Weaver: A Retargetable Compiler Framework for FPQA Quantum Architectures

Oğuzcan Kırmemiş*
oguzcan.kirmemis@gmail.com
Technical University of Munich
Munich, Germany

Francisco Romão*
francisco.romao@tum.de
Technical University of Munich
Munich, Germany

Emmanouil Giortamis
emmanouil.giortamis@in.tum.de
Technical University of Munich
Munich, Germany

Pramod Bhatotia
pramod.bhatotia@tum.de
Technical University of Munich
Munich, Germany

Abstract

While the prominent quantum computing architectures are based on superconducting technology, new quantum hardware technologies are emerging, such as Trapped Ions, Neutral Atoms (or FPQAs), Silicon Spin Qubits, etc. This diverse set of technologies presents fundamental trade-offs in terms of scalability, performance, manufacturing, and operating expenses. To manage these diverse quantum technologies, there is a growing need for a *retargetable compiler* that can efficiently adapt existing code to these emerging hardware platforms. Such a retargetable compiler must be *extensible* to support new and rapidly evolving technologies, *performant* with fast compilation times and high-fidelity execution, and verifiable through rigorous equivalence checking to ensure the functional *equivalence* of the retargeted code.

To this end, we present Weaver, the first extensible, performant, and verifiable retargetable quantum compiler framework with a focus on FPQAs due to their unique, promising features. Weaver introduces wQasm, the first formal extension of the standard OpenQASM quantum assembly with FPQA-specific instructions to support their distinct capabilities. Next, Weaver implements the wOptimizer, an extensible set of FPQA-specific optimization passes to improve execution quality. Last, the wChecker automatically checks for equivalence between the original and the retargeted code. Our evaluation shows that Weaver improves compilation times by $10^3\times$, execution times by 4.4×, and execution fidelity by 10%, on average, compared to superconducting and state-of-the-art (non-retargetable) FPQA compilers.

*Both authors contributed equally to the paper

CGO '25, March 01–05, 2025, Las Vegas, NV, USA

ACM ISBN 979-8-4007-1275-3/25/03
https://doi.org/10.1145/3696443.3708965

CCS Concepts: • **Software and its engineering → Retargetable compilers**; • **Hardware → Quantum technologies**.

Keywords: Quantum Computing, Quantum Software Systems

ACM Reference Format:
Oğuzcan Kırmemiş, Francisco Romão, Emmanouil Giortamis, and Pramod Bhatotia. 2025. Weaver: A Retargetable Compiler Framework for FPQA Quantum Architectures. In *Proceedings of the 23rd ACM/IEEE International Symposium on Code Generation and Optimization (CGO '25), March 01–05, 2025, Las Vegas, NV, USA.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3696443.3708965

1 Introduction

Quantum computing offers the potential to solve computational problems intractable by classical computers by leveraging the principles of quantum mechanics [4, 27], with applications in cryptography [87], optimization [27], chemistry [42, 73], machine learning [8], among others.

Quantum computing is already practically available as quantum processors are mainly offered on the cloud [5, 33, 37]. These processors are predominantly manufactured based on the Superconducting technology [4], although quantum hardware development is progressing rapidly, with various candidate technologies emerging, including Trapped Ions [14], Neutral Atoms [35], and Photonics [65].

These different quantum hardware technologies present trade-offs between performance metrics, manufacturing complexity, and operational requirements. Typical performance metrics include gate speeds, akin to clock speed in CPUs, coherence times, i.e., the time the system maintains quantum mechanical properties, and gate error rates, i.e., the probabilities of erroneous gate output [32]. In terms of operational requirements, some technologies require close to absolute zero temperatures [45, 101], which is extremely costly, while others can operate at room-level temperatures [1, 25].

Given such tradeoffs, it is evident that these diverse quantum technologies will co-exist in parallel. This coexistence creates a need for retargetable quantum compilers that can seamlessly adapt quantum programs to different hardware platforms. Specifically, retargetable compilers allow researchers and developers to leverage the unique advantages of each technology without rewriting or redesigning their quantum algorithms from scratch, thereby enhancing flexibility, accelerating development, and maximizing the utility of existing quantum resources across multiple hardware platforms.

More specifically, a retargetable compiler must offer three fundamental properties. **First**, *extensiblilty* to support new technologies and instructions in evolving technologies. **Second**, *performance and fidelity*, i.e., incur low compilation runtimes and optimize the code to leverage the underlying hardware's unique capabilities for improved execution fidelity. **Third**, *verifiability*, i.e., the compiler is rigorously checked to ensure functional equivalence to the original program.

However, designing such a retargetable compiler framework poses its own challenges. First, extensibility is non-trivial in the largely heterogeneous and evolving quantum landscape since new hardware features are being developed constantly. Second, performance is hard to achieve: fast compilation is challenging due to NP-hard compilation stages [15], and high execution fidelity is challenging due to hardware noise [75]. Last, checking the original and retargeted binary for functional equivalence is especially hard in quantum due to inherent randomness and computational complexity [12].

To target these challenges, we pose the following research question: *How to design an extensible, performant, and verifiable retargetable compiler framework by building on existing open standards?*

To answer this question, we propose Weaver, the first retargetable quantum compiler framework that automatically generates, optimizes, and verifies high-fidelity code for diverse quantum technologies. Weaver primarily focuses on two prominent technologies: superconducting qubits and neutral atoms. The former is widely studied in research and industry settings, offering a mature and comprehensive ecosystem. Neutral atoms, and specifically, Field-Programmable-Quantum-Arrays (FPQAs), show better scalability, more flexible arrangement, lower cooling requirements, and higher parallelism [35, 48, 49, 81], compared to superconducting.

We design Weaver with minimal assumptions about the hardware specifications since FPQAs are still an emerging technology that will evolve over time. Specifically, ① Weaver introduces the ***wQasm* assembly language**, the first formal extension of the standardized and widely used OpenQASM quantum assembly [16] with FPQA-specific backend instructions (§ 4). ② Weaver optimizes programs for FPQAs by implementing ***wOptimizer***, an extensible set of optimization passes that leverage FPQA capabilities to increase parallelization, reduce the program execution time, and lower the hardware noise overheads (§ 5). Lastly ③, Weaver introduces ***wChecker*** which checks for equivalence between the technology-agnostic and the target-compiled binaries (§ 6).

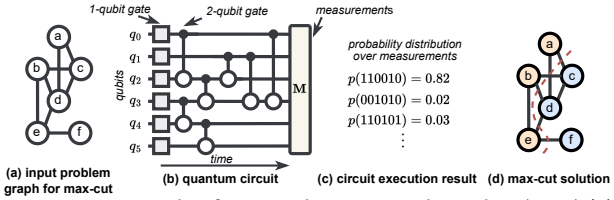

Figure 1. Example of a typical quantum algorithm (§ 2.1) **(a)** Input graph for max-cut. **(b)** The quantum circuit encoding the formulation of max-cut for the graph. **(c)** The result of circuit execution is a probability distribution of bitstrings. **(d)** The result of (c) is interpreted as a max-cut between vertices $\{a,b,e\}$ and $\{c,d,f\}$.

We implement Weaver in Python on top of Qiskit [76] and OpenQASM [17] and PySAT [38]. We evaluate Weaver using state-of-the-art quantum applications on IBM quantum devices and FPQA simulators. Our results show that wOptimizer achieves $5.7*10^3\times$ faster compilation times, improves execution times by 4.4×, and execution fidelity by 10% for small applications and $1.27*10^5$ for applications of increasing size, on average, compared to superconducting architectures and state-of-the-art (non-retargetable) FPQA-specific compilers, namely *Geyser* [68], *Atomique* [102], and *DPQA* [94].

2 Background

2.1 Quantum Computing 101: An Example

To understand the basics of quantum computing, consider the classic max-cut problem [43], which can be solved using the Quantum Approximate Optimization Algorithm (QAOA) [27]. QAOA is a hybrid quantum-classical algorithm that uses a quantum computer to run a parameterized quantum circuit while a classical computer optimizes the parameters. This process aims to find the optimal solution by minimizing a cost function encoded in the quantum circuit. The selection of the QAOA algorithm as a case study for Weaver is grounded in its capacity to exploit the distinctive characteristics of Max-3SAT formulations associated with NP-hard problems. These formulations are effectively represented within a QAOA circuit designed for execution on FPQA architectures.

Figure 1 shows how QAOA solves a max-cut problem on an example graph (a). The problem is encoded as a quantum circuit (b), with each qubit representing a vertex of the input graph. Quantum gates are applied over time to change the state of the qubits, and at the end, measurements provide bitstrings as output. Unlike classical circuits, quantum circuits are probabilistic due to the superposition property of qubits. Quantum gates have probabilistic effects, and the final result is obtained by executing the circuit multiple times. The solution is a probability distribution over all possible bitstrings of the measured qubits, as shown in Figure 1 (c).

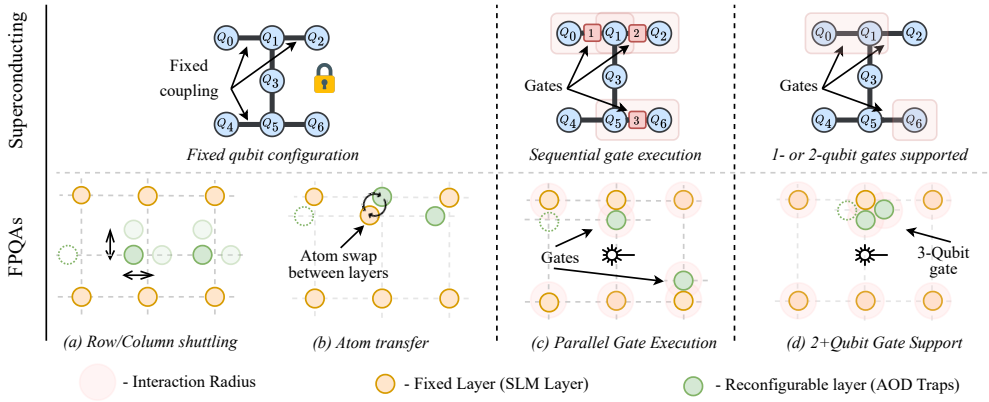

Figure 2. Unique capabilities of FPQA hardware (§ 2.3). FPQAs allow shuttling rows and/or columns of the reconfigurable atom layer, transferring atoms, executing gates in parallel, and native multi-qubit gates support.

In our example, the probability distribution represents the max-cut problem's solution, with high probabilities indicating potential solutions. In Figure 1 (d), the solution is the bitstring `110010`, which has the highest probability. This means qubits q_0, q_1, and q_4 are measured as 1, placing vertices $\{a,b,e\}$ in one partition and vertices $\{c,d,f\}$ in the other.

2.2 QPUs and Performance Metrics

QPU characteristics. Today's Quantum Processing Units (QPUs) are categorized as noisy intermediate-scale quantum (NISQ) devices [75] due to their limited number of qubits (up to a few hundred [37]) and their vulnerability to hardware and environmental noise [29]. Specifically, qubit measurements can result in bit-flip errors, and gate operations may perform incorrectly [32]. Additionally, qubits tend to collapse from the quantum (superposition) state and behave like classical bits (*decoherence effect* [44]) and destructively interfere with each other through *crosstalk effects* [18]. Last, QPU qubits can interact with each other only if they are connected with a physical link, with some QPUs having static connectivity layouts, while others have reconfigurable [4, 35].

Fidelity performance metric. Fidelity [28] is normally used to assess a circuit's performance on noisy QPUs, ranging from [0,1], comparing the noisy and ideal outputs. Similarly, we use the Estimated Probability of Success (EPS), which is the probability of one run outputting the correct result.

2.3 Quantum Hardware Architectures

Quantum hardware diversity. Currently, several candidate technologies are being actively developed, including superconducting qubits [4], trapped ions [14], photonics [65], and Field-Programmable Quantum Arrays (FPQAs) [35]. Each technology's characteristics differ (§ 2.2), rendering it more suitable for certain applications than others.

Prominent technologies. Weaver targets two quantum technologies: superconducting qubits and FPQAs. Superconducting technology has the advantages of fast gate execution [24, 57, 88], but mostly the ease of integration, maturity, and software support, compared to other architectures. However, it also suffers from low and rigid qubit connectivity, low coherence times, and high gate errors [61, 62, 96]. This is shown in Figure 2 (top), where the example qubit configuration is fixed. In contrast, FPQA hardware is gaining traction due to its longer coherence times, higher resistance to environmental noise, and flexible qubit connectivity [35, 48, 49, 81].

