Check for updates

Research
A SCIENCE PARTNER JOURNAL

RESEARCH ARTICLE

A Resource-Virtualized and Hardware-Aware Quantum Compilation Framework for Real Quantum Computing Processors

Hong-Ze Xu[1,2,3*], Xu-Dan Chai[1,3], Meng-Jun Hu[1,3*], Zheng-An Wang[1,3], Yu-Long Feng[1,3], Yu Chen[1,4], Xinpeng Zhang[1,5], Jingbo Wang[1,3], Wei-Feng Zhuang[1,3], Yu-Xin Jin[1,3], Yirong Jin[1,3], Haifeng Yu[1,3], Heng Fan[1,6,7,8,9], and Dong E. Liu[1,2,3,9,10*]

[1]Beijing Academy of Quantum Information Sciences, Beijing 100193, China. [2]State Key Laboratory of Low Dimensional Quantum Physics, Department of Physics, Tsinghua University, Beijing 100084, China. [3]Beijing Key Laboratory of Fault-Tolerant Quantum Computing, Beijing 100193, China. [4]Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China. [5]School of Medical Technology, Beijing Institute of Technology, Beijing 100081, China. [6]Institute of Physics, Chinese Academy of Sciences, Beijing 100190, China. [7]School of Physical Sciences, University of Chinese Academy of Sciences, Beijing 100049, China. [8]CAS Center for Excellence in Topological Quantum Computation, UCAS, Beijing 100190, China. [9]Hefei National Laboratory, Hefei 230088, China. [10]Frontier Science Center for Quantum Information, Beijing 100184, China.

*Address correspondence to: xuhz@baqis.ac.cn (H.-Z.X.); humj@baqis.ac.cn (M.-J.H.); dongeliu@mail.tsinghua.edu.cn (D.E.L.)

As quantum computing systems continue to scale up and become more clustered, efficiently compiling user quantum programs into high-fidelity executable sequences on real hardware remains a key challenge for current quantum compilation systems. In this study, we introduce a system-software framework that integrates resource virtualization and hardware-aware compilation for real quantum computing processors, termed QSteed. QSteed virtualizes quantum processors through a 4-layer abstraction hierarchy comprising the real quantum processing unit (QPU), standard QPU (StdQPU), substructure of the QPU (SubQPU), and virtual QPU (VQPU). These abstractions, together with calibration data, device topology, and noise descriptors, are maintained in a dedicated database to enable unified and fine-grained management across superconducting quantum platforms. At run time, the modular compiler queries the database to match each incoming circuit with the most suitable VQPU, after which it confines layout, routing, gate resynthesis, and noise-adaptive optimizations to that virtual subregion. The complete stack has been deployed on the Quafu superconducting cluster, where experimental runs confirm the correctness of the virtualization model and the efficacy of the compiler without requiring modifications to user code. By integrating resource virtualization with a select-then-compile workflow, QSteed demonstrates a robust architecture for compiling programs on noisy superconducting processors. This architectural approach offers a promising path toward efficient compilation needs across various superconducting quantum computing platforms in the noisy intermediate-scale quantum era.

Citation: Xu HZ, Chai XD, Hu MJ, Wang ZA, Feng YL, Chen Y, Zhang X, Wang J, Zhuang WF, Jin YX, et al. A Resource-Virtualized and Hardware-Aware Quantum Compilation Framework for Real Quantum Computing Processors. *Research* 2025;8:Article 0947. https://doi.org/10.34133/research.0947

Submitted 27 July 2025
Revised 10 September 2025
Accepted 26 September 2025
Published 16 October 2025

Copyright © 2025 Hong-Ze Xu et al. Exclusive licensee Science and Technology Review Publishing House. No claim to original U.S. Government Works. Distributed under a Creative Commons Attribution License (CC BY 4.0).

Downloaded from https://spj.science.org on November 03, 2025

Introduction

In recent years, various quantum computing platforms, including superconducting qubit [1–4], ion trap [5–7], neutral atoms [8–10], and photonic quantum devices [11,12], have achieved remarkable progress, with increasing qubit control fidelities and scalable architectures reaching dozens or even hundreds of qubits. Despite these advances, quantum computing devices remain scarce resources. The advent of quantum cloud platforms has alleviated this limitation by providing remote access to both public and private quantum hardware via cloud services. This paradigm shift substantially lowers the threshold for quantum experimentation and accelerates progress in domains such as quantum chemistry [13,14], machine learning [15–17], combinatorial optimization [18], and fundamental quantum physics [19,20]. However, deploying high-level quantum algorithms on real hardware remains challenging due to the mismatch between algorithmic abstractions and hardware operations. Bridging this gap depends on efficient quantum compilation, which serves as a critical layer for

transforming quantum programs into hardware-executable instructions.

In the field of quantum compilation, several commercial-grade software frameworks have been developed, including IBM's Qiskit [21], Google's Cirq [22], Rigetti's PyQuil [23], Quantinuum's Pytket [24], and Origin Quantum's QPanda [25]. These frameworks provide comprehensive toolchains that are tightly integrated with their respective hardware platforms. Concurrently, academic research has produced a number of specialized toolkits, such as Quartz [26] for circuit super-optimization and BQSKit [27] that incorporates advanced synthesis algorithms. Additionally, there are compilers tailored to specific quantum algorithms, such as Qcover [28] for the quantum approximate optimization algorithm (QAOA) [29]. Despite these advances, most existing frameworks still face challenges in handling hardware noise and adapting to multi-backend cloud platforms. They generally follow a common paradigm: mapping logical circuits directly onto the entire physical device. However, in the noisy intermediate-scale quantum (NISQ) era, hardware noise on quantum processors (such as 2-qubit gate error rates) is nonuniformly distributed. Simply compiling and optimizing circuits by mapping them to the full chip neglects the intrinsic differences of hardware resources. In this context, developing compilation strategies that proactively avoid high-noise hardware resources emerges as an effective pathway to improving quantum program performance. Furthermore, deploying compilation software on multi-backend heterogeneous quantum cloud platforms and achieving unified resource management and hardware-aware compilation still requires further architectural innovation.

In this work, rather than concentrating on specific compilation optimization algorithms, we focus on an architectural-level innovation in compilation design. We introduce QSteed, a novel quantum compilation system designed for deployment on superconducting quantum computing platforms. QSteed embodies a distinct architectural paradigm built on resource virtualization and a select-then-compile workflow. In particular, it tightly integrates a modular compiler with a quantum resource virtualization layer, featuring several key characteristics: (a) The resource virtualization manager uses heuristic strategies to identify high-quality subregions of a quantum chip, which are then abstracted into a queryable database of virtual quantum processing units (VQPUs). This enables efficient and unified management of multiple quantum backends. (b) The quantum compiler first selects an optimal VQPU for a given input circuit based on structural similarity or fidelity metrics. It then performs efficient, hardware-aware transpilation on this smaller subregion to generate a high-fidelity executable circuit. (c) The system is packaged into 2 lightweight application programming interfaces (APIs), one for adding or updating quantum processors in the database and another for user task compilation, simplifying integration with existing superconducting quantum cloud toolchains.

We have deployed QSteed on a quantum cloud computing cluster based on superconducting qubit processors. Specifically, it has been integrated into the Quafu quantum cloud platform, which validates the effectiveness of its overall architecture. To assess QSteed's compilation performance, we executed benchmark circuits with up to 30 qubits on Quafu's Baihua processor and simulated larger circuits using calibration parameters from the Baihua chips. To further examine the robustness of the QSteed architecture, we performed the same simulations with parameter data from Google's Willow processor. Experiments on real hardware and classical simulations show that, for the evaluated small- to medium-scale benchmark circuits, QSteed consistently outperforms leading toolchains such as Qiskit and Pytket in compilation speed while achieving comparable or higher circuit fidelity. Overall, QSteed provides efficient support for practical quantum cloud services. Conceptually, the architecture has the potential to be extended to other quantum computing platforms, but realizing such portability would require redesigning the VQPU database generation and qubit mapping strategies for each specific hardware architecture. At present, all implementations and experiments are based on superconducting qubit platforms.

Results

Overall system architecture design

As schematically depicted in Fig. 1, the QSteed system is designed with a layered architecture. Its core consists of 2 key functional modules: a quantum compiler and a quantum computing resource manager. Through a unified API interface, the system enables efficient and seamless hardware deployment. The operational workflow of the system is as follows:

On the hardware management side, a backend operator registers a quantum processor via the Manager API. This process imports essential metadata, such as device topology, real-time calibration data, and native gate sets, into the resource manager. The manager then proactively virtualizes the physical hardware by constructing a 4-layer hierarchical model: QPU, standard QPU (StdQPU), substructure of the QPU (SubQPU), and VQPU. These models are stored in a relational database, establishing a precomputed and queryable resource pool for the compiler, which enables unified management across multiple superconducting backends.

On the quantum task processing side, a user task (containing information such as the quantum circuit and whether a backend is designated) is forwarded to the compiler through the Compiler API. The compiler adopts a select-then-compile workflow: First, the input circuit is standardized by rewriting it into a hardware-agnostic unified representation. Next, the compiler queries the resource database to select an optimal VQPU that best matches the circuit's structural or fidelity requirements. Once the target VQPU has been determined, the compiler executes a hardware-aware transpilation process tailored to this optimal subregion, generates quantum assembly language (QASM) [30,31] code ready for the target hardware, and returns a detailed compilation report (e.g., compilation time, circuit depth, and gate counts). Finally, it performs hardware constraint verification on the output to ensure that the generated QASM code can run safely and directly on the selected quantum processor.

