighbor connectivity among individual qubits. As q_0 and q_2 are part of a shared CX gate, the current implementation of the circuit violates the connectivity constraints. By introducing a `SWAP` gate between q_0 and q_1 before the CX gate, as well as a second `SWAP` afterwards to return to the starting configuration, learners can compile the sample program for the simple coupling map.*

D. Verification

An important procedure that is part of the quantum design automation pipeline is the verification of circuits. It is required to show that changes made to the circuit during the compilation process did not introduce errors. One popular approach for the verification of compilation outcomes is by performing an equivalence check with the previous state of the circuits [50, 51]. For unitary quantum circuits, this can be achieved by concatenating the inverse of the original circuit to the application of the updated circuit. If this reduces the state to its starting state, the circuits are equivalent and the transformations were applied correctly.

In the context of the proposed framework, performing this check is particularly easy: If the considered circuit only uses self-inverse operations, users can simply drag the operation blocks from the starting circuit to the end of the new circuit in reverse order. By then measuring all qubits and displaying the outcomes over several repeated simulations, students can interactively experience this verification process and gain an intuitive understanding of its workings. In cases where an introduced error only amounts to a probabilistic difference in the final measurement outcomes, they experience how the accuracy of this method depends on the number of performed simulations. By performing the simulations from different starting states, learners can also see how different implementations might be equivalent for some starting states but differ for others.

Especially with growing research of quantum algorithms, leading to bigger and more complex circuits, testing and debugging quantum programs is becoming increasingly important. The learning outcome of this step not only provides students with a deeper insight into an important tool for the verification of circuit transformations, but also emphasizes the importance of testing to new learners.

Example 9. *Revisiting the transformed circuit constructed in Example 8, we want to check its equivalence with the original circuit defined in Fig. 8. After removing the non-unitary measurements from the transformed and original circuits, we construct the inverse of the original circuit by applying its gates in reverse order. The result of this process is shown in Fig. 8b. We then store the measurement results in classical variables and simulate the verification circuit several times. As all of the measurement outcomes are equal to* 0*, we can assume that the circuits are likely equal for the starting state* $|000\rangle$*. However, to check for the full equivalence of the circuits, we must further compare the outcomes for all potential starting states. We achieve this by iterating over all possible starting states in the computational basis and using the "*Run quantum circuits starting from [STATE]*" instruction. This evaluation shows that the circuits are not equivalent for some starting states, such as* $|010\rangle$*.*

E. Going Further

To complete the education process for new learners interested in quantum design automation, they must also be provided with an access pass to the plethora of software tools currently available in the ecosystem. These different tools provide a variety of functionalities, ranging from more sophisticated implementations of the strategies discussed in this section to error correction methods, HPC integration strategies and end user support software. While the *Scratch* programming platform provides an accessible playground that makes concepts easier to grasp for new learners, users should also be given the opportunity to move on to more complex tools as they gain experience with quantum computing. To this end, as a final step, we show learners how the "*Export to MLIR*" block can be used to generate a textual representation of the implemented circuit. This provides them with an interesting insight into the differences of the representations—including the verbosity of the MLIR-based IR. It also gives learners an access path to the remaining ecosystem through further translations.

Example 10. *After exporting the constructed circuit, we store the generated MLIR representation locally. We then use it as input for the* mqt-cc *compilation framework. Through its built-in QIR pipeline, we can then compile the program to a base profile QIR representation for a desired superconducting device topology. This representation can then be used to execute the circuit on the quantum hardware, successfully bridging the gap between the proposed* Scratch*-based platform and the further quantum ecosystem down to actual physical execution.*

VI. Evaluation

To evaluate the pedagogical effectiveness of the proposed framework, we carried out a user study with a total of 24 users. We asked them to complete a set of exercises and fill out a questionnaire to test the retained information. The following first describes the methodology employed for the study, and then presents and discusses the obtained results.

A. Methodology and Study Design

We conducted a user study with post-graduate students from computer science and electrical engineering degrees. These students had some fundamental understanding of quantum computing (such as the concepts of qubits and the effects of individual gates), but no extensive experience in quantum design automation. To capture a realistic range of learning environments, the study was deployed in two distinct formats: an in-person, supervised laboratory session ($N = 13$) and an asynchronous, take-home assignment ($N = 11$).