FPQA-unique capabilities.

Figure 2 presents four unique capabilities of FPQAs that show their advantage over superconducting qubits. First, FPQAs support dynamic qubit configuration and connectivity, referred to as qubit shuttling (Figure 2 (a)). FPQAs provide a fixed layer (yellow atoms) of traps and a moving layer (green atoms) that allows row or column shuttling, thus reconfiguring the qubit connectivity graph. Second, FPQAs enable the swapping of two atoms between two layers, i.e., atom transfer (Figure 2 (b)). This allows reconfiguring the connectivity graph without adding additional overheads on the quantum circuit, as done on superconducting technology. Third, FPQAs implement multi-qubit gates with global pulses (Figure 2 (c)), which allows parallel multi-qubit gate execution. Lastly, FPQAs natively support higher-order gates (Figure 2 (d)). Specifically, multi-qubit gates are not limited to 2-qubit interactions, being able to entangle three or more qubits.

FPQA opportunities and challenges. These features come with some challenges: first, movable rows or columns cannot move over one another, which means that for complete movement of qubits, we need to swap qubits back and forth between the fixed and moving layers; secondly, the global pulses apply gates in qubits that are close to each other, it thus, however, means that if two qubits are not supposed to interact, they need to be separated. Thirdly, gate execution is relatively slow; to counter this problem, the compiler should aim for high parallelization of gate execution.

3 Weaver Overview

Quantum technologies are constantly emerging and evolving, and specific technologies such as FPQAs are gaining traction

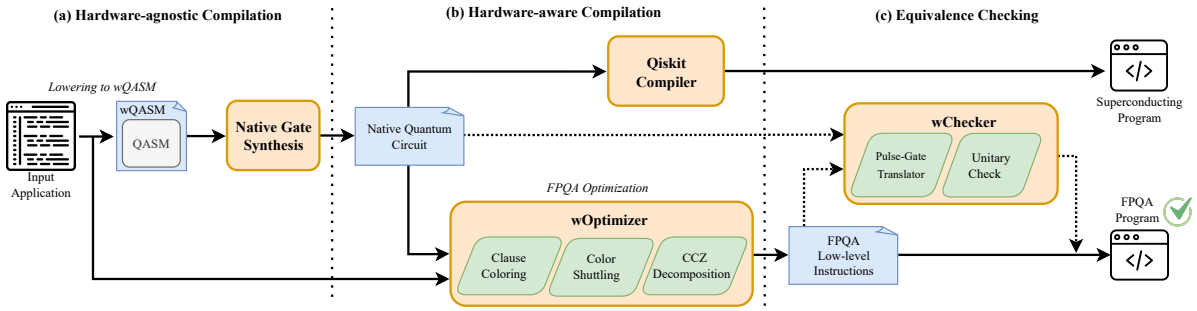

Figure 3. Weaver overview (§ 3). Weaver's workflow consists of three stages: (a) hardware-agnostic compilation, (b) hardware-aware compilation, and (c) equivalence checking. The hardware-agnostic compilation lowers the input application to wQasm. In the hardware-aware compilation stage, the compilation passes are specific to the target technology chosen by the user. Finally, equivalence checking ensures that the hardware-aware optimizations maintain the equivalence to the initial program.

because of their opportunities for performance improvement compared to the widely adopted superconducting architectures. However, designing a retargetable compiler that supports diverse quantum technologies while leveraging their strengths presents its own challenges.

3.1 Design Challenges

Challenge #1: Extensibility. A retargetable compiler must be extensible by supporting new quantum technologies, optimization passes, and instruction sets within a specific technology. To achieve this, it is crucial to adopt a standard and widely used common abstraction that acts as a common denominator across the compiler and other existing infrastructure.

Challenge #2: Performance. A retargetable compiler must be performant by leveraging heuristics for the NP-hard compilation steps to reduce compilation time and leveraging the hardware features to improve fidelity. This requires deploying efficient and target-specific optimizations that exploit the unique characteristics of each quantum device.

Challenge #3: Equivalence checking. A retargetable compiler must automatically check the functional equivalence of the retargeted circuit. The challenge lies in the *exponential* memory and computation requirements to represent and simulate quantum states using classical computers. Moreover, the inherent heterogeneity of quantum technologies introduces additional nondeterminism in the program outputs.

3.2 Weaver Retargetable Compiler Framework

Based on the previous key ideas, we present the design of our retargetable Weaver compiler framework. At a high level, Weaver extends the OpenQASM language and uses it as an IR to apply general-purpose optimizations. Then, Weaver leverages the FPQA-unique functionalities by implementing an extensible set of optimization passes that improve circuit duration, the number of operations, and overall fidelity. Lastly, Weaver verifies the correctness of the optimized circuit against the original one. The architecture of our compiler comprises three stages, as shown in Figure 3.

Hardware-agnostic compilation. Initially, a user submits an application to Weaver as a quantum circuit. Weaver lowers the circuit to wQasm IR and generates a *native* circuit using a gate set compatible with superconducting and FPQA technologies. We detail the wQasm extended instructions, the wQasm grammar, and the wQasm semantics in § 4.

Hardware-aware compilation. Depending on the user's backend choice, Weaver directs the native circuit to either the superconducting or the FPQA path. The superconducting (top arrow) path submits the circuit through the Qiskit Compiler [77], which fully compiles it to the superconducting backend. In the FPQA path (bottom-arrow), Weaver optimizes the circuit through the wCompiler, which applies three sequential optimization passes we detail in § 5. Finally, the circuit is converted to FPQA pulse instructions and is ready to be submitted to FPQA hardware controllers.

Equivalence checking. Lastly, during equivalence checking, Weaver submits the list of low-level FPQA pulses to the wChecker to ensure that the optimization passes maintain the behavior of the original nativized circuit. The wChecker comprises two steps, namely pulse-gate translator and unitary check, which we detail in § 6.

4 wQasm Extensions

wQasm extends OpenQASM for FPQAs with annotations that supply additional information on OpenQASM statements, defining FPQA-specific steps required before each statement.

OpenQASM as the foundation. OpenQASM (Open Quantum Assembly Language) [17] is designed with an assembly-like syntax targeting quantum algorithms and hardware operations. It is extensible through pragmas and annotations, which are compiler directives providing extra information for optimization or hardware-specific requirements. Pragmas are order-independent, while annotations specifically relate to the following OpenQASM statement. We opt for OpenQASM as our IR over other IRs for two reasons: (1) **Broad adoption and industry support**: OpenQASM is widely adopted in the quantum computing community and supported by major

quantum platforms. A good example is the large number of benchmark circuits written in OpenQASM [50, 78, 98]. (2) **Easily extensible**: OpenQASM supports pragmas and annotations, allowing extending the standard instructions.

4.1 wQasm FPQA Instructions

OpenQASM is designed to be hardware-agnostic. FPQA technology provides hardware flexibility and features that instructions can control. This motivates the creation of an annotation-based extension that does not change its hardware-agnostic characteristic. Besides the functionalities presented in § 2, FPQAs also provide two control pulses: **Raman pulse** is an atom-directed pulse that applies single-atom rotations around the x, y, or z axis. This pulse can be applied to one atom or globally by sending it to every initialized atom trap. **Rydberg pulse** is a global pulse that applies a multi-atom operator to all atoms close enough to each other within the Rydberg distance. The operation applied is a controlled-Z or multi-controlled-Z gate, depending on the number of interacting atoms.

4.2 wQasm Grammar

wQasm extends OpenQASM grammar rules with FPQA instructions as a superset of OpenQASM, detailed in Figure 4. In wQasm, OpenQASM instructions are annotated with their FPQA equivalents for direct FPQA programming. Challenges arise when FPQA instructions like shuttling don't directly correspond to logical gates and depend on the FPQA's previous atoms' positions. For instance, Rydberg pulses apply a transformation to atoms that were moved together by previous shuttling instructions. These non-pulse FPQA actions are then associated with the respective logical gates. A global Raman pulse also translates into a logical gate affecting all qubits, while a local Raman pulse uses a single $U3$ gate.

The wQasm file includes redundancy between the logical gate instructions from the original OpenQASM file and the FPQA-specific annotations. The annotations specify steps for each logical gate that must be executed sequentially, as each step depends on the previous state of the FPQA device. In contrast, logical gate instructions can be executed in parallel if their dependencies are met and they do not share qubits, following the order dictated by a dependency graph. This flexibility does not apply to FPQA annotations requiring a fixed sequence. During compilation, Weaver determines the execution order of the gates, and the wChecker then checks that the FPQA annotations correctly implement the same logical circuit as intended by the original OpenQASM code. Once the challenges are addressed, wQasm files can be treated like regular OpenQASM files (ignoring FPQA annotations). This allows them to be retargeted to other quantum architectures, possibly with additional compilation for specific hardware.

4.3 wQasm Semantics

We formalize the wQasm semantics and thoroughly explain its objective, arguments, pre-conditions, and the outcome of

```
⟨program⟩ ::= ⟨version⟩ ? ⟨statementOrScope⟩*

⟨version⟩ ::= 'OpenQASM' ⟨versionSpecifier⟩ ';'

⟨statementOrScope⟩ ::= ⟨statement⟩ | ⟨scope⟩

⟨scope⟩ ::= '{' ⟨statementOrScope⟩ * '}'

⟨statement⟩ ::= ⟨pragma⟩
  | ⟨annotation⟩* (
    | ⟨ioDeclarationStatement⟩
    | ⟨gateStatement⟩
    | ⟨gateCallStatement⟩
    | ...
  )

⟨annotation⟩ ::= ⟨slmDefinition⟩
  | ⟨aodDefinition⟩
  | ⟨atomBind⟩
  | ⟨trapTransfer⟩
  | ⟨aodShuttle⟩
  | ⟨raman⟩
  | ⟨rydberg⟩
  | ⟨annotationKeyword⟩ ⟨remainingLineContent⟩?

⟨slmDefinition⟩ ::= '@slm' ⟨trapPositions⟩

⟨trapPositions⟩ ::= '[' ⟨position⟩ ( ',' ⟨position⟩ )* ']'

⟨position⟩ ::= '(' ⟨float⟩ ',' ⟨float⟩ ')'

⟨aodDefinition⟩ ::= '@aod' ⟨aodRows⟩ ⟨aodColumns⟩

⟨atomBind⟩ ::= '@bind' ⟨identifier⟩
    ( 'slm' ⟨integer⟩ | 'aod' ⟨integer⟩ ⟨integer⟩ )

⟨atomTransfer⟩ ::= '@transfer' ⟨integer⟩
    '(' ⟨integer⟩ ',' ⟨integer⟩ ')'

⟨aodShuttle⟩ ::= '@shuttle'
    ( 'row' | 'column' ) ⟨integer⟩ ⟨integer⟩

⟨raman⟩ ::= '@raman global' ⟨float⟩ ⟨float⟩ ⟨float⟩
  | '@raman local' ⟨identifier⟩ ⟨float⟩ ⟨float⟩ ⟨float⟩

⟨rydberg⟩ ::= '@rydberg'

⟨...⟩ ::= ...
```

Figure 4. Abstract grammar for wQasm in EBNF format. Note that the non-terminals highlighted in purple are renamed from the OpenQASM grammar for simplification purposes. Their definitions, the remaining rules, and the full version of the OpenQASM grammar can be found in OpenQASM specifications [17, 19, 67].

that instruction (post-condition). The proposed annotations are summarized in Table 1.

Initialization of SLM Traps (`@slm`). Configures a fixed layer of atom traps at specified coordinates. Each coordinate (x_i, y_i) represents the position of an atom trap in a 2D plane.

- *Pre-condition:* Input coordinates need to be at a minimum distance from each other to avoid unwanted qubit inference. This distance is usually between 5 to 10 micrometers.
- *Post-condition:* Fixed layer of atom traps is initialized with traps on locations (x_i, y_i).

Initialization of AOD Traps (`@aod`). Set up a reconfigurable-layer of atom traps. It takes the coordinates arrays `x` and `y`, allowing to move trap locations. In contrast to the fixed layer layout, where atoms can be placed in arbitrary locations, the AOD layer is always in a grid-like format.

Table 1. Annotations list for extending OpenQASM to FPQA technology.