By deeply integrating the compilation process with resource management, QSteed provides unified orchestration of quantum resources and enables efficient task compilation and execution. Its virtualization and select-then-compile mechanism decouples hardware characteristics from user tasks, allowing quantum circuits to be dynamically mapped onto the most suitable physical qubits. Related use cases and implementation details are provided in Section S1.

It should be noted that throughout the system design, our primary focus is on 2-qubit gate noise. This emphasis is reflected in several core components, including the fidelity-first strategy in resource virtualization, the fidelity-first policy in VQPU

Downloaded from https://spj.science.org on November 03, 2025

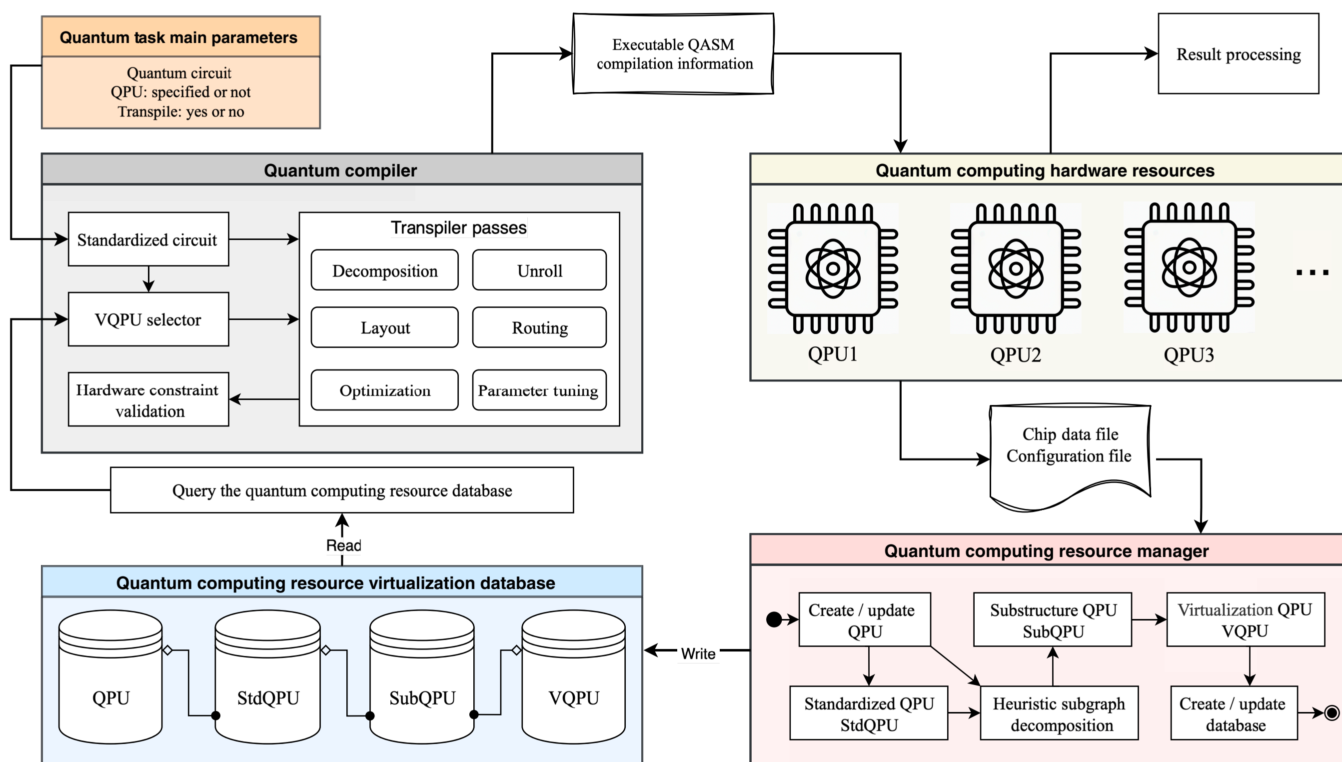

Fig. 1. QSteed system architecture. It consists primarily of 2 components: the quantum compiler and the quantum computing resource manager. The manager models the quantum chip at various abstraction levels, including QPU, StdQPU, SubQPU, and VQPU. These representations are stored in a quantum computing resource virtualization database, enabling unified management of quantum backend devices. The compiler queries the virtualization database to compile the user task onto the optimal physical qubits, returning the optimized executable QASM circuit along with relevant compilation information.

selection, and the noise-aware routing algorithm. While decoherence noise is a consideration, it is only incorporated within the routing algorithm through the introduction of gate parallelism considerations.

Design of the quantum resource virtualization manager

Hierarchical abstraction of quantum processors

Our approach to quantum hardware management is conceptually inspired by classical virtualization. In classical computing, virtualization technologies such as virtual machines build an abstraction layer on top of reliable and homogeneous physical resources, thereby enabling efficient resource sharing and isolation. Similarly, our framework virtualizes quantum processors to mask the underlying physical complexity, manage resources more effectively, and provide a unified backend interface for upper-layer compilation services.

However, the analogy ends at this high-level concept, as the motivations and challenges of quantum virtualization are fundamentally different. Classical virtualization is designed to share stable and homogeneous resources, whereas in the NISQ era quantum virtualization must contend with noisy and heterogeneous resources. The central challenge is not simply partitioning a high-quality whole, but identifying and exploiting relatively high-quality subregions within a chip that is inherently imperfect and subject to time-varying noise. Unlike classical virtualization, which seeks to emulate a complete and stable runtime environment, our quantum virtualization constructs a lightweight hardware abstraction layer that is noise-aware and topology-constrained, specifically tailored to support resource management and compilation optimization. This distinction underpins the design of our framework.

As quantum chips scale up and their architectures become increasingly complex, managing quantum resources across multi-backend clusters is becoming progressively more challenging. To address this, we have developed a virtualization manager that abstracts quantum chips into a database representation, enabling efficient orchestration of large-scale quantum clusters. Within this framework, quantum chips are represented through 4 hierarchical abstraction levels: QPU, StdQPU, SubQPU, and VQPU.

QPU

The QPU serves as the foundational abstraction layer, providing a direct digital representation of a physical quantum processor's characteristics. This "digital twin" encapsulates critical hardware parameters, including the chip's architecture type, qubit connectivity topology, real-time calibration data (e.g., gate fidelities and coherence times), the supported native gate set, gate durations, a unique hardware identifier, and its current operational status. This layer grounds the entire virtualization stack in the precise state of the physical device.

StdQPU

Due to current limitations in micro-/nanofabrication processes, superconducting quantum chips inevitably have defects [32], rendering some qubits unusable and resulting in irregular chip structures. We therefore introduce StdQPU, an idealized

Downloaded from https://spj.science.org on November 03, 2025

StdQPU architecture designed to embed these defective chips. For instance, on superconducting platforms, StdQPU can be modeled as a 2-dimensional grid structure, into which most superconducting chips can be mapped. This abstraction facilitates the chip to be dynamically partitioned into multiple regions capable of hosting concurrent workloads, thereby laying the groundwork for QSteed's future support of multi-program quantum execution [33]. Although the present study does not explore such multi-program scenarios in depth, we retain this layer of abstraction to preserve the completeness of the virtualization stack.

SubQPU

SubQPU represents high-quality substructures identified within a QPU. For a topology graph $G(V,E)$ with N qubits and M coupling edges, enumerating all its substructures has an exponential complexity of $O\left(2^{N+M}\right)$. Consequently, we employ heuristic algorithms to identify valuable SubQPUs. To maximize the diversity of these substructures to suit different types of quantum circuits, we designed 3 complementary heuristic strategies.

Fidelity-first strategy: This strategy aims to identify n-qubit coupled substructures with high average fidelity, which is crucial for noise-sensitive quantum applications. Specifically, for $n = 1$ and $n = 2$, we construct the optimal substructures by sorting nodes or edges based on their respective single- or 2-qubit gate fidelities. For $n \geq 3$, we heuristically construct an n-qubit substructure, subG, by initializing it with the highest-fidelity edge and iteratively adding adjacent qubits sorted by 2-qubit gate fidelity until the substructure reaches size n.

Degree-first strategy: This strategy prioritizes node degree to identify substructures composed of highly connected qubits, which is beneficial for algorithms requiring a high degree of entanglement. Its implementation is similar to the fidelity-first strategy, with the key difference being that during the construction of the substructure subG, adjacent qubits sorted by node degree are iteratively added.

Random selection strategy: By introducing randomness, this strategy explores substructures with different topologies to enrich the database. In each step of the iteration, a neighboring qubit is randomly added to subG.

VQPU

To unify the description of quantum resources, each SubQPU is further abstracted into a VQPU, the final compiler-facing representation. The VQPU omits certain physical details of the quantum chip, retaining only the essential information needed for compilation. This includes the mapping from virtual qubits (numbered starting from 0) to physical qubits, the noisy topological structure, the native gate set, and the identifier of the parent QPU. This abstraction supplies the compiler with a clean, standardized interface. Figure 2B shows 6-qubit VQPU configuration extracted from the Baihua chip.