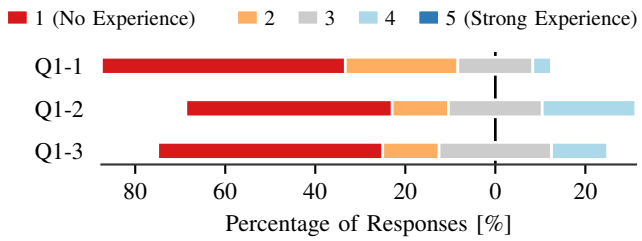
Both groups were provided with a set of exercises which required them to mirror the quantum design automation workflow outlined in Section V—moving from initial circuit construction to resource estimation, optimization, and verification. Upon completing the practical tasks, participants provided structured feedback through a questionnaire designed to assess their comprehension of the underlying concepts and their perceived usability of the block-based environment. The questionnaire is divided into the following topic areas:

Q1 – Prior Experience: This group of questions evaluates the prior experience of participants in the field. It asks participants to rate their expertise in quantum design automation topics such as resource estimation and verification, as well as their previous experience with the *Scratch* programming platform on a Likert scale [81].

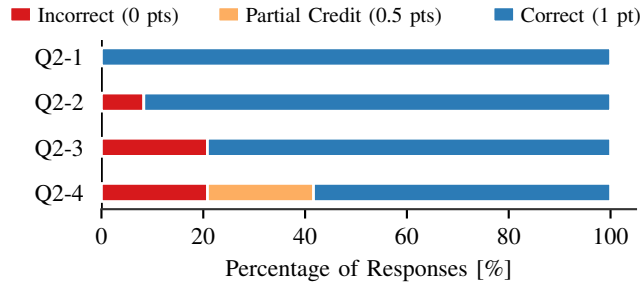
Q2 – Competence: Next, participants are tasked with answering a set of questions related to the concepts covered in the exercises. This allows us to evaluate how well the exercises have conveyed these concepts to new learners.

Q3 – Usability: Finally, this section asks the users for Likert-scale feedback on the ease of use of the proposed framework. Participants are given the ability to rate their experience on a numerical scale and provide textual feedback.

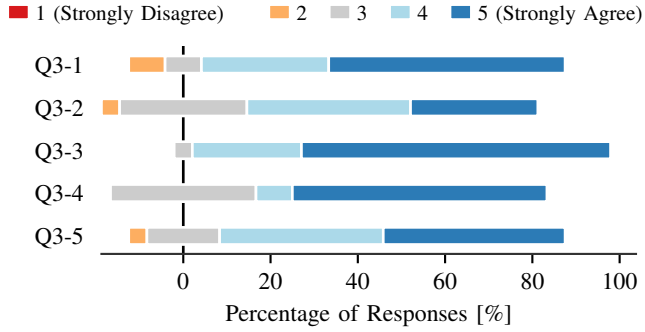
Some participants also took part in an optional follow-up questionnaire one week later. This follow-up once again asks the *competence* questions to evaluate how well they have retained the conveyed concepts.



(a) Self-reported previous experience of users with *Scratch* (*Q1-1*), circuit verification (*Q1-2*), and resource estimation (*Q3-1*).



(b) Assessment results after completing the exercises on reversing quantum circuits (*Q2-1*), verification (*Q2-2*), compilation (*Q2-3*), and resource estimation (*Q2-4*).



(c) Self-reported user experience outcomes after conducting the study, indicating satisfaction with the block-based environment (*Q3-1*), quantum-classical hybrid programs (*Q3-2*), verification (*Q3-3*), resource estimation (*Q3-4*), and their confidence to explain the conveyed concepts to a fellow student (*Q3-5*).

Figure 9. Results of the questionnaires filled out by the participants after completing the exercises.

B. User Study Results

To establish a baseline, participants self-reported their *prior experience* with some of the covered concepts. In particular, the questionnaire asks for the participants' experience with the *Scratch* platform, quantum *resource estimation* (RE), and quantum circuit *verification*. The corresponding results are illustrated in Fig. 9a. They show that a majority of the participants did not have any major experience with either of the considered concepts. In particular, more than 50% of the participants did not report any experience with the *Scratch* platform at all. Similarly, a majority of participants only indicated limited prior experience with the quantum computing concepts covered by the exercises. The median total reported prior expertise—obtained after averaging all individual question outcomes for each participant—was 1.83 (close to "*little experience*"). In general, the evaluation of this topic area shows that the participants previously only had limited contact with the topics covered in the exercises.