Annotation	Arguments	Description	Pre-condition	Post-condition
`@slm`	$[(x_0, y_0), \ldots, (x_n, y_n)]$	Distribute SLM atom trap locations on input coordinates	$\texttt{Dist}((x_i, y_i), (x_j, y_j)) > \texttt{Dist}_{min}$, $i \neq j$	$(\texttt{SLMX}_i, \texttt{SLMY}_i) = (x_i, y_i)$
`@aod`	$[x_0, \ldots x_n]$ $[y_0, \ldots, y_n]$	Setup AOD atom-trap grid based on input coordinates	$\texttt{Dist}((x_i, y_i), (x_j, y_j)) > \texttt{Dist}_{min}$ $x_n < x_{(n+1)}$	$(\texttt{AODX}_i, \texttt{AODYW}_i) = (x_i, y_i)$
`@bind`	q_{id} slm/aod $\texttt{slm}_{index}/(\texttt{aod}_x, \texttt{aod}_y)$	Bind qubit-indexes in SLM with index or AOD atom-traps at x and y	None	$(\texttt{AODX}_{index}, \texttt{AODY}_{index}) \leftarrow q_{id}$ or $(\texttt{SLMX}_i, \texttt{SLMY}_i) \leftarrow q_{id}$
`@transfer`	$\texttt{slm}_{index}$ $(\texttt{aod}_x, \texttt{aod}_y)$	Transfer atom from SLM trap at index to AOD at x and y	$\texttt{Dist}(\texttt{SLM}_{index}, (\texttt{AODX}, \texttt{AODY})) < \texttt{Dist\_Transfer}_{Max}$	(AODX, AODY)' <-- $\texttt{SLM}_{index}$ $\texttt{SLM}_{index}$' <-- (AODX, AODY)
`@shuttle`	row/column index offset	Move AOD row or column at index by offset	$\texttt{Dist}(\texttt{AODX}_i, \texttt{AODX}_{i+1}$ OR $\texttt{AODX}_{i-1}) > \texttt{Dist}_{min}$ $\texttt{Dist}(\texttt{AODY}_i, \texttt{AODY}_{i+1}$ OR $\texttt{AODY}_{i-1}) > \texttt{Dist}_{min}$	$\texttt{AODX}_{index}$ + offset OR $\texttt{AODY}_{index}$ + offset
`@raman local`	q_{id} (x,y,z)	Rotate q_{id} with angles x, y and z	$q_{id} \neq$ None	R(x, y, z) $\lvert q_{id} \rangle$
`@raman global`	(x,y,z)	Rotate all qubits with angles x, y and z	None	R(x, y, z) $\lvert q_{id} \rangle$ $\forall$ id
`@rydberg`	None	Apply CZ on qubits closer than the Rydberg distance	None	If $\texttt{Dist}(q_i, q_j) \leq$ Rydberg_dist: CZ $\lvert q_i\ q_j \rangle$ $\forall$ i, j; i≠j

- *Pre-condition:* Values for the x and y coordinates must be in increasing order, and a minimum distance must be preserved between adjacent rows and columns.
- *Post-condition:* AOD layer is initialized with rows and columns on coordinates given by x and y, respectively.

Binding atoms to qubit IDs (`@bind`). Physical traps are bound to unique qubit IDs; needed for the compilation process.

- *Pre-condition:* No pre-conditon.
- *Post-condition:* Atom at `slm_index` on a SLM layer, or `aod_x`, `aod_y` on a AOD layer is now tied to qubit `q_id`.

Atom transfer between layers (`@transfer`). Transfers an atom from the SLM layer to the AOD layer, or vice versa.

- *Pre-condition:* Destination atom trap needs to be empty for the atom to be moved onto. The involved atom traps need to be close enough.
- *Post-condition:* An atom is moved between layers.

Moving the reconfigurable layer (`@shuttle`).

This instruction applies a move operation, either to a row or column.

- *Pre-condition:* Adjacent rows and columns need to maintain a minimum distance (5-10 micrometers). The target row/column should not move over any other row/column.
- *Post-condition:* Row or column `index` moved by `offset`.

Local Raman pulse (`@raman local`). Applies a Raman local pulse on q_{id}.

- *Pre-condition:* Qubit with ID q_{id} exists.
- *Post-condition:* Qubit with ID q_{id} is rotated by x, y and z around the axis x, y and z, respectively.

Global Raman pulse (`@raman global`). Applies a global Raman pulse with rotations (x,y,z).

- *Pre-condition:* No pre-condition.
- *Post-condition:* All qubits from both layers are rotated by `(x, y, z)` around the x, y and z axis.

Rydberg pulse (`@rydberg`). Applies a global Rydberg pulse.

- *Pre-condition:* No pre-condition.
- *Post-condition:* All atoms within a Rydberg radius of each other are applied a multi-qubit controlled Z gate.

5 wOptimizer Optimizer

The wOptimizer module focuses on optimizing FPQA device-targeted circuits, particularly QAOA circuits addressing combinatorial optimization challenges like the Travelling Salesman Problem, Maximum Cut Problem (MaxCut), or Maximum 3-Satisfiability Problem (MAX-3SAT), crucial in operations research and cost reduction in sectors such as logistics [74].

The Quantum Approximate Optimization Algorithm is a prominent algorithm in the NISQ era, favored for its minimal connectivity needs. It's considered influential towards achieving practical quantum supremacy [85] due to its applicability on NISQ-era hardware. QAOA circuits consist of three parts: initialization of a mixer Hamiltonian, the time evolution of a cost Hamiltonian embedding a given optimization problem, and the time evolution of the mixer Hamiltonian itself. Our optimizations target the implementation of the time evolution of the cost Hamiltonian, as introduced below.

Our approach. wOptimizer focuses on MAX-3SAT problems for the FPQA architecture. This decision is not restrictive, as all NP problems can be reduced to MAX-3SAT. Additionally, the optimization techniques we introduce here can be adapted to general QAOA circuits with some restrictions.

Workflow using an example. Figure 5 illustrates an example formula where the cost Hamiltonian aims to maximize satisfied clauses. An example clause, $(\neg x_0 \vee \neg x_1 \vee \neg x_2)$, is represented as $f(x_0,x_1,x_2) = -x_0x_1x_2$. The overall formula aggregates these clause-specific objective functions into a Boolean polynomial, limiting terms to cubic degrees. For quantum circuit implementation, terms convert to z-axis rotations via a *CNOT* ladder configuration, depicted in Figure 6.

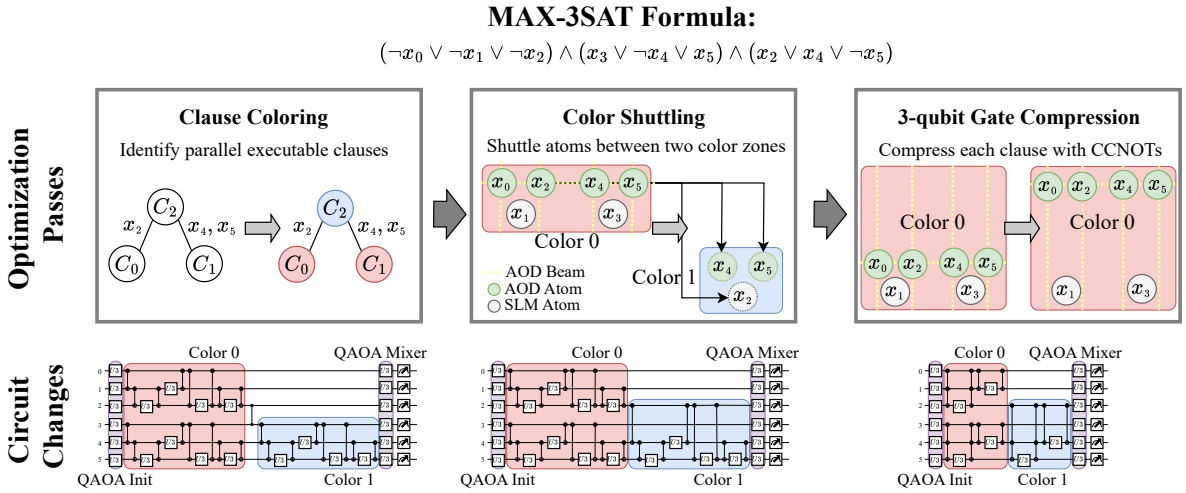

Figure 5. Overview of the optimization passes done by wOPTIMIZER: *clause coloring*, *color shuttling*, *3-qubit gate compression*. *Clause coloring* identifies the clauses that can be executed in parallel to utilize the global lasers in FPQA. *Color shuttling* eliminates the swap overhead between executing two different colors. *3-qubit gate compression* shortens the subcircuit fragments for each clause from 8 *CNOT* to 2 *CNOT* and 2 *CCNOT* gates. This reduces the total number of required pulses, as each *CNOT*/*CCNOT* gate can be implemented with a *CZ*/*CCZ* gate that the FPQA natively supports.

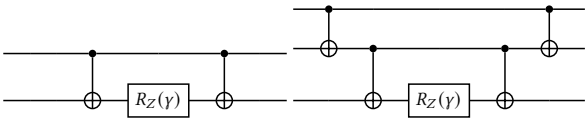

Figure 6. Compilation of terms in the objective polynomial function of MAX-3SAT formula. **(6a)** *Circuit fragment for a quadratic term.* **(6b)** *Implementation of a cubic term.* Terms with single variables are not shown in this figure, but they are simply compiled as a R_Z gate with angle γ.

Algorithm 1 Pseudocode for clause coloring.

```
Input: F ← List of clauses // [[-1, -2, -3], [4, -5, 6], [3, 5, -6]]
Output: C ← Color assignments // [0, 0, 1]
  n ← Number of clauses
  V ← List of clauses // [0, 1, 2]
  E[n][n] ← Initialized to zero // Adjacency matrix
  for C_i, C_j in F and C_i ≠ C_j do
      if C_i ∩ C_j ≠ ∅ then
          E[i][j] = 1
      end if
  end for
  return C ← DSatur(V, E)
```

5.1 WEAVER Optimization Overview

wOPTIMIZER consists of three stages, each targeting a different goal. Figure 5 shows the overview of the optimizations. Each stage takes advantage of FPQA's unique hardware features to develop an optimized circuit. These stages are:

- **Clause coloring:** Identifies independent clauses to maximize the parallelization capabilities of FPQAs.
- **Color shuttling:** Instead of a swap gate-based routing approach, we use the shuttling mechanism to eliminate the swap overhead in the circuit execution completely.
- **3-qubit gate compression:** Multi-qubit gates in each clause get compressed into fragments with 3-qubit gates that are natively supported by FPQAs, reducing the overall circuit depth and gate count.

The following sections illustrate the effects of each stage with the running example in Figure 5.

5.2 Clause Coloring

Challenge. To offset FPQA's slow gate execution times, we maximize parallelization by increasing multi-qubit gate operations with Rydberg pulses. In MAX-3SAT, we notice that the cost Hamiltonian circuits of two independent clauses can be executed in parallel. Thus, we can divide the formula into independent clause clusters to decrease their total number.

Key idea. Clauses' relationships are represented as an undirected graph, with edges between clauses sharing a variable. Clustering becomes a graph coloring problem, where each node is assigned a color so that two neighboring nodes always get assigned different colors. Thus, clauses with the same color can be executed in parallel. For example, in Figure 5, we run the independent clauses C_0 and C_1 simultaneously, followed by the intersecting third clause C_2.

Algorithm. Graph coloring is a well-studied problem. The optimal solution is the one that requires the least colors. The NP-hard nature of the problem makes this task challenging. Brute-force solutions are intractable as the search space increases exponentially. We settle for a heuristical algorithm, DSatur, with quadratic complexity and quality results. Based on DSatur [11] greedy coloring approach, Algorithm 1 shows its implementation using the example in Figure 5.

Algorithm 2 Pseudocode for color shuttling

Input: $S \leftarrow$ Ordered list of atoms in the current color zone, $F \leftarrow$ ordered list of atoms in the next color zone
Output: $R \leftarrow$ List of shuttling instructions

```
R ← [ ]
for a ∈ S do
    if a ∈ F then
        transfer_to_aod(a) // Used in next color
    else
        transfer_to_slm(a) // Unused atom
    end if
end for
while there is an atom a_i ∈ F that is not yet scheduled do
    W ← {a_i} // Current shuttle set
    for a_j ∈ F and a_j not yet scheduled do
        if Order between a_i and a_j is same in S and F then
            W.add(a_j) // shuttle in parallel
        end if
    end for
    R.add(create_shuttle(W))
end while
return R
```

5.3 Color Shuttling

Challenge. QAOA circuits require arbitrary 2-qubit connections; connections that lie on two unconnected qubits are normally implemented in superconducting with *SWAP* gates executed through 3 *CNOT* gates [63], worsening the fidelity loss. In FPQAs, the non-physical connection of qubits and the availability of AOD traps enable qubit reconfiguration. This hardware feature motivates the replacement of logical routing operations with physical ones using qubit shuttling.
Key idea. FPQAs' qubit reconfiguration must follow specific constraints that prevent arbitrary atom movement, making the formulation of a strategy for general quantum circuits tough. However, the clause coloring stage provides a clear conceptual strategy, as depicted in Figure 5.