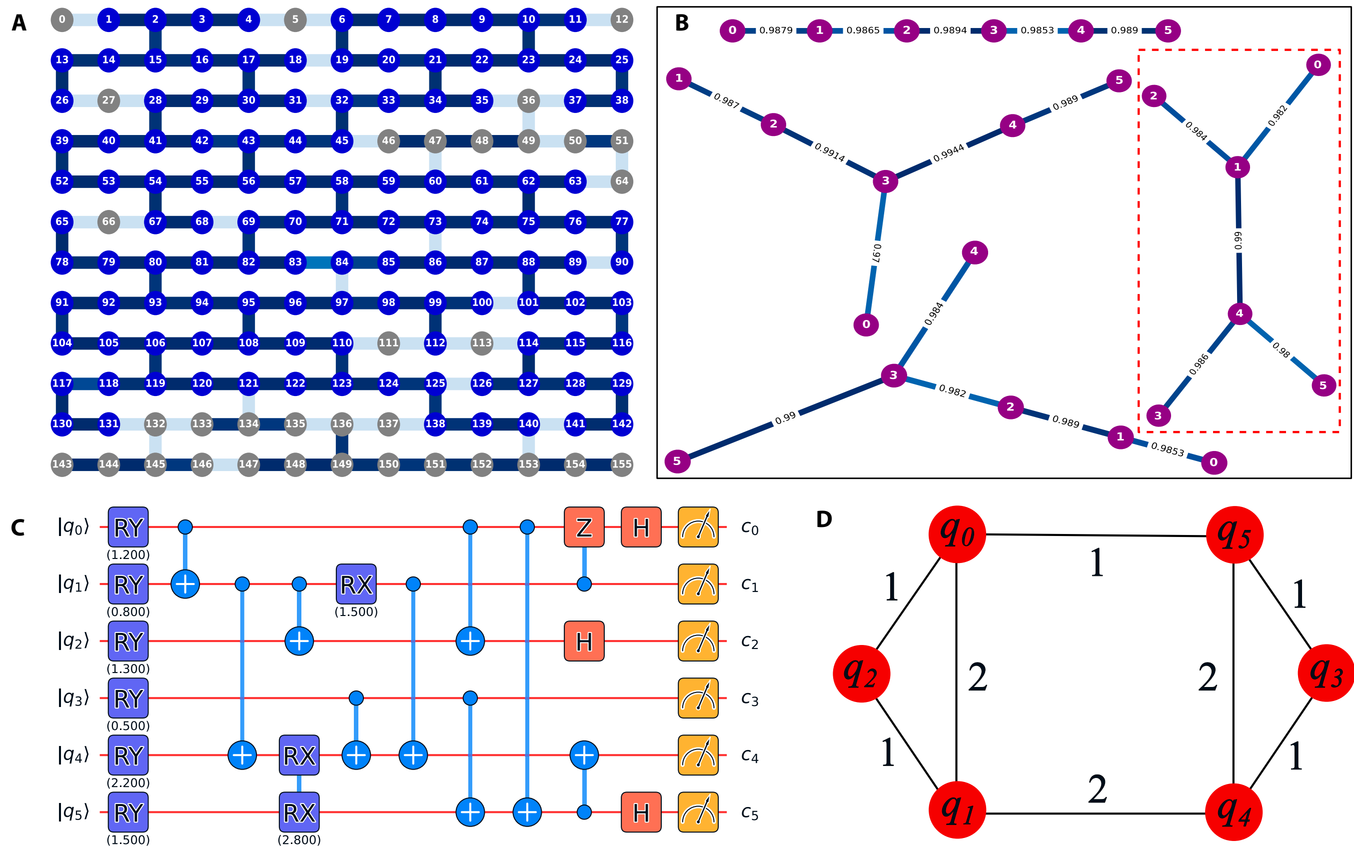

Fig. 2. (A) Schematic of the Baihua quantum chip, with blue regions representing 122 connected qubits. (B) The 6-qubit VQPU structure within the Baihua quantum chip, where the red box highlights the target VQPU (the mapping from virtual qubits v to physical qubits q is $v_{\{0,1,2,3,4,5\}} \to q_{\{61,62,63,74,75,76\}}$) with the structure most similar to the circuit in (C). (C) Quantum circuit. (D) Weighted graph representation of the quantum circuit, where the nodes represent qubits, the edges represent 2-qubit gates, and the edge weights represent the number of 2-qubit gates.

Downloaded from https://spj.science.org on November 03, 2025

Algorithm 1 Find Substructures in QPU Coupling Graph

Input: Qubit coupling graph of QPU $G(V, E)$
Output: List of all substructures $all_substructure$

```
1: N ← len(G.nodes())
2: Sn ← sorted(G.nodes(), fidelity)          ▷ Sort nodes by fidelity
3: Se ← sorted(G.edges(), fidelity)          ▷ Sort edges by fidelity
4: all_substructure = [ ]
5: all_substructure.append((1, Sn))
6: all_substructure.append((2, Se))
7: for n ← 3 to N do
8:     n_substructure = [ ]
9:     for each method in {fidelity, degree, random} do
10:        for each edge in Se do
11:            subG ← Graph()
12:            subG.add_edge(edge)
13:            qubits ← 2
14:            while qubits ≠ n do
15:                neighbors ← sorted(
                            subG.neighbors(), method)
16:                best_neighbor ← neighbors[0]
17:                subG.add_node(best_neighbor)
18:                qubits ← qubits + 1
19:            n_substructure.append(subG)
20:    n_substructure ← sorted(n_substructure, ave_fidelity)
21:    all_substructure.append((n, n_substructure))
```

Complexity analysis of virtualization strategy

The time complexity of quantum hardware virtualization is primarily composed of 2 components. The first corresponds to identifying the optimal substructures of the chip, as described in Algorithm 1. The second pertains to the deletion and insertion costs incurred during database updates.

For a quantum processor with N qubits and M coupling edges, the time complexity of Algorithm 1 is dominated by the main loop structure. After an initial sorting of nodes and edges, which takes $O(NlogN + MlogM)$, the algorithm iterates through each of the M edges to construct substructures of size n (from 3 to N). The core of this process is an iterative growth step, where new qubits are added to the substructure. By employing a priority queue to efficiently select the best neighboring qubit at each step, the inner loop has a complexity of approximately $O(nlogN)$. Consequently, the total time complexity of the algorithm is approximately $O(MN^2logN)$. For sparsely coupled superconducting chips, we have $M = O(N)$, and the complexity simplifies to $O(N^3logN)$. A more detailed derivation can be found in Section S2.1. In addition, we performed numerical simulations on representative superconducting chip topologies, including the square-lattice topology adopted by Google and USTC's Zuchongzhi processor, the heavy-hexagonal topology promoted by IBM, as well as the hexagonal topology. The numerical results (see Figs. S2 to S5) show that for chips with up to 200 qubits, our algorithm can complete the identification of optimal subregions within a few minutes, and the fitted scaling is consistent with $O(N^3logN)$. This demonstrates that the approach is fully practical for current NISQ devices.

The update time of the database depends on the specific implementation and architecture adopted. In our current implementation, a MySQL database is used, where record deletion and insertion are carried out in a serial manner. Numerical results (see Figs. S3 to S5) indicate that the associated complexity scales as $O(N^4)$. Possible directions for future improvement may include adopting primary-key-based differential deletion and batch insertion strategies, as well as redesigning the database architecture or considering migration to more efficient systems such as MongoDB or Redis.

Design of a modular hardware-aware quantum compiler

Standardized circuit

Before executing the core matching and compilation workflow, QSteed first normalizes user-submitted quantum circuits via a standardized preprocessing module. By correcting common formatting issues, such as invalid measurements, missing classical registers, and redundant qubits, this module effectively ensures that subsequent processes execute reliably. Its primary functions include the following.

Measurement and classical-register correction: Quantum programs require classical registers to store measurement results. This module automatically analyzes the QASM code, appends any omitted measurement instructions, and provisions the requisite classical registers, thereby averting runtime errors attributable to user oversight.

Redundant qubit cleaning: By analyzing the QASM code, this module identifies and removes qubits that do not participate in any effective quantum operations within the circuit. This step reduces unnecessary computational overhead in later compilation phases, conserves scarce qubit resources on NISQ devices, mitigates potential qubit interference, and ultimately helps improve circuit fidelity.

VQPU selector

Benefiting from the construction of the VQPU database, we no longer need to consider the low-level particulars of the entire quantum chips. Instead, we can focus on matching a suitable VQPU for each quantum circuit. The VQPU selection module begins by filtering all candidate VQPUs from the database that meet the circuit's qubit count requirement. Subsequently, one of the following strategies is employed to determine the final VQPU.

Fidelity-first strategy: This strategy selects the VQPU with the highest overall fidelity, defined as the product of all 2-qubit gate fidelities within the VQPU. Since the VQPU database is presorted by this metric during its construction, this strategy ensures rapid identification of the optimal VQPU for the task.

Structure-first strategy: This strategy aims to reduce the overhead of SWAP gates generated during the subsequent qubit mapping process by identifying the VQPU that most closely resembles the circuit topology. This is a subgraph isomorphism problem within the field of quantum compilation [34–36]. In QSteed, we employ the following practical implementation: We represent both the quantum circuit and the VQPU as weighted graphs. In the circuit graph, vertices correspond to logical qubits, and edges correspond to 2-qubit gates (multi-qubit gates are expanded into a fully connected subgraph), where edge weights signify the number of 2-qubit gates. For example, the quantum circuit in Fig. 2C can be represented as the weighted graph shown in Fig. 2D. Similarly, in the VQPU graph, vertices denote physical qubits, edges represent direct coupling pairs, and edge weights indicate the fidelity of 2-qubit gates. For structural comparison, we first normalize the edge weights in both graphs to the range [0,1]. Then, we query whether there exists

Downloaded from https://spj.science.org on November 03, 2025

a graph isomorphic to the circuit diagram among the candidate VQPUs (ignoring edge weights). If such an isomorphic match is found, that VQPU is selected, enabling perfect mapping of the quantum circuit onto the hardware without requiring additional SWAP gates. When no isomorphic match exists, we quantify graph similarity with the Weisfeiler–Lehman subtree kernel [37] and select the VQPU that yields the highest similarity score. In cases where multiple VQPUs have the same score, we break ties by choosing the VQPU with the highest overall fidelity. Detailed algorithmic descriptions appear in Section S2.