Conversely, Fig. 9b shows the results of the *competence* questions, gathered through assessment test questions as parts of the questionnaire. *Q2-1* to *Q2-3* are single-choice questions that ask the participants to select one of four possible answers related to quantum circuit compilation and *verification*. Participants received full credits for selecting the correct answer and no points for selecting an incorrect answer. *Q2-4* is a multiple-choice question that requires participants to select potential use cases of *resource estimation*. Partial credits are given in cases, where participants did not select all correct choices but also did not select any incorrect choices. The median total score of all participants was 3.5 out of 4 points, with a mean of 3.40 and a standard deviation of 0.61. Furthermore, the median participant reported they were easily able to complete the exercise, with only some participants requiring significant additional time. This shows that the exercises, carried out through the proposed framework, effectively conveyed the discussed topics, with most participants correctly responding to a majority of the questions.

To gauge the proposed framework's *usability*, we also asked them to self-report their experience during the completion of the tasks. Fig. 9c illustrates the outcomes of this topic. In particular, a majority of participants was satisfied with the platform, either agreeing of strongly agreeing with its usefulness in learning quantum computing concepts. Notably, for the process of inverting quantum circuits through the *Scratch* extension, only one of the 24 participants selected "*neither agree nor disagree*", while all other participants agreed or strongly agreed with their satisfaction. More than half of the participants felt confident that they would be able to explain the covered concepts to their peers after the exercises. In total, the median evaluation feedback for *Q3* was 4.3. This indicates that the proposed framework provided a satisfactory user experience for the participants.

To evaluate the retention of the conveyed concepts, we performed a second round of *competence* assessments one week after the exercises were performed. A total of $N = 20$ participants took part of this second assessment.

Table I compares the assessment results obtained directly after the exercises with those in the retention test. Notably, participants achieved equal or better results in all questions, except for *Q2-3*, where the mean score fell by a total 4.2%. As none of these differences are significant, with even a difference of 4.2% only amounting to approximately one particpant, the test shows that participants were not only able to solve the test questions successfully directly after completing the exercises, but even after some time had already passed.

As the study was carried out through different modalities—a supervised in-person session and a take-home exercise—we evaluated the framework's robustness across different learning

Table I
CONCEPT RETENTION ($N = 20$)

Concept	Initial Pass Rate	1-Week Pass Rate	Δ
Q2-1: Reversing Circuits	100.0%	100.0%	0%
Q2-2: Verification	91.7%	95.0%	3.3%
Q2-3: Compilation	79.2%	75.0%	-4.2%
Q2-4: RE	68.8%	70.0%	1.4%

environments by comparing the different cohorts. The remote cohort achieved an overall mean *competence* score of 3.36 (SD $= 0.64$, MD $= 3.5$), closely mirroring the in-person mean score of 3.42 (SD $= 0.61$, MD $= 3.5$). Similarly, the remote cohort reported an overall median *usability* score of 4.4, while the in-person cohort reported a similar median score of 4.0.

A *Mann-Whitney U* test [82, 83] revealed no statistically significant difference in competence scores ($p = 0.831$) or perceived usability ($p = 0.134$) between the two groups. This strongly suggests that the proposed block-based framework is highly effective as a standalone, asynchronous educational tool, requiring no direct supervision to yield positive learning outcomes.

Finally, the questionnaire also allowed participants to provide additional open-text feedback. This feedback strongly supports the previous findings in this evaluation. For instance, one participant notes: "It is nice to have easily modifiable circuits and intuitive blocks provided by the quantum extension". Further participants specifically commented on performing equivalence checking through the exercises, mentioning, for instance: "I liked the equivalence checking, because it visualized what it means to apply the reverse matrice very well". The questionnaire also asked for problems that arose during the completion of the tasks and usability friction points. The main consensus of this question was that additional visualization methods would improve the clarity of the interface. One user further asked to "[...] provide pre-built blocks to print everything", as the manual concatenation of measurement outcomes can be tedious for larger circuits. Nonetheless, the participants largely agreed that the exercises were interesting and insightful, with one stating "I liked working with Scratch and building simple circuits in an intuitive manner".