Each color group requires two AOD columns and one AOD row, and they are executed sequentially. Color groups are set diagonally to avoid AOD constraints described in Section 4. After completing a color, the atoms for the upcoming color are transferred to AOD traps and shuttled to their next locations.

Parallel shuttling is a simple task; as long as the order between atoms in a color zone is kept in the AOD row, they can be shuttled in parallel without violating any AOD constraint. In our example, after executing the first color, the order of the atoms in the AOD row will be $x_2 > x_4 > x_5$. However, the new expected order is $x_4 > x_2 > x_5$ in the next color zone. This requires a two-step shuttle: first, transferring x_4 and x_5 together, followed by x_2, to proceed with the next zone.
Algorithm. The Color shuttling stage processes atom orders within and between color zones, transferring atoms to the next diagonal zones, as described previously and detailed in Algorithm 2. The implementation of the shuttling instruction is not shown on the pseudocode, which is trivial if the order of clauses within a color is fixed before compilation time.

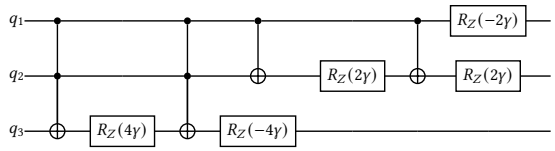

Figure 7. Implementation of the first clause ($\neg x_0 \vee \neg x_1 \vee \neg x_2$) from the example in Figure 5 (§ 5). While the CCNOT part implements the terms x_0x_2, x_1x_2 and $x_0x_1x_2$, the rest of the circuit computes the missing single variable terms and the quadratic term x_0,x_1.

5.4 3-qubit Gate Compression

Challenge. Using multi-qubit gates is challenging because most gate compilers focus on 2-qubit gate decompositions. Moreover, higher-order gates' fidelity is less than lower-order ones. We must ensure that enough lower-order gates are compressed in the circuit to offset the worse fidelity of the higher-order gate. Additionally, research mainly targets compilation to *CNOT* gates. However, FPQAs use C_nZ gates.
Key idea. MAX-3SAT requires 3-qubit interactions. The key idea is to implement each clause with *CCNOT* gates instead of solely *CNOT* gates. However, *CCNOT* gates are not enough to implement all the terms in the clause, as shown in Figure 7, due to the symmetric behavior of the *CCNOT* gate regarding its control qubits. The interactions between the control qubits must be implemented with a separate *CNOT* ladder. A 3-qubit gate is, thus, implemented with 2 *CCNOT* and 2 *CNOT* gates, a more efficient approach compared to using 8 *CNOT* gates.
Algorithm. Gate compression largely depends on the difference of fidelity parameters of *CZ* and *CCZ* gates on the target hardware. The compression stage first determines whether using the compression is beneficial; if so, it identifies the appropriate sub-circuit for each clause in the current color zone, which depends on the number of negative/positive literals in the clause. For clauses with mixed literals, the control bits in the *CCNOT* gate are set to zero with single-qubit rotation gates. Other cases can be corrected by adjusting the signs of the angles as in Figure 7. The algorithm keeps track of each clause's control qubits throughout the execution of the formula. This helps us to calculate the term between the control qubits with a single 2-qubit interaction instead of repeated applications. The actual implementation of the sub-circuit can be seen in Figure 5. Firstly, the atoms in a clause are positioned in a triangular layout employing a global Rydberg pulse for the *CCNOT* section, and then the control qubits are repositioned to facilitate or avoid additional interactions. Afterward, the control qubits are shuttled apart from the target qubit to implement the missing quadratic interaction. An additional shuttle can be applied to control qubits to prevent unnecessary quadratic terms.

5.5 Complexity Analysis

The complexity of Weaver's wOptimizer is bound by the clause coloring procedure, which follows the $O(N^2)$ complexity of DSatur [11]. The color shuttling step loops through the

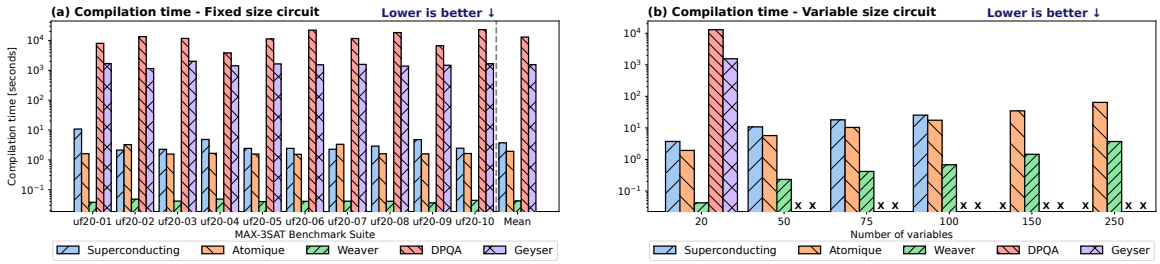

Figure 8. Compilation time (§ 8.2). **(a)** Compilation time for fixed-size circuits of 20 variables. **(b)** Compilation time for variable size circuits. Geyser and DPQA timed out above 20 variables. Superconducting was run with up to 100 variables, limited by the 127-qubit available backends.

traps in the AOD from end to start, checks if the atom is used for the next color group, and schedules a corresponding shuttling instruction. If a previous shuttling operation exists and it does not collide with the current one, they are then merged together and executed in parallel. The first input to Algorithm 2, i.e., the current positions of the atoms, is calculated by the previous coloring step. Since we loop the atoms through the AOD traps, it is guaranteed that the order is preserved, which is a hard constraint for Algorithm 2. Therefore, the color-shuttling procedure requires no additional sorting. As the coloring procedure precomputes execution sites (the second input for Algorithm 2) and the colors are sorted once with $O(N\log N)$ complexity, shuttling a single color takes $O(N)$, resulting in $O(N^2)$ complexity for at most $O(N)$ colors. 3-qubit gate compression for a single color takes $O(N)$ time, as all the individual shuttling instructions, Raman, and Rydberg pulses can be implemented in constant time. Similar to color shuttling, since there are at most $O(N)$ colors, the overall complexity of 3-qubit gate compression also becomes $O(N^2)$. Thus, wOPTIMIZER's complexity is determined by the $O(N^2)$ complexity of the coloring step for N variables/qubits/atoms.

6 wCHECKER Equivalence Checker

The wCHECKER checks the functional equivalence between the nativized and the FPQA-optimized circuits. To achieve this, it checks the wQASM file, which contains hardware-level instructions for each logical gate and hardware-specific Rydberg and Raman pulses for FPQAs, to see if these pulse instructions can mirror their logical gate equivalent. To maintain logical circuit integrity, (global or local) Raman pulses can be verified by comparing their pulse angles and addresses to their logical specifications, as outlined in § 4. Conversely, Rydberg pulses are more challenging to check, given that we need to know the positions of the atoms inside the FPQA beforehand. As such, wCHECKER must simulate the atom movements (atom transfers and shuttle instructions) before each Rydberg pulse instruction. The SLM and AOD initialization instructions at the start of a wQASM file determine the number of atoms and their starting positions in the system.

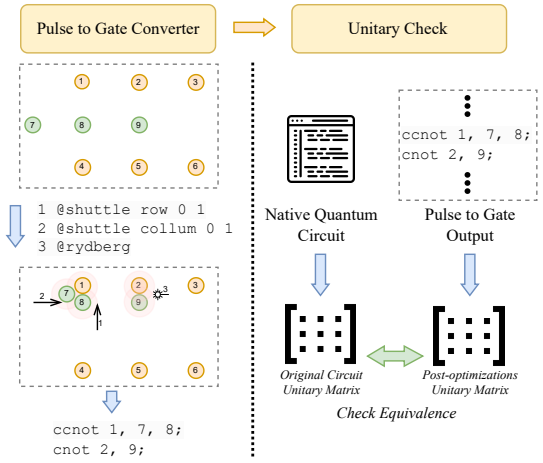

Figure 9. Example workflow of the wCHECKER (§ 6). wCHECKER comprises a pulse-to-gate converter and a unitary checker. Initially, three pulse instructions are translated into their respective gate instructions. A `@shuttle` instruction moves the row with qubits 7, 8, and 9 up, and a second instruction moves the column with qubit 7 to the right. A Rydberg pulse applies *ccnot* to 1, 7, 8, and *cnot* to 2, 9. Then, a native quantum circuit is converted to a unitary matrix with gate instructions for the comparison of two unitary gates.

Figure 9 shows wCHECKER in action, specifically the verification of a logical CCZ gate between the qubits 1, 7, and 8. wCHECKER sequentially simulates the two shuttle instructions and then checks each atom pair according to the new positions of the atoms. For correct compilation, wCHECKER has to verify that **(1)** the atoms 1, 7, and 8 are interacting with each other, **(2)** they are in equal distance to each other, **(3)** no other atoms are currently interacting with any other atoms.

The complexity of wCHECKER is $O(N^2M)$ steps for a circuit with N qubits and M instructions, bound by Rydberg pulses, which require checking each qubit pair for interactions, causing quadratic scaling with qubits. Each remaining instruction can be checked in at most $O(N)$ time. For example, validating a shuttling instruction involves looping through AOD columns/rows to ensure no crossover with neighbors. Likewise, single-qubit gates, i.e., Raman pulses, can be checked in $O(1)$ time.

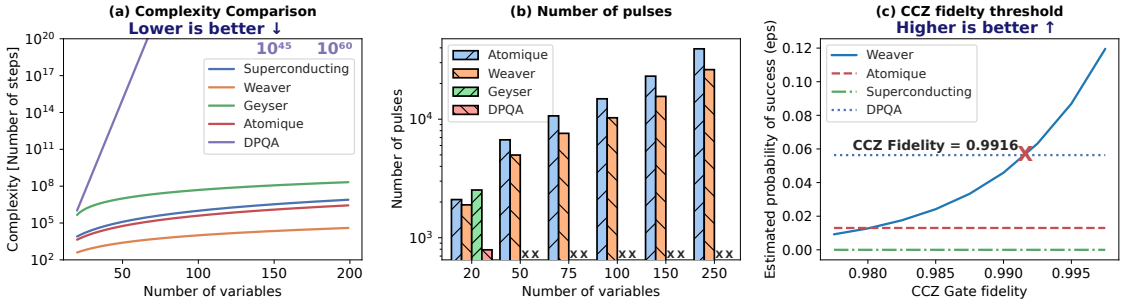

Figure 10. Performance comparison between Weaver and baselines (§ 8). **(a)** Visual representation of compilation complexity of each system, also represented on Table 2. **(b)** Comparing the average number of gates in the solutions generated by each system. **(c)** Threshold complexity of CCZ gate where Weaver's solution's fidelity surpasses all baselines.

7 Implementation

We implement Weaver on top of Qiskit 1.0.2 [39] and OpenQASM 3.0 [17], and use PySAT 3.2.0 [38] for the wOptimizer module. The native gate synthesis in Figure 3 is done by the Qiskit compiler by setting the appropriate basis gate set, $B = \{U3, CZ\}$. To retarget a wQasm file for a superconducting device, we use the Qiskit compiler [77].

wOptimizer uses PySAT to handle MAX-3SAT formulas and represents the FPQA device as a class with adjustable hardware parameters for compilation. Since FPQAs are still emerging, Weaver aims to remain hardware-agnostic, supporting flexible SLM trap layouts and varying numbers of AOD rows and columns. A key simplification is that it assumes only digital computation; for example, if three atoms are within range and a Rydberg pulse is applied, it treats the operation as a *CCZ* gate, which is accurate only if the atoms are equidistant. If not, the operation could correspond to a different multi-qubit gate.

wChecker begins by parsing the wQasm file using Qiskit's OpenQASM passes, then processes each FPQA annotation in a visitor pattern to reconstruct the FPQA class created by Weaver during compilation. Each FPQA instruction shares a basic interface for validating the operation and generating the equivalent wQasm instruction with annotations. As detailed in § 6, wChecker compares the FPQA annotations against the logical circuit in the output wQasm file.