Quantum circuit transpiler

Once the target VQPU has been identified, the quantum circuit transpiler is responsible for compiling the logical circuit into a physical circuit that can be executed on that VQPU. The key challenge of this process is to satisfy the hardware's topological constraints while performing gate-level optimizations to enhance the fidelity of the final circuit. To facilitate flexible and efficient compilation, the transpiler of QSteed adopts a modular pipeline architecture and employs directed acyclic graph (DAG) as the intermediate representation of circuits, which aligns with the design philosophy of mainstream frameworks such as Qiskit [21] and Pytket [24]. This design offers a high degree of flexibility and extensibility, enabling researchers to conveniently test new compilation algorithms.

Our transpiler is specifically composed of several key components: Transpiler, Pass, PassFlow, Model, and Backend, as depicted in Fig. 3. The Transpiler serves as the main entry point, responsible for selecting a particular PassFlow to execute the transpilation process according to a preset optimization level. Users also have the option to customize their own PassFlow. A PassFlow is an ordered sequence of various Pass instances, each responsible for a specific task such as gate decomposition, qubit mapping, circuit optimization, and variational circuit parameter tuning. Users can create a custom Pass by inheriting from the BasePass class to implement specific functionalities. Throughout transpilation, a shared Model object maintains all state information, including the current qubit layout and the Backend description derived from the selected VQPU. Each Pass obtains this state via get_model, performs its transformation, and writes back the updated state using set_model. This disciplined data flow keeps the Model synchronized across the pipeline, ensuring that every subsequent pass operates on the latest information.

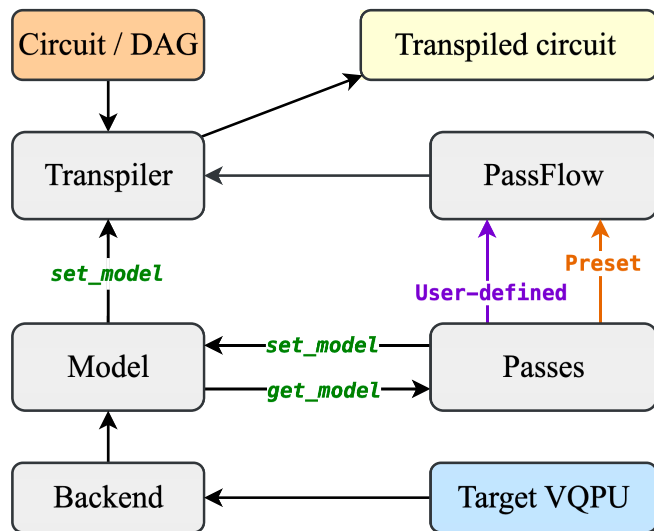

Fig. 3. The quantum circuit transpilation and optimization workflow adopts a modular design, consisting of core components: Transpiler, Pass, PassFlow, Model, and Backend. The target VQPU is first modeled as a Backend object and then abstracted into a Model object. During the transpilation process, the Model object stores and passes the parameter data required for transpilation. The transpiler selects a predefined or user-defined PassFlow based on the optimization level to execute the transpilation process. A PassFlow is composed of a sequence of functional components, each responsible for a specific transpilation task. The transpilation process is primarily performed on the directed acyclic graph (DAG) representation of the quantum circuit. Upon completion, the transpiled quantum circuit is output.

Among all transpilation steps, qubit mapping is often the most critical, directly determining the final circuit's depth and fidelity. This process dynamically assigns logical qubits to physical qubits and is typically divided into 2 key stages, layout and routing. Layout establishes the initial mapping of logical qubits to physical qubits, while routing ensures that all 2-qubit gate operations adhere to the hardware's connectivity constraints by inserting SWAP gates. Accordingly, the remainder of this subsection focuses on the qubit mapping process; the other transpilation steps follow standardized methodologies detailed in Section S3.

Numerous qubit mapping algorithms have been developed [38,39], among which the SABRE algorithm [40] is widely adopted in mainstream compilation frameworks like Qiskit due to its recognized efficiency and scalability. However, this algorithm still has certain limitations, such as its trivial initial layout and its insensitivity to hardware noise. Currently, various optimization algorithms specifically targeting initial layout [35,41,42], as well as routing algorithms that consider hardware noise [43–45], have been proposed. The qubit mapping algorithm implemented in QSteed introduces the following improvements over the original SABRE algorithm.

Structure-aware initial layout: Let the quantum circuit's weighted graph be G_{QC} and the graph of the optimal VQPU determined by the VQPU selector be G_{VQPU}. We apply 2 layout initialization methods: (a) Degree initialization: Nodes in G_{QC} and G_{VQPU} are sorted by degree, and logical qubits q are mapped one-to-one onto virtual qubits v based on the sorted order. For instance, if the sorted orders are $[q_1, q_3, q_0, q_2]$ for G_{QC} and $[v_2, v_0, v_3, v_1]$ for G_{VQPU}, the initial layout is $q_{\{1,3,0,2\}} \leftrightarrow v_{\{2,0,3,1\}}$. (b) Weight initialization: If node degrees are identical, sorting is performed based on edge weights, and nodes are then mapped one-to-one.

Noise-aware routing: The original SABRE algorithm employs the nearest neighbor cost function, defined by the distance matrix $D[i][j]$, which represents the shortest path between physical qubits Q_i and Q_j. However, for NISQ hardware with nonuniform 2-qubit gate fidelities, this approach may fail to maximize the overall circuit fidelity. To improve this, we replace the distance matrix $D[i][j]$ with a fidelity matrix $\mathrm{Fi}[i][j]$, where each entry identifies the path with the highest cumulative fidelity, calculated as the product of all 2-qubit gate fidelities along the path. We then define an improved, noise-aware heuristic cost function H_{Fi},

$$H_{\mathrm{Fi}} = \max\left(\mathrm{decay}(SWAP.q_1), \mathrm{decay}(SWAP.q_2)\right) * \left\{ \frac{1}{|F|} \sum_{gate \in F} \mathrm{Fi}[\pi(gate.q_1)][\pi(gate.q_2)] + W^* \frac{1}{|E|} \sum_{gate \in E} \mathrm{Fi}[\pi(gate.q_1)][\pi(gate.q_2)] \right\}, \tag{1}$$

Downloaded from https://spj.science.org on November 03, 2025

where F represents the front layer of gates in the quantum circuit, E denotes the extended layer of gates, and $\pi()$ represents the mapping from logical qubits $q_{\{1,2,\ldots,n\}}$ to physical qubits $Q_{\{1,2,\ldots,N\}}$. The parameter $W \in (0,1)$ is a weight factor, and $\text{decay}(q_i) = 1 + \delta$ characterizes the parallelism of the compiled circuit. Detailed explanations of these parameters can be found in [40].

However, solely pursuing the highest-fidelity path may lead to an increased number of SWAP operations, which in turn deepens the circuit and exacerbates decoherence. Therefore, we further design a hybrid heuristic H_M that integrates both distance-based and fidelity-based cost functions. Specifically, at each step of the routing process, given the set of candidate SWAPs S, we first compute the distance-based cost H_D (replacing $\text{Fi}[i][j]$ in Eq. (1) with the distance matrix $\text{D}[i][j]$) for each SWAP and retain only those with the minimum value. If multiple candidates remain, we evaluate the fidelity-based cost H_{Fi} within this subset and select the SWAP with the highest score. If a tie still exists, one SWAP is selected at random from the remaining candidates as the final result, denoted by s_{opt}. This method is formally defined as:

$$s_{\text{opt}} := \text{Random}\left(\underset{s \in \arg\min_{s \in S} H_D(s)}{\arg\max} \; H_{\text{Fi}}(s) \right). \tag{2}$$

Hardware constraint validation

Before dispatching the compiled QASM code to the target backend, QSteed performs a series of final hardware compatibility checks. This step does not constitute formal program verification in the strict academic sense [46–48]; rather, it is a pragmatic pre-execution safeguard commonly embedded in production-grade compilation frameworks. Its purpose is to intercept tasks that are destined to fail because they violate the physical limitations of the hardware, thereby averting futile consumption of real quantum resources. Specifically, QSteed's hardware constraint validation module includes the following.

Two-qubit coupling constraints: Verifies that every 2-qubit gate in the circuit respects the qubit coupling graph of the target processor, thereby ensuring topological compatibility with the designated device.

Gate count and circuit depth limits: Checks that the total number of gates and the circuit depth remain below the backend's execution thresholds, preventing invalid tasks with excessively low fidelity.

Non-empty circuit check: Confirms that the circuit contains at least one quantum operation, averting the submission of empty jobs that would waste quantum resources and trigger runtime errors.