C. Threats to Validity

While the conducted evaluations have shown positive results, some additional factors must be considered when observing the outcomes of the questionnaires. Firstly, the *competence* questions consisted only of four single or multiple choice questions. While this does not exhaustively capture the participants' full understanding of the concepts, the questions were selected specifically to cover the most important aspects of the exercises. Furthermore, self-reported usability scores may be subject to bias. However, the anonymity of questionnaire results mitigates this. While students who participated in the study remotely may have utilized unauthorized external resources during the exercises and while filling out the questionnaire, the comparison of the two cohorts largely eliminates this concern, finding no significant statistical difference among them. Additionally, because the retention test was optional, the 1-week follow-up results may be subject to self-selection bias, as more motivated students were more likely to complete it.

Finally, as all participants were post-graduate computer science and electrical engineering students, the obtained results may not generalize to more general learners. Future work requires larger, cross-disciplinary cohorts to validate the proposed framework's usability across different levels of experience.

VII. CONCLUSION

In this work, we presented a block-based framework for the development of quantum circuits as an educational tool for teaching quantum design automation concepts. The framework is built as an extension of the *Scratch* programming platform, allowing users to combine classical instructions for control flow, data storage, and user interfacing with quantum instructions for the creation of circuits. Created circuits can be executed using a built-in simulator and analyzed by an integrated resource estimation procedure. Through direct translations to other intermediate representations, created quantum circuits can further be used to facilitate the access to the wider quantum computing ecosystem. A user study confirmed that, after the completion of a set of quantum design automation tasks, most participants gained a better understanding of quantum design automation concepts, with a majority of participants rating the framework's usability positively. A web version of the proposed framework can be accessed online at https://munich-quantum-toolkit.github.io/scratch-quantum/.

Future work includes the extension of the simulation backend to support a wider set of different simulator types, allowing learners to compare the performance of simulators for various quantum circuits, as well as the implementation of additional visualization methods within the framework to further increase accessibility.

ACKNOWLEDGMENTS

While preparing the manuscript, Claude Sonnet 4.6, Claude Opus 4.6, Gemini 3 Pro and gpt-5.3-codex were used through GitHub Copilot to improve readability, spelling, grammar, and clarity throughout the work. Each LLM output was reviewed by the authors and edited manually as needed. The authors take full responsibility for the final content. The authors acknowledge funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program grant agreement No. 101001318, as well as from the Munich Quantum Valley, which is supported by the Bavarian state government with funds from the Hightech Agenda Bayern Plus. Furthermore, this work was supported by the BMFTR under grant No. 01MQ25001I (FullStaQD) and by BMIMI, BMWET, the State of Upper Austria and the State of Tyrol within the COMET module Quantum Algorithm Engineering (FFG grant No. 923923) managed by the Austrian Research Promotion Agency FFG.

References

[1] Laung-Terng Wang, Yao-Wen Chang, and Kwang-Ting (Tim) Cheng, *Electronic Design Automation: Synthesis, Verification, and Test*. Morgan Kaufmann, 2009.

[2] "IBM Quantum Composer." (2016), [Online]. Available: https://quantum.cloud.ibm.com/composer (visited on 04/17/2026).

[3] S. Gayathri Devi, S. Manjula Gandhi, S. Chandia, and P. Boobalaragavan, "Exploring IBM Quantum Experience," in *Quantum Computing: A Shift from Bits to Qubits*, Springer Nature, 2023.

[4] Mark Koch, Alan Lawrence, Kartik Singhal, Seyon Sivarajah, and Ross Duncan, *GUPPY: Pythonic Quantum-Classical Programming*, 2025. arXiv: 2510.12582 [cs].

[5] Ville Bergholm *et al.*, *Pennylane: Automatic differentiation of hybrid quantum-classical computations*, 2022. arXiv: 1811.04968 [quant-ph].

[6] Qiskit contributors, *Qiskit: An open-source framework for quantum computing*, 2023.

[7] "Quirk." (2019), [Online]. Available: https://github.com/Strilanc/Quirk (visited on 04/17/2026).

[8] Mark Koch *et al.*, *HUGR: A Quantum-Classical Intermediate Representation*, 2025. arXiv: 2510.11420 [cs].

[9] D. Ittah *et al.*, "Catalyst: A python JIT compiler for auto-differentiable hybrid quantum programs," *Journal of Open Source Software*, 2024.

[10] David Ittah, Thomas Häner, Vadym Kliuchnikov, and Torsten Hoefler, "QIRO: A Static Single Assignment-based Quantum Program Representation for Optimization," *ACM Transactions on Quantum Computing*, 2022.

[11] QuEra Computing Inc, *Kirin: Kernel intermediate representation infrastructure*, 2025. [Online]. Available: https://queracomputing.github.io/kirin/latest/.