8 Evaluation

We evaluate Weaver across three dimensions: (a) compilation time (§ 8.2), (b) execution time (§ 8.3), and (c) fidelity (§8.4).

8.1 Experimental Methodology

Experimental setup. We run Weaver and all the baselines on a server with a 64-core AMD EPYC 7713P processor and 1TB of DDR4 memory. We use *IBM Washington* [36] as our superconducting backend model.

Evaluation metrics. We evaluate **(1)** compilation time in seconds, **(2)** execution time in seconds, and the number of pulses generated, and **(3)** fidelity as the Estimated Probability of Success (EPS) [82].

Benchmarks. We use the MAX-3SAT formulas from the SATLIB benchmark [66]. These formulas define statements with varying numbers of clauses and variables. More clauses lead to longer quantum circuits, while the number of variables corresponds to the number of qubits.

Experimental methodology. We run **(1)** 10 different MAX-3SAT problems with the fixed size of 20 variables and **(2)** with an increasing number of variables, where each data point is the average of the 10 SAT problems of that size.

Framework and configuration. We use Qiskit version 1.0.2 with FPQA hardware parameters from [83] and [26] based on Rubidium atoms. We set a timeout of 20 hours for the evaluated compilers to find a solution.

Baselines. We compare Weaver to state-of-the-art FPQA compilers: *Geyser* [68], *Atomique* [102], and *DPQA* [94]. For superconducting, we use *Qiskit* [77] compiler.

8.2 Compilation Time

RQ1: *What is Weaver's performance w.r.t. compilation time, compared to the baselines?*

Hypothesis. We measure the end-to-end compilation time of Weaver and the baselines using the `time` library. Weaver should achieve lower compilation times than the baselines due to lower computational complexity, as shown in Table 2. Weaver scales quadratically as the number of variables increases, while the baselines scale cubically or with the number of quantum operations, which are generally significantly more than the number of variables.

Results. Figure 8 (a) shows the compilation time for 10 benchmarks with a size of 20 variables. Figure 8 (b) shows the results of increasing the circuit size of the MAX3SAT problems from 20 up to 250 variables. Weaver consistently finds solutions $5.7x10^3\times$ faster than all other systems. *Geyser* and *DPQA* are the most time-intensive systems, on average $1.5x10^3\times$ slower than Superconducting, *Atomique* and *Weaver*.

Analysis. Figure 10 (a) shows the compilation complexity comparison as the number of steps. Geyser's complexity is based on the number of quantum operations on the benchmark circuit. For this, 6 circuits with sizes from 20 to 250 were used to find a fitting function to express Geyser's complexity

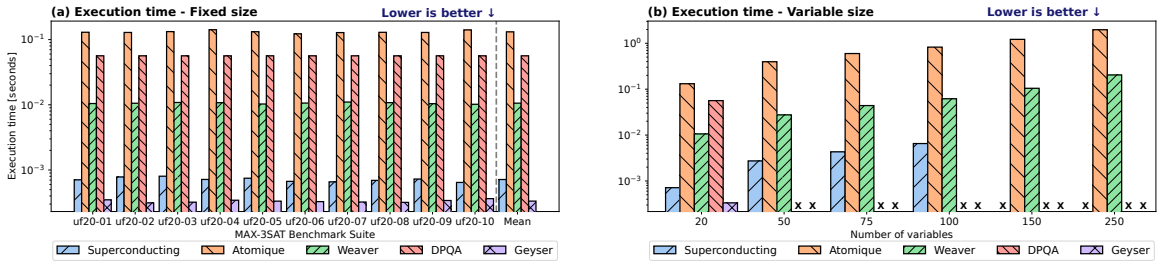

Figure 11. Execution time (§ 8.3) **(a)** Execution time for fixed-size circuits of 20 variables. **(b)** Execution time for increasing benchmark sizes. Geyser and DPQA timed out above 20 variables. Superconducting was run with up to 100 variables, limited by the 127-qubit available backends.

Table 2. Compilation complexity comparison (§ 8.2). *N is the number of benchmark variables, and K is the number of quantum circuit operations (generally, $K \gg N$). The complexity of Qiskit and Atomique stem from Sabre [51].*

Compiler	Computational complexity
Qiskit [77]	$O(N^3)$
Atomique [103]	$O(N^3)$
Geyser [68]	$O(K^2)$
DPQA [94]	$O(2^K)$
Weaver	$O(N^2)$

based on the number of variables. Notably, Weaver exhibits the lowest complexity as the number of variables increases.
RQ1 takeaway. Weaver's heuristics achieve faster compilation times than the baselines, up to 24× lower than Atomique; up to 10^5× lower than DPQA and 10^4× lower, on average.

8.3 Execution Time

RQ2: *What is Weaver's performance w.r.t. execution time, compared to the baselines?*
Hypothesis. We measure how long the quantum circuit runs on a quantum device by adding the times of each pulse and shuttling operation, considering the maximum movement speed. Typically, longer circuit duration indicates a higher chance for decoherence errors [51]; hence, lower is better.

Minimizing execution times is challenging due to the NP-hard nature of finding optimal solutions at various stages in compilation [15] and Weaver aims to balance compilation time and execution time. We expect circuits compiled with Weaver to have lower execution times since Weaver increases parallelization, as detailed in § 5.2.
Results. Figure 11 (a) shows the execution time of Weaver's solutions compared to the baselines. Weaver consistently achieves 8.2× better execution times than *Atomique* and *DPQA*. Superconducting (*Qiskit*) has faster quantum gate times than FPQAs, which explains the results.

Although Geyser incurs the lowest execution times, it times out for problems larger than 20 variables, as shown in Figure 11 (b).
Analysis. Figure 10 (b) presents the mean number of laser pulses. FPQA pulses for atom shuttling are time-consuming, given that the movement must be relatively slow to avoid losing the atom or inducing noise. Geyser does not use atom movement in its solutions, which explains its fast solutions while showing many pulses. DPQA applies the most atom movement of all systems, which explains the low number of pulses but longer execution time.
RQ2 takeaway. Weaver's parallelization strategy and the low number of pulses reduce execution times by 12.4× in the best case and 4.4× on average, compared to the baselines.

8.4 Fidelity

RQ3: *What is Weaver's performance w.r.t. EPS, compared to the baselines?*
Hypothesis. EPS measures the likelihood that a circuit runs correctly in one execution, calculated by accumulating the errors of each pulse operation. Weaver aims to improve EPS compared to the baselines, using heuristics to tackle this NP-hard problem and balancing between compilation time and the EPS of the solutions found.
Results. Figure 12 shows the EPS of Weaver solutions compared to the baselines. Weaver consistently achieves better EPS than other baselines except for *DPQA*, which finds a better solution for 20 variables benchmarks. Increasing the benchmark sizes increases Weaver's improvement on EPS over its closest competitor, *Atomique*.
Analysis. Weaver's approach leverages 3-qubit gates (CCZ) despite their current susceptibility to higher error rates. This distinguishes Weaver from most baseline approaches, which avoid 3-qubit gates to minimize error. Figure 10 (c) illustrates the threshold of the CCZ gate fidelity required for Weaver to outperform all pictured baselines on a 20-variable benchmark. The threshold of 0.9916 is only a 1.2% improvement over the currently used CCZ error of 0.98.
RQ3 takeaway. Weaver achieves on average 10% improvement in EPS when compared to Atomique, evaluated on 20 variable benchmarks, while for a larger benchmark of 150 variables, the improvement is on the order of 10^8×.

9 Related Work

Hardware-independent compilers and optimizations. This is an active area of research, and the following non-exhaustive list features universally applicable compiler optimizations (even when framed for superconducting QPUs)

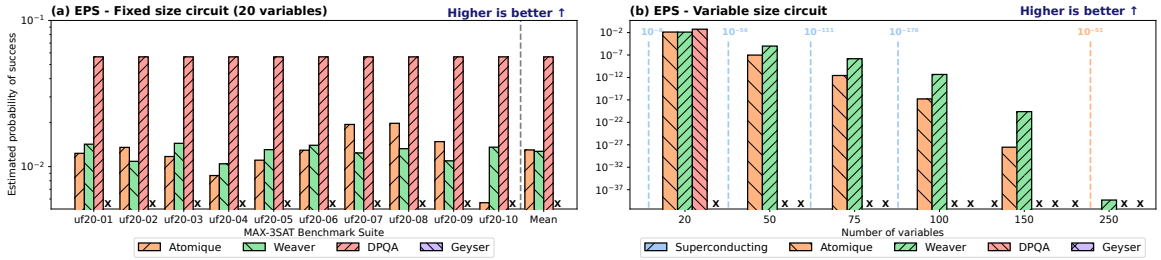

Figure 12. Fidelity as EPS (§ 8.4). **(a)** Execution time for fixed-size circuits of 20 variables. **(b)** Execution time for increasing benchmark sizes. Geyser was not considered since the block approximation step causes EPS computation to be unfair. DPQA timed out above 20 variables.

[10, 21, 23, 31, 52, 53, 56, 72, 84, 86, 90, 96, 107]. Notably, this work is orthogonal to Weaver and can be an additional compiler stage. For instance, the optimizations of *Paulihedral* [52] can be applied before Weaver, while readout error mitigation [23, 84, 96] is applicable as a post-processing step.

Optimizations for superconducting QPUs. Superconducting QPUs are extensively studied and their optimizations and can be categorized as follows: (1) qubit mapping and routing [40, 51, 60, 61, 70, 71, 89, 95, 97, 106, 108, 109], (2) instruction/pulse scheduling [13, 22, 55, 62, 91, 99], and (3) error suppression/mitigation [23, 58, 69, 96]. However, such work does not apply to FPQAs due to its specificity to superconducting's limitations (i.e., limited connectivity and short coherence times), while the unique features of FPQAs as described in § 2.3 directly address these limitations.

Compilers for neutral atoms/FPQAs. Compiler frameworks and optimizations for neutral atoms are an emerging area of research. Implementing these compilers to leverage the capabilities of neutral atoms, as discussed in § 2.3, presents a set of challenging problems. **(1)** While qubit shuttling is utilized in several studies [9, 64, 93, 103, 104], not all neutral atom compilers take advantage of atom shuttling; some systems opt for a fixed atom grid and rely on SWAP operations [7, 54, 68]. **(2)** Supporting 2+ qubit gates is another challenging capability in neutral atom architectures, primarily due to the difficulties in synthesizing quantum circuits to utilize gates with more than two qubits. Currently, only Geyser [68] achieves this by approximating blocks of the original circuit to a template sequence composed of unitary and Toffoli gates. Other compilers can map and route circuits with multi-qubit gates if they are included in the quantum circuit in the first place [7, 54, 82]. In contrast, Weaver leverages all FPQA capabilities, including both qubit shuttling and three-qubit gates, making it the first compiler, to our knowledge, to do so.

Retargetable quantum compilers. Research on retargetable compilers is growing since different quantum technologies provide different advantages that are beneficial to different applications. $t|ket\rangle$ [90] is able to target different quantum superconducting providers (IBM Q, Rigetti, ProjectQ, etc.) while sharing circuit optimizations before the circuit is compiled to a certain vendor's API. On the other hand, XACC [59] provides support for different quantum technologies and tries to share optimizations when possible. This approach may not be optimal since applying architecture-specific optimizations can benefit the quantum circuit performance. Weaver targets superconducting and neutral atoms technology through a common input QASM file but applies distinct optimizations to each technology.

Domain-specific optimizations. There exist optimizations leveraging the distinctive features of particular algorithms and/or circuit structures to improve fidelity beyond the capabilities of general-purpose compilers [2, 3, 6, 20, 30, 34, 41, 47, 79, 80, 92, 100, 105]. However, the listed work above either (1) is orthogonal, i.e., can be applied as a pre/post-processing step [6, 34, 92] or targets a different domain ([47]), (2) is specific to superconducting QPUs ([2]) or (3), focuses on compiler performance w.r.t. runtime only ([30]).

10 Conclusion

While superconducting devices are the leading quantum technology, they face scalability issues, which are addressed by the emerging and promising FPQA technology. To this end, we presented Weaver, a retargetable compiler framework that retargets existing superconducting code to FPQAs while leveraging their distinct capabilities, namely the dynamic qubit connectivity, higher-order gates, and high gate parallelism. Weaver compiles circuits $\approx 10^3\times$ faster, while the compiled circuits have 4.4× lower execution times and up to $10^5\times$ higher execution fidelity.