Experiments and simulations

We first conducted real-device experiments using the Quafu superconducting quantum computing cloud platform, selecting the Baihua quantum processor (as shown in Fig. 2A) as the target backend. Through the cloud platform, we accessed the most recent calibration data of the chip, including qubit coupling topology as well as single- and 2-qubit gate fidelities. The median values of key performance parameters for the Baihua chip are a single-qubit error rate of 7.6×10^{-4}, a 2-qubit error rate of 2×10^{-2}, a relaxation time T_1 of 71.608 μs, and a dephasing time T_2 of 23.724 μs (these values may vary after recalibration of the chip). To ensure the reliability of the calibration data provided by the platform, we employed the ErrorGnoMark [49] software for benchmarking, which is designed to provide a comprehensive diagnostic of quantum chips. Specifically, we assessed single-qubit gate and CNOT gate infidelities using cross-entropy benchmarking (XEB) [1] and channel spectrum benchmarking (CSB) [50], thereby establishing a reliable hardware noise baseline for the subsequent experiments. Although CSB can characterize process infidelity, statistical infidelity, and rotation angle errors, our compilation study concerns only gate fidelity, so we report process infidelity exclusively. Gate benchmarking results are summarized in Fig. 4. Specifically, Fig. 4A illustrates single-qubit process infidelities obtained through CSB, which approximate average gate infidelities. CNOT process infidelities, also derived from CSB, are reported in Fig. 4B. Figure 4C details average CNOT error rates from XEB, reflecting the difference between measured and ideal output distributions. A theoretical expectation of consistency exists for the data presented in Fig. 4B and C. The observed discrepancies may be attributable to noise instability.

To ensure fairness in performance comparisons among different compilers under realistic noise conditions, all frameworks were evaluated using the same calibration data obtained from the Baihua chip. Subsequently, all benchmarking runs for QSteed, Qiskit, and Pytket were executed within a single continuous time block, ensuring that they operated under the same physical hardware state and thereby minimizing the impact of calibration drift. In addition, to avoid the influence of short-term chip fluctuations, each data point for algorithm-specific

Downloaded from https://spj.science.org on November 03, 2025

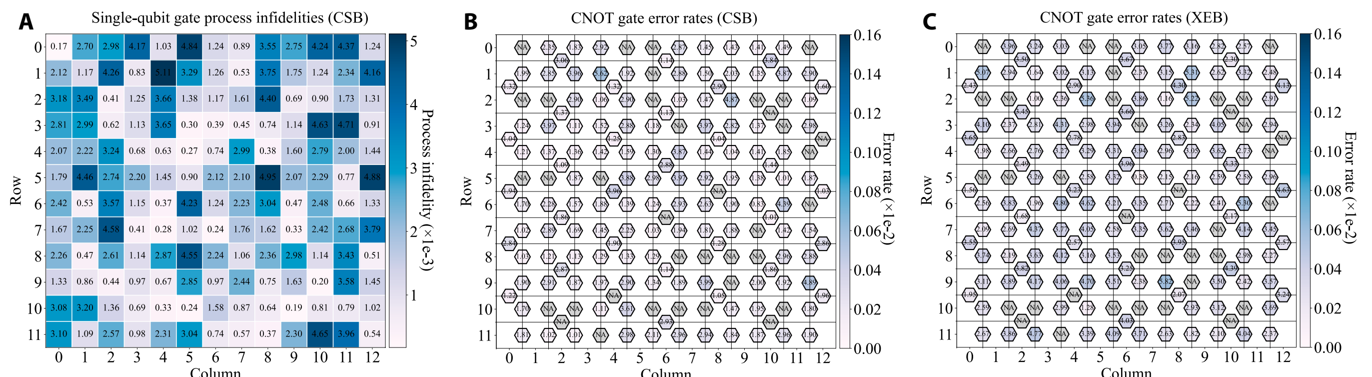

Fig. 4. Benchmarking results of the Baihua quantum chip. (A) Single-qubit process infidelities from CSB. (B) CNOT gate error rates from CSB. (C) CNOT gate error rates from XEB. NA means that the error rate is too high or gates are unavailable.

circuits is reported as the median of 5 consecutive runs, while results for random circuits are averaged over approximately 50 runs, providing statistically robust performance metrics. The compilation results are presented in Fig. 5. Across most evaluated benchmarks, QSteed achieves shorter compilation times while maintaining comparable or higher Hellinger fidelity. This advantage stems from its prebuilt VQPU database, which restricts optimization to an appropriate VQPU rather than the entire chip. In contrast, Qiskit and Pytket take the entire chip as input, which prolongs their compilation time. Moreover, although QSteed does not lead to shallower circuits or fewer gates, its noise- and topology-aware VQPU selection strategy, coupled with a noise-aware qubit mapping pass, systematically yields higher Hellinger fidelity than either Qiskit or Pytket, with most of the fidelity gains attributable to the former. Notably, the fidelity-first strategy compiles more quickly because it simply selects the VQPU with the highest aggregate fidelity, whereas the structure-first strategy incurs additional overhead to compute graph-similarity scores in order to identify the best-matching VQPU. However, when a circuit is exactly isomorphic to a candidate VQPU, such as the linear ising-n10 circuit, the structure-first strategy achieves higher Hellinger fidelity. Detailed results for QSteed under other compilation strategies are provided in Fig. S6.

For larger-scale quantum circuits, direct execution on physical hardware becomes unreliable due to the cumulative impact of hardware noise, and simulation for obtaining ideal results presents significant computational demands. Therefore, we use the circuit cost function, C, as a proxy for the expected circuit fidelity to evaluate compilation performance for larger circuits. The upper panel of Fig. 6 shows benchmarking results for circuits scaling from a dozen to over a hundred of qubits. Based on characterization data from the Baihua quantum processor, parameters in Eq. (5) were set to $F_i^{1q} = 0.996$ and $K = 0.995$. The analysis demonstrates that for moderate-scale circuits (up to ≈50 qubits), QSteed employing the fidelity-first strategy offers the shortest compilation times while maintaining circuit cost C comparable to, or lower than, that of Qiskit and Pytket. However, as circuit size increases, QSteed's compilation speed advantage over Qiskit gradually wanes. This is primarily because Qiskit has migrated its performance-critical kernels to Rust, wrapped by a Python interface, whereas QSteed has so far refactored only the DAG and qubit mapping modules in C++. Secondly, particularly when the circuit's qubit count approaches the chip's full capacity, searching for an optimal mapping over the entire processor becomes essentially as costly as searching within the VQPU subspace. Moreover, for large-scale circuits, Qiskit generally achieves lower C. This is primarily attributed to its deeper optimization passes, such as those utilizing commutation relations and advanced circuit synthesis, while the current QSteed only employs optimizations like gate elimination and merging.

To validate QSteed's robustness across diverse processor architectures, we conducted the same simulations on Google's 105-qubit superconducting Willow processor, which features a 2D grid topology (using the 97-qubit data available in [4]). Employing chip parameters from [4], the constants in Eq. (5) were set to $F_i^{1q} = 0.9996$ and $K = 0.995$. The lower panel of Fig. 6 shows the results. On Willow, we observed similar trends to those on Baihua: QSteed demonstrated an advantage in compilation speed while achieving comparable or superior circuit cost for small- to medium-scale circuits.

These results indicate that QSteed's virtualization mechanism and select-then-compile workflow are robust. Moreover, the consistent benefits observed on 2 distinct superconducting

Downloaded from https://spj.science.org on November 03, 2025

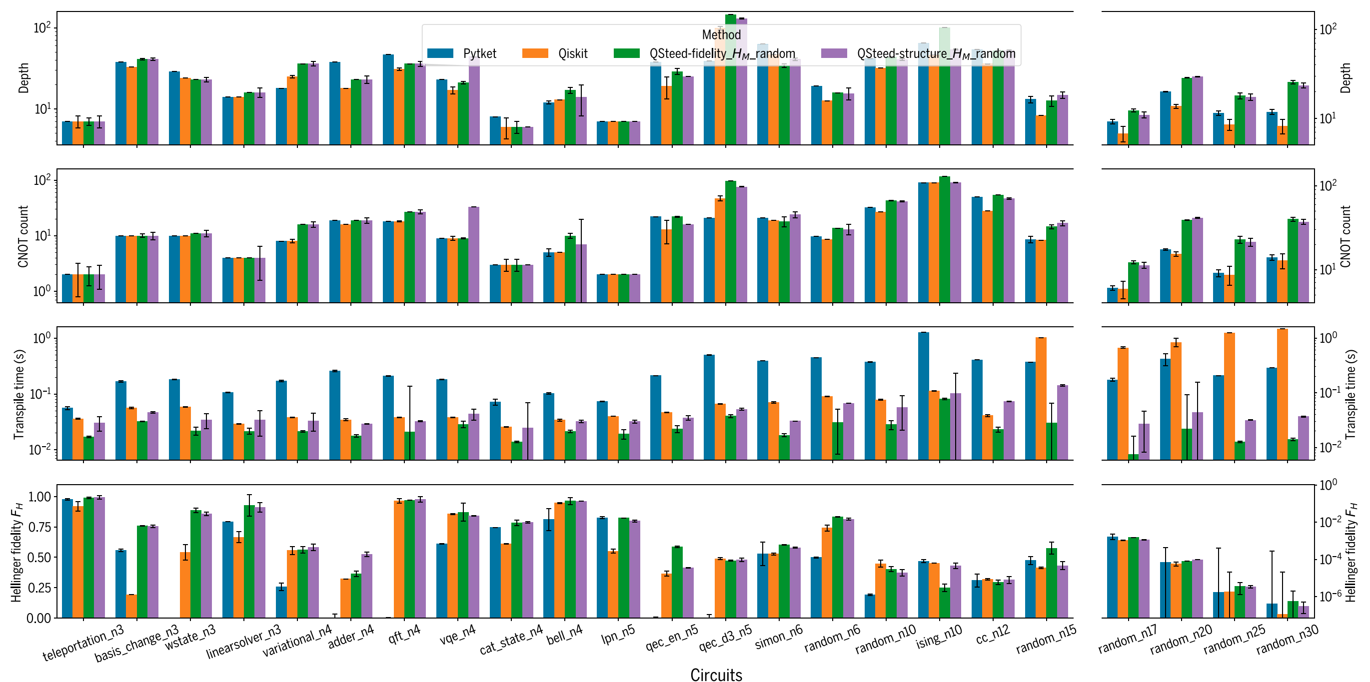

Fig. 5. Performance comparison of different quantum compilers on the Baihua quantum processor. The results for each benchmark circuit represent the median of 5 runs, while the results for random types are averaged over more than 50 runs of randomly generated circuits. Error bars indicate the standard error of the mean. The vertical axes, from top to bottom, represent the compiled circuit depth, the number of CNOT gates, the transpilation time, and the Hellinger fidelity F_H (higher is better). Except for the random circuits, the circuit names (trailing number denotes qubits) on the horizontal axis are derived from the QASMBench benchmarking suite [56].