[12] L. Burgholzer *et al.*, "The MQT Compiler Collection: A blueprint for a future-proof quantum-classical compilation framework," in *Design, Automation and Test in Europe*, 2026.

[13] Mitchel Resnick *et al.*, "Scratch: Programming for all," *Comm. of the ACM*, 2009.

[14] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond, "The Scratch Programming Language and Environment," *ACM Trans. Comput. Educ.*, 2010.

[15] Robert Wille *et al.*, "The MQT Handbook: A Summary of Design Automation Tools and Software for Quantum Computing," in *IEEE International Conference on Quantum Software (QSW)*, 2024.

[16] Paul Molitor and Janett Mohnke, *Equivalence checking of digital circuits: Fundamentals, principles, methods*. Springer, 2010.

[17] Daniel Brand, "Verification of large synthesized designs," in *Int'l Conf. on CAD*, 1993.

[18] João Marques-Silva and Thomas Glass, "Combinational equivalence checking using satisfiability and recursive learning," in *Design, Automation and Test in Europe*, 1999.

[19] S. Jha, Y. Lu, M. Minea, and E. M. Clarke, "Equivalence checking using abstract BDDs," in *Int'l Conf. on Comp. Design*, 1997.

[20] Seyon Sivarajah *et al.*, "T|ket⟩: A retargetable compiler for NISQ devices," *Quantum Science and Technology*, 2020.

[21] Yuri Alexeev *et al.*, "Artificial intelligence for quantum computing," *Nature Communications*, 2025.

[22] Nils Quetschlich, Lukas Burgholzer, and Robert Wille, "MQT Predictor: Automatic Device Selection with Device-Specific Circuit Compilation for Quantum Computing," *ACM Transactions on Quantum Computing*, 2025.

[23] Thomas Fösel, Murphy Yuezhen Niu, Florian Marquardt, and Li Li, *Quantum circuit optimization with deep reinforcement learning*, 2021. arXiv: 2103.07585 [quant-ph].

[24] Lorenzo Moro, Matteo G. A. Paris, Marcello Restelli, and Enrico Prati, "Quantum compiling by deep reinforcement learning," *Communications Physics*, 2021.

[25] A. Tudisco *et al.*, "Fidelity-Based Quantum Device Selection Using Graph Neural Networks," in *Int'l Conf. on Quantum Software*, 2026.

[26] Ang Li, Samuel Stein, Sriram Krishnamoorthy, and James Ang, "QASMBench: A Low-Level Quantum Benchmark Suite for NISQ Evaluation and Simulation," *ACM Transactions on Quantum Computing*, 2023.

[27] Nils Quetschlich, Lukas Burgholzer, and Robert Wille, "MQT Bench: Benchmarking Software and Design Automation Tools for Quantum Computing," *Quantum*, 2023.

[28] Gushu Li *et al.*, "Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019.

[29] Felix Zilk *et al.*, "A compiler for universal photonic quantum computers," in *IEEE/ACM International Workshop on Quantum Computing Software (QCS)*, 2022.

[30] L. Schmid *et al.*, "Computational Capabilities and Compiler Development for Neutral Atom Quantum processors—Connecting Tool Developers and Hardware Experts," *Quantum Science and Technology (QST)*, 2024.

[31] Ji-Hoon Kang and Hoon Ryu, *PennyLane-Lightning MPI: A massively scalable quantum circuit simulator based on distributed computing in CPU clusters*, 2025. arXiv: 2508.13615 [quant-ph].

[32] Harun Bayraktar *et al.*, "cuQuantum SDK: A High-Performance Library for Accelerating Quantum Science," in *Int'l Conf. on Quantum Computing and Engineering*, 2023.

[33] Ali Javadi-Abhari *et al.*, *Quantum computing with Qiskit*, 2024. arXiv: 2405.08810 [quant-ph].

[34] T. Grurl, J. Fuß, and R. Wille, "Noise-aware Quantum Circuit Simulation With Decision Diagrams," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2022.

[35] A. Zulehner and R. Wille, "Advanced Simulation of Quantum Computations," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2018.

[36] T. Forster, N. Quetschlich, M. Soeken, and R. Wille, "Improving Hardware Requirements for Fault-Tolerant Quantum Computing by Optimizing Error Budget Distributions," in *Int'l Conf. on Quantum Computing and Engineering*, 2025.