Acknowledgements

We would like to thank the anonymous reviewers for their valuable insights. This work was supported by the Bavarian State Ministry of Science and the Arts with funds from the Hightech Agenda Bayern Plus, as part of the Munich Quantum Valley (MQV) initiative (6090181).

A Artifact Appendix

A.1 Abstract

Our artifacts include Weaver's framework, focusing on Weaver's proposed FPQA optimization and all the other baseline FPQA compilers for comparison. These include Geyser [68], Atomique [103] and DPQA [94]. The artifact also includes code to run with Qiskit [77] for a Superconducting compiler baseline. Weaver's compilation procedure takes advantage of the 3 variable sets of the MAX-3SAT problems as explained in 5. This appendix provides the necessary information to install all the required Python libraries and reproduce the experiments presented in the paper. Our artifacts demonstrate Weaver's ability to achieve $10^3\times$ faster execution time than baseline approaches, as well as 4.4× better compilation time and 10% on average better fidelity.

A.2 Artifact Check-list (Meta-information)

- **Program:** Weaver FPQA optimization's source code. The source codes for Geyser, Atomique, and DPQA baseline compilers are included. The source code for the Qiskit compiler can be installed as a Python library.
- **Data set:** The MAX-3SAT problems used are included in the artifact and were taken from the SATLIB benchmark [66].
- **Hardware:** The artifact does not require any specific hardware or features.
- **Software:** Weaver and the other baseline compilers were tested on Python 3.11.2.
- **Execution:** The artifact needs to execute each of the baseline compilers for the given benchmark circuits. Weaver, Qiskit, and Atomique should each take no more than 1 hour to run all the benchmarks. However, Geyser, especially DPQA, might require up to 10 hours each to run all benchmarks.
- **Metrics:** Execution time, compilation time, and result fidelity are the main three metrics that were measured; additionally, to complement each of these metrics, we also measure algorithm complexity, number of pulses, and the CCZ fidelity threshold.
- **Output:** The artifact should output the 4 figures presented on the paper, which should be similar to the ones shown, with some negligible variation.
- **Experiments:** Preparing the experiments should take no more than 10 minutes; this involves downloading the repo and setting up the Python environment. The whole experiment can be left running in the background. However, it may take up to 24 hours to complete.
- **Publicly available:** Yes.
- **License:** MIT License. Weaver doesn't use any external license.
- **Archive:** Weaver's artifact is archived at: https://doi.org/10.5281/zenodo.14084047 [46]

A.3 Description

A.3.1 Artifact. All the artifact's source code can be found at https://github.com/TUM-DSE/weaver.

A.3.2 Benchmarks. The benchmarks used are taken from a collection of MAX-3SAT problem formulations used in MAX-3SAT solvers competitions. The problem formulations used are of the sizes 20, 50, 75, 100, 150 and 250. Each problem size has 10 variants with different combinations of variables.

A.4 Installation

All the required libraries are contained in the file `pyprojec t.toml`. To install the dependencies, it is suggested to create a virtual environment with any virtual environment Python framework or follow these steps:

1. `python -m venv .venv`
2. `pip install pdm`
3. `pdm install`

After `pdm` installs all the packages and dependencies, you can move to execute the `run` script.

A.4.1 Experiment Workflow. To reproduce the plots presented in the paper, one simply needs to run: `python run.py` on the main folder. The script will execute 7 scripts and finally plot 4 figures. The scripts are the following:

1. **MAX-3SAT to quantum circuit** [≈ 10 *minutes*]**:** Transpiles MAX-3SAT instances to QAOA quantum circuits with Qiskit and basis gates *RX, RZ, X, Y, Z, H, ID, CZ*. Since most baseline compilers get a quantum circuit as input, the transpilation from MAX-3SAT instances to the quantum circuit is done here to save time and keep experiment consistency.
2. **Atomique** [≈ 30 *minutes*]**:** Runs Atomique compiler on the previously transpiled quantum circuits. Atomique compiles quantum circuits of all sizes (20, 50, 75, 100, 150, 250) and 10 variant circuits for each size.
3. **Superconducting (Qiskit)** [≈ 20 *minutes*]**:** Runs Qiskit transpiler on the previously transpiled circuits. This script runs the compiler for sizes (20, 50, 75, and 100). It doesn't run for larger circuits since the targetted backend (*Washington*) is of size 127 qubits. Again, each size runs for 10 variants.
4. **Geyser** [≈ 7 *hours*]**:** Runs Geyser compiler for 10 different circuits of size 20 variables. Running Geyser on larger circuits takes longer than the defined timeout compilation time (20 hours).
5. **Weaver** [≈ 20 *minutes*]**:** Runs Weaver compilation procedure. Similarly to Atomique, Weaver also runs for all sizes of the benchmarks and all variants.
6. **Quantum circuit to DPQA format** [≈ 2 *minutes*]**:** Converts a quantum circuit to the format required by the DPQA compiler. The format is a `.json` file with sets of two-qubit gates. Expected duration $\tilde{1}0$ minutes.

7. **DPQA** [≈ 15 *hours*]**:** Runs the DPQA compiler. It runs 10 variants of benchmarks of size 20. For the same reason as Geyser, we didn't run the compiler on larger benchmarks.
8. **Plotting** [≈ 1 *minute*]**:** Finally, the script plots the 4 figures presented on the paper based on the results of the previous compilations: Figure 8, Figure 10, Figure 11, and Figure 12. The complexity comparison Figure 10 (b) and CCZ fidelity threshold Figure 10 (c) plots are the only ones that plot fixed lines and values pre-calculated.

A.5 Evaluation and Expected Results

Throughout the experiment, the script will output information about its progress; this is a hint that the script is still running and hasn't halted, also considering how long each step of the script takes. The output plots may have some variance when compared to the plots presented on the paper, however this variance is not expected to surpass 5%.

A.5.1 Methodology. Submission, reviewing, and badging methodology:

- http://cTuning.org/ae/submission-20190109.html
- http://cTuning.org/ae/reviewing-20190109.html
- https://www.acm.org/publications/policies/artifact-review-badging

References

[1] CS Adams and E Riis. 1997. Laser cooling and trapping of neutral atoms. *Progress in quantum electronics* 21, 1 (1997), 1–79.

[2] Mahabubul Alam, Abdullah Ash-Saki, and Swaroop Ghosh. 2020. Circuit Compilation Methodologies for Quantum Approximate Optimization Algorithm. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 215–228. https://doi.org/10.1109/MICRO50266.2020.00029

[3] Sashwat Anagolum, Narges Alavisamani, Poulami Das, Moinuddin Qureshi, and Yunong Shi. 2024. Elivagar: Efficient Quantum Circuit Search for Classification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (, La Jolla, CA, USA,) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 336–353. https://doi.org/10.1145/3620665.3640354

[4] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. 2019. Quantum supremacy using a programmable superconducting processor. *Nature* 574, 7779 (2019), 505–510.

[5] awsQuantum [n. d.]. AWS Bracket. https://aws.amazon.com/braket/. Accessed: 2022-04-11.

[6] Ramin Ayanzadeh, Narges Alavisamani, Poulami Das, and Moinuddin Qureshi. 2023. FrozenQubits: Boosting Fidelity of QAOA by Skipping Hotspot Nodes. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 311–324. https://doi.org/10.1145/3575693.3575741

[7] Jonathan M. Baker, Andrew Litteken, Casey Duckering, Henry Hoffmann, Hannes Bernien, and Frederic T. Chong. 2021. Exploiting Long-Distance Interactions and Tolerating Atom Loss in Neutral Atom Quantum Architectures. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 818–831. https://doi.org/10.1109/ISCA52012.2021.00069

[8] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. 2017. Quantum machine learning. *Nature* 549, 7671 (2017), 195–202.

[9] Sebastian Brandhofer, Ilia Polian, and Kevin Krsulich. 2023. Optimal Partitioning of Quantum Circuits using Gate Cuts and Wire Cuts. *arXiv preprint arXiv:2308.09567* (2023).

[10] Sergey Bravyi, Sarah Sheldon, Abhinav Kandala, David C. Mckay, and Jay M. Gambetta. 2021. Mitigating measurement errors in multiqubit experiments. *Phys. Rev. A* 103 (Apr 2021), 042605. Issue 4. https://doi.org/10.1103/PhysRevA.103.042605

[11] Daniel Brélaz. 1979. New methods to color the vertices of a graph. *Commun. ACM* 22, 4 (apr 1979), 251–256. https://doi.org/10.1145/359094.359101

[12] Lukas Burgholzer and Robert Wille. 2020. Advanced equivalence checking for quantum circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40, 9 (2020), 1810–1824.

[13] Jianxin Chen, Dawei Ding, Weiyuan Gong, Cupjin Huang, and Qi Ye. 2024. One Gate Scheme to Rule Them All: Introducing a Complex Yet Reduced Instruction Set for Quantum Computing. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (, La Jolla, CA, USA,) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 779–796. https://doi.org/10.1145/3620665.3640386

[14] Juan I Cirac and Peter Zoller. 1995. Quantum computations with cold trapped ions. *Physical review letters* 74, 20 (1995), 4091.

[15] Alexander Cowtan, Silas Dilkes, Ross Duncan, Alexandre Krajenbrink, Will Simmons, and Seyon Sivarajah. 2019. On the qubit routing problem. *arXiv preprint arXiv:1902.08091* (2019).

[16] Andrew Cross, Ali Javadi-Abhari, Thomas Alexander, Niel De Beaudrap, Lev S Bishop, Steven Heidel, Colm A Ryan, Prasahnt Sivarajah, John Smolin, Jay M Gambetta, et al. 2022. OpenQASM 3: A broader and deeper quantum assembly language. *ACM Transactions on Quantum Computing* 3, 3 (2022), 1–50.

[17] Andrew Cross, Ali Javadi-Abhari, Thomas Alexander, Niel De Beaudrap, Lev S. Bishop, Steven Heidel, Colm A. Ryan, Prasahnt Sivarajah, John Smolin, Jay M. Gambetta, and Blake R. Johnson. 2022. OpenQASM3: A Broader and Deeper Quantum Assembly Language. *ACM Transactions on Quantum Computing* 3, 3, Article 12 (sep 2022), 50 pages. https://doi.org/10.1145/3505636

[18] Andrew W. Cross, Lev S. Bishop, Sarah Sheldon, Paul D. Nation, and Jay M. Gambetta. 2019. Validating quantum computers using randomized model circuits. *Phys. Rev. A* 100 (Sep 2019), 032328. Issue 3. https://doi.org/10.1103/PhysRevA.100.032328

[19] Andrew W. Cross, Lev S. Bishop, John A. Smolin, and Jay M. Gambetta. 2017. Open Quantum Assembly Language. arXiv:1707.03429 [quant-ph] https://arxiv.org/abs/1707.03429

[20] Siddharth Dangwal, Gokul Subramanian Ravi, Poulami Das, Kaitlin N. Smith, Jonathan Mark Baker, and Frederic T. Chong. 2024. VarSaw: Application-tailored Measurement Error Mitigation for Variational Quantum Algorithms. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4* (, Vancouver, BC, Canada,) *(ASPLOS '23)*. Association for Computing Machinery, New York, NY, USA, 362–377. https://doi.org/10.1145/3623278.3624764

[21] Poulami Das, Eric Kessler, and Yunong Shi. 2023. The Imitation Game: Leveraging CopyCats for Robust Native Gate Selection in NISQ Programs. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 787–801. https://doi.org/10.1109/HPCA56546.2023.10071025

[22] Poulami Das, Swamit Tannu, Siddharth Dangwal, and Moinuddin Qureshi. 2021. ADAPT: Mitigating Idling Errors in Qubits via Adaptive Dynamical Decoupling. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (Virtual Event, Greece)

(MICRO '21). Association for Computing Machinery, New York, NY, USA, 950–962. https://doi.org/10.1145/3466752.3480059

[23] Poulami Das, Swamit Tannu, and Moinuddin Qureshi. 2021. JigSaw: Boosting Fidelity of NISQ Programs via Measurement Subsetting. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (Virtual Event, Greece) *(MICRO '21)*. Association for Computing Machinery, New York, NY, USA, 937–949. https://doi.org/10.1145/3466752.3480044

[24] Leonardo DiCarlo, Jerry M Chow, Jay M Gambetta, Lev S Bishop, Blake R Johnson, DI Schuster, J Majer, Alexandre Blais, Luigi Frunzio, SM Girvin, et al. 2009. Demonstration of two-qubit algorithms with a superconducting quantum processor. *Nature* 460, 7252 (2009), 240–244.