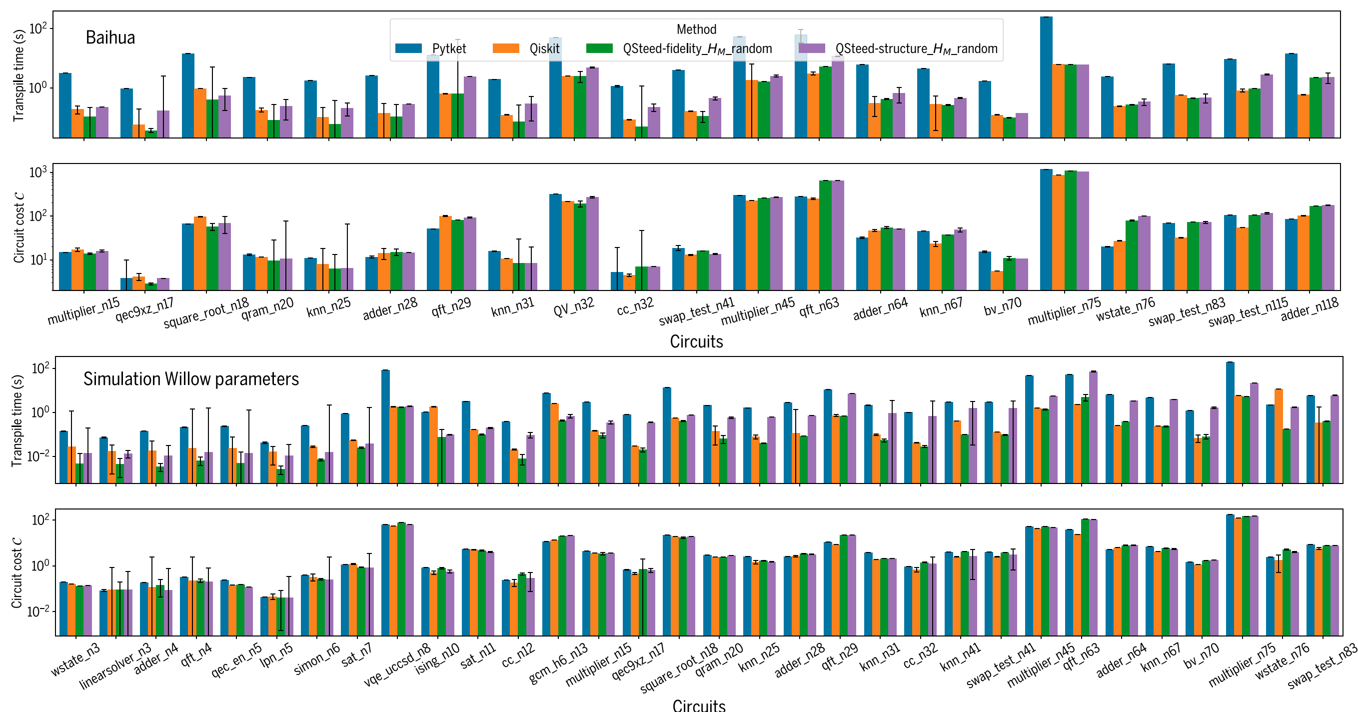

Fig. 6. Simulation results. For large-scale quantum circuits, the sampling results obtained from NISQ devices are unreliable, and obtaining ideal results from simulators is also challenging, which hinders the calculation of the Hellinger fidelity F_H. Therefore, we numerically simulate the circuit cost function C for each compiled circuit based on the qubit fidelity information of the Baihua quantum processor. To further evaluate the robustness of our compilation framework, we also compute C based on the Willow quantum processor parameters reported in [4]. The upper and lower panels respectively present the performance comparison of various quantum compilers on the Baihua processor and comparison of the simulation results based on data of Willow processor. For each benchmark circuit, results represent the median over 5 runs. Error bars indicate the standard error of the mean. Lower values of transpilation time and the circuit cost function C indicate better compilation performance. The circuit names (trailing number denotes qubits) along the horizontal axis are taken from the QASMBench benchmarking suite [56].

processor structures indicate that the framework's advantages are not confined to a single hardware topology, although validating its applicability across other quantum hardware platforms remains future work.

Discussion

In this work, we have designed and implemented QSteed, a novel compilation architecture that abstracts complex quantum hardware into a database of high-quality VQPUs. The data associated with these VQPUs are updated upon hardware recalibration. This abstraction obviates the need for the compiler to directly interface with the entire complex hardware backend. Instead, it maps user tasks to the optimal VQPU within the database, thereby enabling an efficient select-then-compile workflow. Experimental and simulation results demonstrate that the QSteed compiler consistently outperforms mainstream compilers such as Qiskit and Pytket on most benchmark circuits, showing strong practicality.

We also acknowledge that QSteed faces scalability challenges when considering future fault-tolerant quantum computers that may comprise thousands or even millions of qubits. At such scales, even the offline precomputation of the quantum resource virtualization database could become a bottleneck, suggesting that it may be necessary for more scalable or parallelizable heuristics for optimal VQPU identification. Potential directions include restricting the search depth, exploiting regularities in chip topologies for optimization, and parallelizing the search from multiple starting points.

Another limitation concerns calibration drift. Although the VQPU database stays dynamically synchronized with calibration data, if the backend hardware is not recalibrated in a timely manner, our current system architecture cannot detect parameter drift. This is a general issue that cannot be fully resolved at the compilation or virtualization level alone, but requires coordination with hardware benchmarking and automated calibration systems. In future work, we plan to integrate our system architecture with such benchmarking and calibration mechanisms to mitigate this limitation.

In addition, although our implementation and validation have primarily targeted superconducting processors, the architectural paradigm of QSteed holds potential for broader applicability. However, extending QSteed to other quantum hardware platforms requires addressing 2 key challenges:

First, applying QSteed's resource virtualization mechanism to other hardware platforms necessitates the identification of high-quality subregions within the specific hardware architecture and the corresponding construction of a VQPU database. Below, we explore possible adaptation pathways for several leading platforms: For one-dimensional linear ion trap, although they are logically fully connected, the fidelity of 2-qubit gates varies substantially with ion position [5]. Therefore, a fidelity-first strategy could still be envisioned to identify and extract SubQPUs composed of ions linked by high-fidelity connections. For modular quantum charge-coupled device (QCCD) architectures, which consist of multiple traps where ions are fully connected within each trap but have limited connectivity between traps [51], we can foresee that a degree-first strategy

Downloaded from https://spj.science.org on November 03, 2025

would be an effective heuristic for finding the most highly connected physical regions to serve as SubQPUs. For zoned architecture neutral atom devices, the hardware is divided into entanglement, storage, and readout zones based on the laser-addressable range [8,10]. To execute a 2-qubit gate, relevant atoms must be moved from a storage zone to an entanglement zone and returned afterward. Since atom transport error is proportional to the distance moved, it would be necessary to design a completely new heuristic, such as a distance-first strategy. This strategy, as its name suggests, would iteratively select atoms with the shortest required movement distance when constructing substructures. All generated SubQPUs could still be abstracted as VQPUs and stored in the same database.

Second, enabling the select-then-compile workflow requires designing corresponding VQPU selection strategies for different hardware platforms. For linear ion trap and QCCD architectures, fidelity-first and structure-first strategies may still prove effective, but their efficacy must be further validated on the specific hardware. For zoned architecture neutral atom devices, one could prioritize VQPUs that are sorted according to the distance-first strategy. Once a target VQPU is determined, the subsequent prerequisite is to design a hardware-specific transpilation process. Although the transpiler's modular pipeline architecture is conceptually general, the qubit mapping step must be tailored to the specific hardware. Unlike superconducting processors with fixed and limited connectivity, trapped-ion systems offer logical all-to-all connectivity; however, in scalable QCCD structures, interactions between multiple traps rely on physical ion shuttling, which makes minimizing this slow and error-prone movement the primary focus of the mapping strategy [52]. For neutral-atom arrays, which achieve dynamically reconfigurable connectivity via optical tweezers, the qubit mapping objective is to efficiently utilize atom movement and maximize parallelism [53,54].

Looking forward, the architectural principles embodied by QSteed, namely, resource virtualization and the select-then-compile paradigm, offer a promising path toward scalable, hardware-aware quantum compilation across heterogeneous physical platforms. This abstraction layer not only facilitates compiler portability but also creates opportunities for a unified compilation workflow in heterogeneous quantum clusters, a critical step toward the broader practical deployment of quantum computing systems.