[37] T. Forster, N. Quetschlich, and R. Wille, "Quantum Circuit Optimization for the Fault-Tolerance Era: Do We Have to Start from Scratch?" In *Int'l Conf. on Quantum Computing and Engineering*, 2025.

[38] Martin Suchara *et al.*, "QuRE: The Quantum Resource Estimator toolbox," in *IEEE International Conference on Computer Design (ICCD)*, 2013.

[39] Wim van Dam, Mariia Mykhailova, and Mathias Soeken, *Using azure quantum resource estimator for assessing performance of fault tolerant quantum computation*, 2023. arXiv: 2311.05801 [quant-ph].

[40] Li Zhou, Nengkun Yu, and Mingsheng Ying, "An applied quantum Hoare logic," in *Conference on Programming Language Design and Implementation*, Jun. 8, 2019.

[41] Yipeng Huang and Margaret Martonosi, "Statistical assertions for validating patterns and finding bugs in quantum programs," in *International Symposium on Computer Architecture*, 2019.

[42] Gushu Li *et al.*, *Proq: Projection-based Runtime Assertions for Debugging on a Quantum Computer*, 2020. arXiv: 1911.12855 [quant-ph].

[43] D. Rovara, L. Burgholzer, and R. Wille, "A Framework for the Efficient Evaluation of Runtime Assertions on Quantum Computers," in *Int'l Conf. on Quantum Software*, 2026.

[44] Mingsheng Ying, *Birkhoff-von Neumann quantum logic as an assertion language for quantum programs*, May 4, 2022. arXiv: 2205.01959[quant-ph] [quant-ph].

[45] Ji Liu, Gregory T. Byrd, and Huiyang Zhou, "Quantum circuits for dynamic runtime assertions in quantum computation," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020.

[46] Ji Liu and Huiyang Zhou, "Systematic approaches for precise and approximate quantum state runtime assertion," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2021.

[47] Hasini Witharana, Daniel Volya, and Prabhat Mishra, *quAssert: Automatic generation of quantum assertions*, Mar. 2, 2023. arXiv: 2303.01487 [quant-ph].

[48] D. Rovara, L. Burgholzer, and R. Wille, "Automatically Refining Assertions for Efficient Debugging of Quantum Programs," in *Int'l Conf. on Quantum Computing and Engineering*, 2025.

[49] D. Rovara, L. Burgholzer, and R. Wille, "A Framework for Debugging Quantum Programs," in *Int'l Conf. on Quantum Software*, 2025.

[50] Shigeru Yamashita and Igor L. Markov, "Fast equivalence-checking for quantum circuits," in *2010 IEEE/ACM International Symposium on Nanoscale Architectures*, 2010.

[51] Lukas Burgholzer and Robert Wille, "Advanced Equivalence Checking for Quantum Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[52] Krishna Upadhyay, Vinaik Chhetri, A.B. Siddique, and Umar Farooq, "Analyzing the Evolution and Maintenance of Quantum Software