[25] Jürgen Eschner, Giovanna Morigi, Ferdinand Schmidt-Kaler, and Rainer Blatt. 2003. Laser cooling of trapped ions. *JOSA B* 20, 5 (2003), 1003–1015.

[26] Simon J. Evered, Dolev Bluvstein, Marcin Kalinowski, Sepehr Ebadi, Tom Manovitz, Hengyun Zhou, Sophie H. Li, Alexandra A. Geim, Tout T. Wang, Nishad Maskara, Harry Levine, Giulia Semeghini, Markus Greiner, Vladan Vuletic, and Mikhail D. Lukin. 2023. High-fidelity parallel entangling gates on a neutral atom quantum computer. *Nature* 622, 7982 (Oct. 2023), 268–272. https://doi.org/10.1038/s41586-023-06481-y arXiv:2304.05420 [cond-mat, physics:physics, physics:quant-ph].

[27] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. 2014. A Quantum Approximate Optimization Algorithm. arXiv:1411.4028 [quant-ph]

[28] fidelity-qiskit [n. d.]. Qiskit Hellinger fidelity. https://qiskit.org/documentation/stubs/qiskit.quantum_info.hellinger_fidelity.html. Accessed: 2022-04-11.

[29] Konstantinos Georgopoulos, Clive Emary, and Paolo Zuliani. 2021. Modeling and simulating the noisy behavior of near-term quantum computers. *Physical Review A* 104, 6 (2021), 062432.

[30] Pranav Gokhale, Yongshan Ding, Thomas Propson, Christopher Winkler, Nelson Leung, Yunong Shi, David I. Schuster, Henry Hoffmann, and Frederic T. Chong. 2019. Partial Compilation of Variational Algorithms for Noisy Intermediate-Scale Quantum Machines. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (Columbus, OH, USA) *(MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 266–278. https://doi.org/10.1145/3352460.3358313

[31] Pranav Gokhale, Ali Javadi-Abhari, Nathan Earnest, Yunong Shi, and Frederic T. Chong. 2020. Optimized Quantum Compilation for Near-Term Algorithms with OpenPulse. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 186–200. https://doi.org/10.1109/MICRO50266.2020.00027

[32] google-nisq-properties [n. d.]. Quantum Computer Datasheet. https://quantumai.google/hardware/datasheet/weber.pdf. Accessed: 2023-07-17.

[33] googleQuantum [n. d.]. Google Cirq. https://quantumai.google/cirq. Accessed: 2022-04-11.

[34] Tianyi Hao, Kun Liu, and Swamit Tannu. 2023. Enabling High Performance Debugging for Variational Quantum Algorithms Using Compressed Sensing. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (Orlando, FL, USA) *(ISCA '23)*. Association for Computing Machinery, New York, NY, USA, Article 9, 13 pages. https://doi.org/10.1145/3579371.3589044

[35] Loïc Henriet, Lucas Beguin, Adrien Signoles, Thierry Lahaye, Antoine Browaeys, Georges-Olivier Reymond, and Christophe Jurczak. 2020. Quantum computing with neutral atoms. *Quantum* 4 (2020), 327.

[36] IBM. [n. d.]. FakeWashington | IBM Quantum Documentation — docs.quantum.ibm.com. https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.fake_provider.FakeWashington. [Accessed 30-08-2024].

[37] ibmQuantum [n. d.]. IBM Quantum. https://www.ibm.com/quantum-computing/. Accessed: 2022-04-11.

[38] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. 2018. PySAT: A Python Toolkit for Prototyping with SAT Oracles. In *SAT*. 428–437. https://doi.org/10.1007/978-3-319-94144-8_26

[39] Ali Javadi-Abhari, Matthew Treinish, Kevin Krsulich, Christopher J. Wood, Jake Lishman, Julien Gacon, Simon Martiel, Paul D. Nation, Lev S. Bishop, Andrew W. Cross, Blake R. Johnson, and Jay M. Gambetta. 2024. Quantum computing with Qiskit. https://doi.org/10.48550/arXiv.2405.08810 arXiv:2405.08810 [quant-ph]

[40] Yuwei Jin, Fei Hua, Yanhao Chen, Ari Hayes, Chi Zhang, and Eddy Z. Zhang. 2024. Exploiting the Regular Structure of Modern Quantum Architectures for Compiling and Optimizing Programs with Permutable Operators. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4* (, Vancouver, BC, Canada,) *(ASPLOS '23)*. Association for Computing Machinery, New York, NY, USA, 108–124. https://doi.org/10.1145/3623278.3624751

[41] Yuwei Jin, Zirui Li, Fei Hua, Tianyi Hao, Huiyang Zhou, Yipeng Huang, and Eddy Z Zhang. [n. d.]. Tetris: A Compilation Framework for VQA Applications in Quantum Computing. ([n. d.]).

[42] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M Chow, and Jay M Gambetta. 2017. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *nature* 549, 7671 (2017), 242–246.

[43] Richard M Karp. 2010. *Reducibility among combinatorial problems*. Springer.

[44] P. V. Klimov, J. Kelly, Z. Chen, M. Neeley, A. Megrant, B. Burkett, R. Barends, K. Arya, B. Chiaro, Yu Chen, A. Dunsworth, A. Fowler, B. Foxen, C. Gidney, M. Giustina, R. Graff, T. Huang, E. Jeffrey, Erik Lucero, J. Y. Mutus, O. Naaman, C. Neill, C. Quintana, P. Roushan, Daniel Sank, A. Vainsencher, J. Wenner, T. C. White, S. Boixo, R. Babbush, V. N. Smelyanskiy, H. Neven, and John M. Martinis. 2018. Fluctuations of Energy-Relaxation Times in Superconducting Qubits. *Phys. Rev. Lett.* 121 (Aug 2018), 090502. Issue 9. https://doi.org/10.1103/PhysRevLett.121.090502

[45] Sebastian Krinner, Simon Storz, Philipp Kurpiers, Paul Magnard, Johannes Heinsoo, Raphael Keller, Janis Luetolf, Christopher Eichler, and Andreas Wallraff. 2019. Engineering cryogenic setups for 100-qubit scale superconducting circuit systems. *EPJ Quantum Technology* 6, 1 (2019), 2.

[46] Oğuzcan Kırmemiş, Francisco Manuel Antunes Romão, Emmanouil Giortamis, and Pramod Bhatotia. 2024. *"Weaver: A Retargetable Compiler Framework for FPQA Quantum Architectures" Artifact*. https://doi.org/10.5281/zenodo.14368066

[47] Lingling Lao and Dan E. Browne. 2022. 2QAN: A Quantum Compiler for 2-Local Qubit Hamiltonian Simulation Algorithms. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York, New York) *(ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 351–365. https://doi.org/10.1145/3470496.3527394

[48] Harry Levine, Alexander Keesling, Ahmed Omran, Hannes Bernien, Sylvain Schwartz, Alexander S Zibrov, Manuel Endres, Markus Greiner, Vladan Vuletić, and Mikhail D Lukin. 2018. High-fidelity control and entanglement of Rydberg-atom qubits. *Physical review letters* 121, 12 (2018), 123603.

[49] Harry Levine, Alexander Keesling, Giulia Semeghini, Ahmed Omran, Tout T Wang, Sepehr Ebadi, Hannes Bernien, Markus Greiner, Vladan Vuletić, Hannes Pichler, et al. 2019. Parallel implementation of high-fidelity multiqubit gates with neutral atoms. *Physical review letters* 123, 17 (2019), 170503.

[50] Ang Li, Samuel Stein, Sriram Krishnamoorthy, and James Ang. 2021. QASMBench: A Low-level QASM Benchmark Suite for NISQ Evaluation and Simulation. *arXiv preprint arXiv:2005.13018* (2021).

[51] Gushu Li, Yufei Ding, and Yuan Xie. 2019. Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices. In *Proceedings of the*

Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 1001–1014. https://doi.org/10.1145/3297858.3304023

[52] Gushu Li, Anbang Wu, Yunong Shi, Ali Javadi-Abhari, Yufei Ding, and Yuan Xie. 2022. Paulihedral: A Generalized Block-Wise Compiler Optimization Framework for Quantum Simulation Kernels. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '22)*. Association for Computing Machinery, New York, NY, USA, 554–569. https://doi.org/10.1145/3503222.3507715

[53] Peiyi Li, Ji Liu, Alvin Gonzales, Zain Hamid Saleem, Huiyang Zhou, and Paul Hovland. 2024. QuTracer: Mitigating Quantum Gate and Measurement Errors by Tracing Subsets of Qubits. *arXiv preprint arXiv:2404.19712* (2024).

[54] Yongshang Li, Yu Zhang, Mingyu Chen, Xiangyang Li, and Peng Xu. 2023. Timing-Aware Qubit Mapping and Gate Scheduling Adapted to Neutral Atom Quantum Computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 11 (2023), 3768–3780. https://doi.org/10.1109/TCAD.2023.3261244

[55] Andrew Litteken, Lennart Maximilian Seifert, Jason D. Chadwick, Natalia Nottingham, Tanay Roy, Ziqian Li, David Schuster, Frederic T. Chong, and Jonathan M. Baker. 2023. Dancing the Quantum Waltz: Compiling Three-Qubit Gates on Four Level Architectures. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (Orlando, FL, USA) *(ISCA '23)*. Association for Computing Machinery, New York, NY, USA, Article 71, 14 pages. https://doi.org/10.1145/3579371.3589106

[56] Filip B Maciejewski, Zoltán Zimborás, and Michał Oszmaniec. 2020. Mitigation of readout noise in near-term quantum devices by classical post-processing based on detector tomography. *Quantum* 4 (2020), 257.

[57] Johannes Majer, JM Chow, JM Gambetta, Jens Koch, BR Johnson, JA Schreier, L Frunzio, DI Schuster, Andrew Addison Houck, Andreas Wallraff, et al. 2007. Coupling superconducting qubits via a cavity bus. *Nature* 449, 7161 (2007), 443–447.

[58] Satvik Maurya, Chaithanya Naik Mude, William D. Oliver, Benjamin Lienhard, and Swamit Tannu. 2023. Scaling Qubit Readout with Hardware Efficient Machine Learning Architectures. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (Orlando, FL, USA) *(ISCA '23)*. Association for Computing Machinery, New York, NY, USA, Article 7, 13 pages. https://doi.org/10.1145/3579371.3589042

[59] Alexander J. McCaskey, Dmitry I. Lyakh, Eugene F. Dumitrescu, Sarah S. Powers, and Travis S. Humble. 2019. XACC: A System-Level Software Infrastructure for Heterogeneous Quantum-Classical Computing. arXiv:1911.02452 [quant-ph] https://arxiv.org/abs/1911.02452

[60] Abtin Molavi, Amanda Xu, Martin Diges, Lauren Pick, Swamit Tannu, and Aws Albarghouthi. 2022. Qubit Mapping and Routing via MaxSAT. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1078–1091. https://doi.org/10.1109/MICRO56248.2022.00077

[61] Prakash Murali, Jonathan M. Baker, Ali Javadi-Abhari, Frederic T. Chong, and Margaret Martonosi. 2019. Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 1015–1029. https://doi.org/10.1145/3297858.3304075

[62] Prakash Murali, David C. Mckay, Margaret Martonosi, and Ali Javadi-Abhari. 2020. Software Mitigation of Crosstalk on Noisy Intermediate-Scale Quantum Computers. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 1001–1016. https://doi.org/10.1145/3373376.3378477

[63] Michael A. Nielsen and Isaac L. Chuang. 2011. Quantum Computation and Quantum Information: 10th Anniversary Edition.

[64] Natalia Nottingham, Michael A. Perlin, Dhirpal Shah, Ryan White, Hannes Bernien, Frederic T. Chong, and Jonathan M. Baker. 2024. Circuit decompositions and scheduling for neutral atom devices with limited local addressability. arXiv:2307.14996 [quant-ph] https://arxiv.org/abs/2307.14996

[65] Jeremy L O'brien, Akira Furusawa, and Jelena Vučković. 2009. Photonic quantum technologies. *Nature photonics* 3, 12 (2009), 687–695.

[66] University of British Columbia. [n. d.]. SATLIB – cs.ubc.ca/ hoos/SATLIB/benchm.html. https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html. [Accessed 30-08-2024].