Methods

We compared QSteed with the most widely used quantum compilation software, including Qiskit (v1.2.4) [21] and Pytket (v1.33.1) [24]. In the Qiskit and Pytket compilation frameworks, the entire quantum chip is required as the input for the compiler, whereas the QSteed compilation framework selects an appropriate VQPU from the resource virtualization database before performing the transpilation. To represent their best-effort performance, we configure Qiskit to its highest optimization level (optimization_level = 3), and Pytket follows the compilation flow adopted in MQTbench [55]. QSteed employs different compilation strategies, denoted as QSteed-*x*_*y*_*z*, where *x*, *y*, and *z* represent the VQPU selection strategy, the routing heuristic, and the initial layout method, respectively.

We selected algorithmic circuits from the QASMBench suite [56] and generated random circuits using gates like CNOT, RZZ, and RX to form our benchmark set. Each algorithmic benchmark was compiled and executed 5 times, with the median value taken as the performance metric. For random benchmarks, the metric was computed as the average over more than 50 circuit instances. Specifically, we adopted the following evaluation metrics.

Compilation time: For compilers designed for hardware deployment, shorter compilation times effectively reduce user waiting times and improve hardware utilization.

Circuit depth: Due to the limited coherence time of qubits in NISQ devices, shallower circuits tend to produce more reliable computational results.

Gate count: Different quantum processors may employ different native gate sets. Here, we uniformly assume the native gate set is {CNOT, RX, RY, RZ}, and we primarily focus on the number of CNOT gates after compilation, as their error rates are typically an order of magnitude higher than those of single-qubit gates.

Hellinger fidelity: We use the Hellinger fidelity, F_H, to quantify the actual fidelity of a compiled circuit. It is defined from the Hellinger distance d_H as [57]:

$$F_H = \left(1 - d_H^2\right)^2 = \left(\sum_{s\in\{0,1\}^n} \sqrt{P_s^{\exp} P_s^{\text{ideal}}}\right)^2, \tag{3}$$

where $P^{\exp}$ and P^{ideal} are the experimental and ideal probability distributions, respectively. This metric intuitively quantifies the deviation between experimental and ideal outcomes. All ideal results in this work are obtained using the Qiskit simulator (for circuits with up to 15 qubits) [21] or the NVIDIA CUDA-Q simulator (for circuits with more than 15 qubits).

Circuit cost function: Due to the performance limitations of NISQ hardware, obtaining reliable results for large-scale circuits through hardware sampling becomes extremely challenging. Under such conditions, we adopt a circuit cost function [58] as a proxy to evaluate compiler performance. To this end, we assume a simplified noise model: Markovian gate errors without temporal correlations, no crosstalk between parallel operations, decoherence errors approximated as an exponential function of the circuit depth, and the neglect of measurement errors as well as other hardware-specific imperfections. Under these assumptions, the effective fidelity of the circuit can be approximately defined as:

$$F_{\text{eff}} = K^D \prod_i F_i^{1q} \prod_j F_j^{2q}, \tag{4}$$

where D denotes circuit depth, K is a factor that penalizes deep circuits, and F^{1q}/F^{2q} represent single/2-qubit gate fidelitie. This metric is similar to the cost function proposed by IBM Quantum in [59]. To avoid vanishingly small values arising from multiplicative products, we further take the negative logarithm, leading to the circuit cost function C [58]:

$$C = -\log F_{\text{eff}} = -D\log K - \sum_i \log F_i^{1q} - \sum_j \log F_j^{2q}. \tag{5}$$

In this work, the fidelity of single-qubit gates is set to the average value, while the fidelity of 2-qubit gates is based on their individual values. Theoretically, C is strictly negatively correlated with the effective fidelity, which itself serves as a reasonable approximation to the true hardware fidelity under the simplified noise model, capturing the dominant contributions from gate errors and decoherence. Empirical evidence

Downloaded from https://spj.science.org on November 03, 2025

from Fig. S7 shows that for circuits up to 30 qubits, the circuit cost $\mathcal{C}$ exhibits a clear negative correlation with the Hellinger fidelity F_H (Spearman rank correlation $\rho = -0.686$), demonstrating that circuits with higher F_H generally have lower $\mathcal{C}$. Accordingly, we adopt this metric as a proxy for the expected circuit fidelity in our simulation experiments.

Acknowledgments

We acknowledge the contributions to the QSteed code by H. Chen, J.-F. Zeng, and B.-K. Yuan, as well as the helpful discussions with Y. Gu.

Funding: This work is supported by the Beijing Natural Science Foundation (grant no. Z220002), the National Natural Science Foundation of China (grant no. 92365111), the Innovation Program for Quantum Science and Technology (grant no. 2021ZD0302400), and the National Natural Science Foundation of China (grant nos. 92365206 and T2121001).

Author contributions: D.E.L. initialized and supervised the project. H.-Z.X., M.-J.H., and D.E.L. developed the concept and the structure of the quantum compilation framework. H.-Z.X. developed the software package with the help from M.-J.H., Z.-A.W., Y.-L.F., Y.C., X.Z., J.W., W.-F.Z., and Y.-X.J. X.-D.C. developed the ErrorGnoMark software for quantum benchmarking. H.-Z.X. and D.E.L. prepared the manuscript. Y.J., H.Y., and H.F. contributed to the discussion and analysis of the results.

Competing interests: The authors declare that they have no competing interests.

Data Availability

The data supporting the plots within this paper are available from the corresponding author upon reasonable request. The code for QSteed is available at: https://github.com/BAQIS-Quantum/qsteed.

Supplementary Materials

Sections S1 to S5
Figs. S1 to S11

References

1. Arute F, Arya K, Babbush R, Bacon D, Bardin JC, Barends R, Biswas R, Boixo S, Brandao FGSL, Buell DA, et al. Quantum supremacy using a programmable superconducting processor. *Nature*. 2019;574(7779):505–510.
2. Gong M, Wang S, Zha C, Chen MC, Huang HL, Wu Y, Zhu Q, Zhao Y, Li S, Guo S, et al. Quantum walks on a programmable two-dimensional 62-qubit superconducting processor. *Science*. 2021;372(6545):948–952.
3. Zhang X, Jiang W, Deng J, Wang K, Chen J, Zhang P, Ren W, Dong H, Xu S, Gao Y, et al. Digital quantum simulation of Floquet symmetry-protected topological phases. *Nature*. 2022;607(7919):468–473.
4. Google Quantum AI and Collaborators. Quantum error correction below the surface code threshold. *Nature*. 2024;638:920–926.
5. Wright K, Beck KM, Debnath S, Amini JM, Nam Y, Grzesiak N, Chen J-S, Pisenti NC, Chmielewski M, Collins C, et al. Benchmarking an 11-qubit quantum computer. *Nat Commun*. 2019;10(1):5464.
6. Pogorelov I, Feldker T, Marciniak CD, Postler L, Jacob G, Krieglsteiner O, Podlesnic V, Meth M, Negnevitsky V, Stadler M, et al. Compact ion-trap quantum computing demonstrator. *PRX Quant*. 2021;2(2):Article 020343.
7. Hou Y-H, Yi Y-J, Wu Y-K, Chen Y-Y, Zhang L, Wang Y, Xu Y-L, Zhang C, Mei Q-X, Yang H-X, et al. Individually addressed entangling gates in a two-dimensional ion crystal. *Nat Commun*. 2024;15(1):9710.
8. Bluvstein D, Levine H, Semeghini G, Wang TT, Ebadi S, Kalinowski M, Keesling A, Maskara N, Pichler H, Greiner M, et al. A quantum processor based on coherent transport of entangled atom arrays. *Nature*. 2022;604(7906):451–456.
9. Evered SJ, Bluvstein D, Kalinowski M, Ebadi S, Manovitz T, Zhou H, Li SH, Geim AA, Wang TT, Maskara N, et al. High-fidelity parallel entangling gates on a neutral-atom quantum computer. *Nature*. 2023;622(7982):268–272.
10. Bluvstein D, Evered SJ, Geim AA, Li SH, Zhou H, Manovitz T, Ebadi S, Cain M, Kalinowski M, Hangleiter D, et al. Logical quantum processor based on reconfigurable atom arrays. *Nature*. 2024;626(7997):58–65.
11. Zhong H-S, Wang H, Deng YH, Chen MC, Peng LC, Luo YH, Qin J, Wu D, Ding X, Hu Y, et al. Quantum computational advantage using photons. *Science*. 2020;370(6523):1460–1463.
12. Deng T-H, Gu YC, Liu HL, Gong SQ, Su H, Zhang ZJ, Tang HY, Jia MH, Xu JM, Chen MC, et al. Gaussian boson sampling with pseudo-photon-number-resolving detectors and quantum computational advantage. *Phys Rev Lett*. 2023;131:Article 150601.
13. Cao Y, Romero J, Olson JP, Degroote M, Johnson PD, Kieferová M, Kivlichan ID, Menke T, Peropadre B, Sawaya NPD, et al. Quantum chemistry in the age of quantum computing. *Chem Rev*. 2019;119(19):10856–10915.
14. Wei S, Li H, Long G. A full quantum eigensolver for quantum chemistry simulations. *Research*. 2020;2020:1486935.
15. Biamonte J, Wittek P, Pancotti N, Rebentrost P, Wiebe N, Lloyd S. Quantum machine learning. *Nature*. 2017;549(7671):195–202.
16. Zhou M-G, Liu ZP, Yin HL, Li CL, Xu TK, Chen ZB. Quantum neural network for quantum neural computing. *Research*. 2023;6:0134.
17. Peral-Garcia D, Cruz-Benito J, Garcia-Peñalvo FJ. Systematic literature review: Quantum machine learning and its applications. *Comput Sci Rev*. 2024;51:Article 100619.
18. Blekos K, Brand D, Ceschini A, Chou CH, Li RH, Pandya K, Summer A. A review on quantum approximate optimization algorithm and its variants. *Phys Rep*. 2024;1068:1–66.
19. Fauseweh B. Quantum many-body simulations on digital quantum computers: State-of-the-art and future challenges. *Nat Commun*. 2024;15(1):2123.
20. Di Meglio A, Jansen K, Tavernelli I, Alexandrou C, Arunachalam S, Bauer CW, Borras K, Carrazza S, Crippa A, Croft V, et al. Quantum computing for high-energy physics: State of the art and challenges. *PRX Quant*. 2024;5: Article 037001.
21. Javadi-Abhari A, Treinish M, Krsulich K, Wood CJ, Lishman J, Gacon J, Martiel S, Nation PD, Bishop LS, Cross AW, et al. Quantum computing with qiskit. arXiv. 2024. https://arxiv.org/abs/2405.08810
22. Cirq Developers. Cirq. 2024.
23. Smith RS, Curtis MJ, Zeng WJ. A practical quantum instruction set architecture. arXiv. 2016. https://arxiv.org/abs/1608.03355