Repositories," in *2025 IEEE International Conference on Quantum Software (QSW)*, 2025.
[53] Olivia Di Matteo, "On the need for effective tools for debugging quantum programs," in *International Workshop on Quantum Software Engineering*, 2024.
[54] Sara Ayman Metwalli and Rodney Van Meter, "Testing and Debugging Quantum Circuits," *IEEE Transactions on Quantum Engineering*, 2024.
[55] Andriy Miranskyy, Lei Zhang, and Javad Doliskani, *On testing and debugging quantum software*, 2021. arXiv: 2103.09172 [quant-ph].
[56] Antonio García de la Barrera, Ignacio García-Rodríguez de Guzmán, Macario Polo, and Mario Piattini, "Quantum software testing: State of the art," *Journal of Software: Evolution and Process*, 2023.
[57] N. Quetschlich, T. Forster, A. Osterwind, D. Helms, and R. Wille, "Towards Equivalence Checking of Classical Circuits Using Quantum Computing," in *Int'l Conf. on Quantum Computing and Engineering*, 2024.
[58] D. Rovara, N. Quetschlich, and R. Wille, "A Framework to Formulate Pathfinding Problems for Quantum Computing," in *Int'l Symp. on Multi-Valued Logic*, 2026.
[59] R. Wille and L. Burgholzer, "MQT QMAP: Efficient Quantum Circuit Mapping," in *Int'l Symp. on Physical Design*, 2023.
[60] D. Rovara, L. Burgholzer, and R. Wille, "Quantum Hardware-Efficient Selection of Auxiliary Variables for QUBO Formulations," in *Design, Automation and Test in Europe*, 2026.
[61] D. Rovara, L. Burgholzer, and R. Wille, *Qubit Reuse Beyond Reorder and Reset: Optimizing Quantum Circuits by Fully Utilizing the Potential of Dynamic Circuits*, 2025. arXiv: 2511.22712 [quant-ph].
[62] Kumar Prateek and Soumyadev Maity, "Quantum Programming on Azure Quantum—An Open Source Tool for Quantum Developers," in *Quantum Computing: A Shift from Bits to Qubits*, Springer Nature, 2023.
[63] Andrew Cross *et al.*, "OpenQASM 3: A Broader and Deeper Quantum Assembly Language," *ACM Transactions on Quantum Computing*, 2022.
[64] Chris Lattner *et al.*, "MLIR: Scaling Compiler Infrastructure for Domain Specific Computation," in *IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2021.
[65] Raphael Seidel *et al.*, *Qrisp: A Framework for Compilable High-Level Programming of Gate-Based Quantum Computers*, 2024. arXiv: 2406.14792 [quant-ph].
[66] QIR Alliance, *QIR Specification*, see also https://qir-alliance.org, 2021. [Online]. Available: https://github.com/qir-alliance/qir-spec.
[67] P. Hopf *et al.*, "Integrating quantum software tools with(in) MLIR," in *Proceedings of the Supercomputing Asia and International Conference on High Performance Computing in Asia Pacific Region*, 2026.
[68] Javier Zayas Gallardo, Enrique Moguel, Carlos Canal, and Jose Garcia-Alonso, "Quirk+: A Tool for Quantum Software Development Based on Quirk," in *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering - Companion (SANER-C)*, 2024.
[69] A. D. Córcoles *et al.*, "Exploiting Dynamic Quantum Circuits in a Quantum Algorithm with Superconducting Qubits," *Physical Review Letters*, 2021.
[70] J. M. Pino *et al.*, "Demonstration of the trapped-ion quantum CCD computer architecture," *Nature*, 2021.
[71] D.J. Egger *et al.*, "Pulsed Reset Protocol for Fixed-Frequency Superconducting Qubits," *Physical Review Applied*, 2018.
[72] D. T. McClure *et al.*, "Rapid Driven Reset of a Qubit Readout Resonator," *Physical Review Applied*, 2016.
[73] Lov K. Grover, "A fast quantum mechanical algorithm for database search," in *Theory of computing*, 1996.
[74] Daniel Escanez-Exposito, Marcos Rodriguez-Vega, Carlos Rosa-Remedios, and Pino Caballero-Gil, "QScratch: Introduction to quantum mechanics concepts through block-based programming," *EPJ Quantum Technology*, 2025.
[75] Carlos Rosa-Remedios, Pino Caballero-Gil, and Daniel Escanez-Exposito, "WIP: Teaching Basic Concepts of Quantum Computing Using Scratch," in *2024 IEEE Frontiers in Education Conference (FIE)*, 2024.
[76] Michal Armoni, Orni Meerbaum-Salant, and Mordechai Ben-Ari, "From Scratch to "Real" Programming," *ACM Trans. Comput. Educ.*, 2015.
[77] Jeff contributors. "Jeff Bridges Compilation." (2025), [Online]. Available: https://github.com/unitaryfoundation/jeff (visited on 04/13/2026).
[78] Giulia Meuli, Mathias Soeken, Martin Roetteler, and Thomas Häner, "Enabling accuracy-aware Quantum compilers using symbolic resource estimation," *Proc. ACM Program. Lang.*, 2020.
[79] Michael A. Nielsen and Isaac L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.
[80] Anthony Ransford, M. S. Allman, Jake Arkinstall, J. P. Campora III, and Samuel F. Cooper et al., *Helios: A 98-qubit trapped-ion quantum computer*, 2025. arXiv: 2511.05465 [quant-ph].
[81] R. Likert, "A technique for the measurement of attitudes," *Archives of Psychology*, 1932.
[82] H. B. Mann and D. R. Whitney, "On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other," *The Annals of Mathematical Statistics*, 1947.
[83] Patrick E. McKnight and Julius Najab, "Mann-Whitney U Test," in *The Corsini Encyclopedia of Psychology*, John Wiley & Sons, Ltd, 2010.