[67] OpenQASM. [n. d.]. OpenQasm 3.0 Grammar. Retrieved June 26, 2024 from https://openqasm.com/grammar/index.html

[68] Tirthak Patel, Daniel Silver, and Devesh Tiwari. 2022. Geyser: A Compilation Framework for Quantum Computing with Neutral Atoms. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York, New York) *(ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 383–395. https://doi.org/10.1145/3470496.3527428

[69] Tirthak Patel and Devesh Tiwari. 2020. DisQ: A Novel Quantum Output State Classification Method on IBM Quantum Computers Using Openpulse. In *Proceedings of the 39th International Conference on Computer-Aided Design* (Virtual Event, USA) *(ICCAD '20)*. Association for Computing Machinery, New York, NY, USA, Article 139, 9 pages. https://doi.org/10.1145/3400302.3415619

[70] Tirthak Patel and Devesh Tiwari. 2020. VERITAS: Accurately Estimating the Correct Output on Noisy Intermediate-Scale Quantum Computers. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–16. https://doi.org/10.1109/SC41405.2020.00019

[71] Tirthak Patel and Devesh Tiwari. 2021. Qraft: Reverse Your Quantum Circuit and Know the Correct Program Output. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 443–455. https://doi.org/10.1145/3445814.3446743

[72] Tirthak Patel, Ed Younis, Costin Iancu, Wibe de Jong, and Devesh Tiwari. 2022. Quest: systematically approximating quantum circuits for higher output fidelity. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 514–528.

[73] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L O'brien. 2014. A variational eigenvalue solver on a photonic quantum processor. *Nature communications* 5, 1 (2014), 4213.

[74] Frank Phillipson. 2024. Quantum Computing in Logistics and Supply Chain Management an Overview. arXiv:2402.17520 [quant-ph] https://arxiv.org/abs/2402.17520

[75] John Preskill. 2018. Quantum Computing in the NISQ era and beyond. *Quantum* 2 (aug 2018), 79. https://doi.org/10.22331/q-2018-08-06-79

[76] Qiskit contributors. 2023. Qiskit: An Open-source Framework for Quantum Computing. https://doi.org/10.5281/zenodo.2573505

[77] qiskit-transpiler [n. d.]. Qiskit Transpiler. https://qiskit.org/documentation/apidoc/transpiler.html. Accessed: 2022-06-09.

[78] Nils Quetschlich, Lukas Burgholzer, and Robert Wille. 2022. MQT Bench: Benchmarking software and design automation tools for quantum computing. *arXiv preprint arXiv:2204.13719* (2022).

[79] Gokul Subramanian Ravi, Pranav Gokhale, Yi Ding, William Kirby, Kaitlin Smith, Jonathan M. Baker, Peter J. Love, Henry Hoffmann, Kenneth R. Brown, and Frederic T. Chong. 2022. CAFQA: A Classical Simulation Bootstrap for Variational Quantum Algorithms. In *Proceedings of the 28th ACM International Conference on Architectural Support for*

Programming Languages and Operating Systems, Volume 1 (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 15–29. https://doi.org/10.1145/3567955.3567958

[80] Gokul Subramanian Ravi, Kaitlin Smith, Jonathan M. Baker, Tejas Kannan, Nathan Earnest, Ali Javadi-Abhari, Henry Hoffmann, and Frederic T. Chong. 2023. Navigating the Dynamic Noise Landscape of Variational Quantum Algorithms with QISMET. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 515–529. https://doi.org/10.1145/3575693.3575739

[81] Mark Saffman, Thad G Walker, and Klaus Mølmer. 2010. Quantum information with Rydberg atoms. *Reviews of modern physics* 82, 3 (2010), 2313.

[82] Ludwig Schmid, David F Locher, Manuel Rispler, Sebastian Blatt, Johannes Zeiher, Markus Müller, and Robert Wille. 2024. Computational capabilities and compiler development for neutral atom quantum processors—connecting tool developers and hardware experts. *Quantum Science and Technology* 9, 3 (2024), 033001.

[83] Ludwig Schmid, David F Locher, Manuel Rispler, Sebastian Blatt, Johannes Zeiher, Markus Müller, and Robert Wille. 2024. Computational capabilities and compiler development for neutral atom quantum processors—connecting tool developers and hardware experts. *Quantum Science and Technology* 9, 3 (apr 2024), 033001. https://doi.org/10.1088/2058-9565/ad33ac

[84] Alireza Seif, Haoran Liao, Vinay Tripathi, Kevin Krsulich, Moein Malekakhlagh, Mirko Amico, Petar Jurcevic, and Ali Javadi-Abhari. 2024. Suppressing Correlated Noise in Quantum Computers via Context-Aware Compiling. arXiv:2403.06852

[85] Ruslan Shaydulin and Yuri Alexeev. 2019. Evaluating Quantum Approximate Optimization Algorithm: A Case Study. In *2019 Tenth International Green and Sustainable Computing Conference (IGSC)*. 1–6. https://doi.org/10.1109/IGSC48788.2019.8957201

[86] Yunong Shi, Nelson Leung, Pranav Gokhale, Zane Rossi, David I. Schuster, Henry Hoffmann, and Frederic T. Chong. 2019. Optimized Compilation of Aggregated Instructions for Realistic Quantum Computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 1031–1044. https://doi.org/10.1145/3297858.3304018

[87] Peter W. Shor. 1999. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Rev.* 41, 2 (1999), 303–332. https://doi.org/10.1137/S0036144598347011 arXiv:https://doi.org/10.1137/S0036144598347011

[88] Irfan Siddiqi. 2021. Engineering high-coherence superconducting qubits. *Nature Reviews Materials* 6, 10 (2021), 875–891.

[89] Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Caroline Collange, and Fernando Magno Quintao Pereira. 2018. Qubit Allocation. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization* (Vienna, Austria) *(CGO 2018)*. Association for Computing Machinery, New York, NY, USA, 113–125. https://doi.org/10.1145/3168822

[90] Seyon Sivarajah, Silas Dilkes, Alexander Cowtan, Will Simmons, Alec Edgington, and Ross Duncan. 2020. t|ket>: a retargetable compiler for NISQ devices. *Quantum Science and Technology* 6, 1 (2020), 014003.

[91] Kaitlin N. Smith, Gokul Subramanian Ravi, Prakash Murali, Jonathan M. Baker, Nathan Earnest, Ali Javadi-Cabhari, and Frederic T. Chong. 2022. TimeStitch: Exploiting Slack to Mitigate Decoherence in Quantum Circuits. *ACM Transactions on Quantum Computing* 4, 1, Article 8 (oct 2022), 27 pages. https://doi.org/10.1145/3548778

[92] Samuel Stein, Nathan Wiebe, Yufei Ding, Peng Bo, Karol Kowalski, Nathan Baker, James Ang, and Ang Li. 2022. EQC: Ensembled Quantum Computing for Variational Quantum Algorithms. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York, New York) *(ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 59–71. https://doi.org/10.1145/3470496.3527434

[93] Bochen Tan, Dolev Bluvstein, Mikhail D. Lukin, and Jason Cong. 2022. Qubit Mapping for Reconfigurable Atom Arrays. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design* (, San Diego, California,) *(ICCAD '22)*. Association for Computing Machinery, New York, NY, USA, Article 107, 9 pages. https://doi.org/10.1145/3508352.3549331

[94] Daniel Bochen Tan, Dolev Bluvstein, Mikhail D. Lukin, and Jason Cong. 2024. Compiling Quantum Circuits for Dynamically Field-Programmable Neutral Atoms Array Processors. *Quantum* 8 (March 2024), 1281. https://doi.org/10.22331/q-2024-03-14-1281 arXiv:2306.03487 [quant-ph].

[95] Swamit S. Tannu and Moinuddin Qureshi. 2019. Ensemble of Diverse Mappings: Improving Reliability of Quantum Computers by Orchestrating Dissimilar Mistakes. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (Columbus, OH, USA) *(MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 253–265. https://doi.org/10.1145/3352460.3358257

[96] Swamit S Tannu and Moinuddin K Qureshi. 2019. Mitigating measurement errors in quantum computers by exploiting state-dependent bias. In *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture*. 279–290.

[97] Swamit S. Tannu and Moinuddin K. Qureshi. 2019. Not All Qubits Are Created Equal: A Case for Variability-Aware Policies for NISQ-Era Quantum Computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 987–999. https://doi.org/10.1145/3297858.3304007

[98] Teague Tomesh, Pranav Gokhale, Victory Omole, Gokul Subramanian Ravi, Kaitlin N Smith, Joshua Viszlai, Xin-Chuan Wu, Nikos Hardavellas, Margaret R Martonosi, and Frederic T Chong. 2022. Supermarq: A scalable quantum benchmark suite. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 587–603.

[99] Vinay Tripathi, Huo Chen, Mostafa Khezri, Ka-Wa Yip, E.M. Levenson-Falk, and Daniel A. Lidar. 2022. Suppression of Crosstalk in Superconducting Qubits Using Dynamical Decoupling. *Phys. Rev. Appl.* 18 (Aug 2022), 024068. Issue 2. https://doi.org/10.1103/PhysRevApplied.18.024068

[100] Cenk Tüysüz, Giuseppe Clemente, Arianna Crippa, Tobias Hartung, Stefan Kühn, and Karl Jansen. 2023. Classical splitting of parametrized quantum circuits. *Quantum Machine Intelligence* 5, 2 (2023), 34.

[101] Sergio O Valenzuela, William D Oliver, David M Berns, Karl K Berggren, Leonid S Levitov, and Terry P Orlando. 2006. Microwave-induced cooling of a superconducting qubit. *Science* 314, 5805 (2006), 1589–1592.

[102] Hanrui Wang, Pengyu Liu, Daniel Bochen Tan, Yilian Liu, Jiaqi Gu, David Z. Pan, Jason Cong, Umut A. Acar, and Song Han. 2024. Atomique: A Quantum Compiler for Reconfigurable Neutral Atom Arrays. https://doi.org/10.48550/arXiv.2311.15123 arXiv:2311.15123 [quant-ph].

[103] Hanrui Wang, Pengyu Liu, Daniel Bochen Tan, Yilian Liu, Jiaqi Gu, David Z. Pan, Jason Cong, Umut A. Acar, and Song Han. 2024. Atomique: A Quantum Compiler for Reconfigurable Neutral Atom Arrays. arXiv:2311.15123

[104] Hanrui Wang, Daniel Bochen Tan, Pengyu Liu, Yilian Liu, Jiaqi Gu, Jason Cong, and Song Han. 2024. Q-Pilot: Field Programmable Qubit Array Compilation with Flying Ancillas. In *Proceedings of the 61st ACM/IEEE Design Automation Conference* (San Francisco, CA, USA) *(DAC '24)*. Association for Computing Machinery, New York, NY, USA, Article 306, 6 pages. https://doi.org/10.1145/3649329.3658470

[105] Meng Wang, Bo Fang, Ang Li, and Prashant J. Nair. 2024. Red-QAOA: Efficient Variational Optimization through Circuit Reduction. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (, La Jolla, CA, USA,) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 980–998. https://doi.org/10.1145/3620665.3640363

[106] Robert Wille, Lukas Burgholzer, and Alwin Zulehner. 2019. Mapping Quantum Circuits to IBM QX Architectures Using the Minimal Number of SWAP and H Operations. In *Proceedings of the 56th Annual Design Automation Conference 2019* (Las Vegas, NV, USA) *(DAC '19)*. Association for Computing Machinery, New York, NY, USA, Article 142, 6 pages. https://doi.org/10.1145/3316781.3317859

[107] Amanda Xu, Abtin Molavi, Lauren Pick, Swamit Tannu, and Aws Albarghouthi. 2023. Synthesizing Quantum-Circuit Optimizers. *Proceedings of the ACM on Programming Languages* 7, PLDI (jun 2023), 835–859. https://doi.org/10.1145/3591254

[108] Chi Zhang, Ari B. Hayes, Longfei Qiu, Yuwei Jin, Yanhao Chen, and Eddy Z. Zhang. 2021. Time-Optimal Qubit Mapping. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 360–374. https://doi.org/10.1145/3445814.3446706

[109] Alwin Zulehner, Alexandru Paler, and Robert Wille. 2019. An Efficient Methodology for Mapping Quantum Circuits to the IBM QX Architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38, 7 (July 2019), 1226–1236. https://doi.org/10.1109/TCAD.2018.2846658

Received 2024-09-09; accepted 2024-11-04