Downloaded from https://spj.science.org on November 03, 2025

24. Sivarajah S, Dilkes S, Cowtan A, Simmons W, Edgington A, Duncan R. T|ket⟩: A retargetable compiler for NISQ devices. *Quant Sci Technol*. 2020;6(1):Article 014003.
25. Dou M, Zou T, Fang Y, Wang J, Zhao D, Yui L, Chen B, Guo W, Li Y, et al. Qpanda: high-performance quantum computing framework for multiple application scenarios. arXiv. 2022. https://arxiv.org/abs/2212.14201
26. Xu M, Li Z, Padon O, Lin S, Pointing J, Hirth A, Ma H, Palsberg J, Aiken A, Acar UA, et al. Quartz: Superoptimization of quantum circuits. In: *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. New York (NY): Association for Computing Machinery; 2022.
27. Younis E, Iancu CC, Lavrijsen W, Davis M, Smith E. Berkeley quantum synthesis toolkit (bqskit) v1. Computer Software. 2021. https://doi.org/10.11578/dc.20210603.2
28. Xu H-Z, Zhuang W-F, Wang Z-A, Huang K-X, Shi Y-H, Ma W-G, Li T-M, Chen C-T, Xu K, Feng Y-L, et al. Quafu-qcover: Explore combinatorial optimization problems on cloud-based quantum computers. *Chin Phys B*. 2024;33(5):Article 050302.
29. Farhi E, Goldstone J, Gutmann S. A quantum approximate optimization algorithm. arXiv. 2014. https://arxiv.org/abs/1411.4028
30. Cross AW, Bishop LS, Smolin JA, Gambetta JM. Open quantum assembly language. arXiv. 2017. https://arxiv.org/abs/1411.4028
31. Fu X, Riesebos L, Rol MA, Van Straten J, Van Someren J, Khammassi N, Ashraf I, Vermeulen RFL, Newsum V, Loh KKL, et al. eqasm: An executable quantum instruction set architecture. In: *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. Washington (DC): IEEE; 2019. p. 224–237.
32. Krantz P, Kjaergaard M, Yan F, Orlando TP, Gustavsson S, Oliver WD. A quantum engineer's guide to superconducting qubits. *Appl Phys Rev*. 2019;6(2):021318.
33. Niu S, Todri-Sanial A. Enabling multi-programming mechanism for quantum computing in the NISQ era. *Quantum*. 2023;7:925.
34. Li S, Zhou X, Feng Y. Qubit mapping based on subgraph isomorphism and filtered depth-limited search. *IEEE Trans Comput*. 2020;70(11):1777–1788.
35. Park S, Kim D, Kweon M, Sim J-Y, Kang S. A fast and scalable qubit-mapping method for noisy intermediate-scale quantum computers. In: *Proceedings of the 59th ACM/IEEE Design Automation Conference*. IEEE; 2022. p. 13–18.
36. Ge Y, Wenjie W, Yuheng C, Kaisen P, Xudong L, Zixiang Z, Yuhan W, Ruocheng W, Junchi Y. Quantum circuit synthesis and compilation optimization: Overview and prospects. arXiv. 2024. https://arxiv.org/abs/2407.00736
37. Shervashidze N, Schweitzer P, van Leeuwen EJ, Mehlhorn K, Borgwardt KM. Weisfeiler-Lehman graph kernels. *J Mach Learn Res*. 2011;12(77):2539–2561.
38. Kusyk J, Saeed SM, Uyar MU. Survey on quantum circuit compilation for noisy intermediate-scale quantum computers: Artificial intelligence to heuristics. *IEEE Trans Quant Eng*. 2021;2:1–16.
39. Zhu C, Wu X, Yang Z, Wang J, Wu A, Zheng S, Wang X. Quantum compiler design for qubit mapping and routing: A cross-architectural survey of superconducting, trapped-ion, and neutral atom systems. arXiv. 2025. https://arxiv.org/abs/2505.16891
40. Li G, Ding Y, Xie Y. 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*, ASPLOS '19. New York (NY): Association for Computing Machinery; 2019.
41. Zhu P, Guan Z, Cheng X. A dynamic look-ahead heuristic for the qubit mapping problem of nisq computers. *IEEE Transa Compu Aided Des Integ Circ Syst*. 2020;39(12): 4721–4735.
42. Huang C-Y, Lien C-H, Mak WK. Reinforcement learning and dear framework for solving the qubit mapping problem. In: Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design. New York (NY): IEEE; 2022. p. 1–9.
43. Murali P, Baker JM, Javadi-Abhari A, Chong FT, Martonosi M. 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*. New York (NY): Association for Computing Machinery; 2019. p. 1015–1029.
44. Niu S, Suau A, Staffelbach G, Todri-Sanial A. A hardware-aware heuristic for the qubit mapping problem in the nisq era. *IEEE Trans Quant Eng*. 2020;1:1–14.
45. Escofet P, Gonzalvo A, Alarcón E, Almudéver CG, Abadal S. Route-forcing: Scalable quantum circuit mapping for scalable quantum computing architectures. In: *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*. Montreal (QC, Canada): IEEE; 2024. p. 909–920.
46. Liu J, Zhan B, Wang S, Ying S, Liu T, Li Y, Ying M, Zhan N. Formal verification of quantum algorithms using quantum hoare logic. In: Dillig I, Tasiran S, editors. *Computer aided verification (CAV 2019)*. Cham (Switzerland): Springer International Publishing; 2019. p. 187–207.
47. Yu N, Palsberg J. Quantum abstract interpretation. In: *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2021*. New York (NY): Association for Computing Machinery; 2021.
48. Baltazar P, Chadha R, Mateus P. Quantum computation tree logic– Model checking and complete calculus. *Int J Quant Info*. 2008;06(02):219–236.
49. BAQIS-Quantum. Errorgnomark. 2025. https://github.com/BAQIS-Quantum/ErrorGnoMark/tree/errorgnomark-v2
50. Yanwu G, Zhuang W-F, Chai X, Liu DE. Benchmarking universal quantum gates via channel spectrum. *Nat Commun*. 2023;14:5880.
51. Pino JM, Dreiling JM, Figgatt C, Gaebler JP, Moses SA, Allman MS, Baldwin CH, Foss-Feig M, Hayes D, Mayer K, et al. Demonstration of the trapped-ion quantum ccd computer architecture. *Nature*. 2021;592(7853):209–213.
52. Upadhyay S, Saki AA, Topaloglu RO, Ghosh S. A shuttle-efficient qubit mapper for trapped-ion quantum computers. In: *Proceedings of the Great Lakes Symposium on VLSI*. New York (NY): Association for Computing Machinery; 2022. p. 305–308.
53. Wan-Hsuan Lin, Daniel Bochen Tan, and Jason Cong. Reuse-aware compilation for zoned quantum architectures based on neutral atoms. *In: 2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. Las Vegas (NV): IEEE; 2025. p. 127–142.
54. Huang C, Zhao X, Xu H, Zhuang W, Hu M-J, Liu DE, Wang J. Zap: Zoned architecture and parallelizable compiler for field programmable atom array. arXiv. 2024. https://arxiv.org/abs/2411.14037

Downloaded from https://spj.science.org on November 03, 2025

55. Quetschlich N, Burgholzer L, Wille R. MQT bench: Benchmarking software and design automation tools for quantum computing. *Quantum*. 2023;7:1062.
56. Li A, Stein S, Krishnamoorthy S, Ang J. Qasmbench: A low-level quantum benchmark suite for nisq evaluation and simulation. *ACM Trans Quant Comput*. 2023;4(2):1–26.
57. Geng A, Moghiseh A, Redenbach C, Schladitz K. A hybrid quantum image edge detector for the nisq era. *Quant Mach Intell*. 2022;4(2):15.
58. Kharkov Y, Ivanova A, Mikhantiev E, Kotelnikov A. Arline benchmarks: Automated benchmarking platform for quantum compilers. arXiv. 2022. https://arxiv.org/abs/2202.14025
59. Jurcevic P, Javadi-Abhari A, Bishop LS, Lauer I, Bogorin DF, Brink M, Capelluto L, Günlük O, Itoko T, Kanazawa N, et al. Demonstration of quantum volume 64 on a superconducting quantum computing system. *Quant Sci Technol*. 2021;6(2):Article 025020.

Downloaded from https://spj.science.org on November 03, 2025