DisMap: Calibration-Aware Distributed Compilation for Multi-Chip Quantum Systems

Zefan Du, Miguel Palma, Zijian Mo, Wenqi Wei, Juntao Chen, Rajkumar Buyya and Ying Mao

***Abstract*—Distributed quantum computing (DQC) offers a practical path to scaling beyond the qubit and wiring limits of monolithic processors, but current compilation flows provide limited support for heterogeneous inter-chip communication. In multi-chip systems, remote operations are often slower, noisier, and less reliable than local gates, and their cost depends strongly on link quality, latency, and device calibration. As a result, compilation decisions such as circuit partitioning, link selection, and qubit mapping become tightly coupled, yet are often handled separately or with simplified communication models in existing solutions. We present DisMap, a calibration-aware distributed compiler that integrates the heterogeneous costs of both inter- and intra-chip operations into a unified hierarchical cost model. Given per-chip topology, noise calibration data, and inter-chip link specifications, DisMap constructs a global system topology, applies adaptive circuit partitioning to minimize expensive cross-chip interactions, and performs iterative distributed qubit mapping to place logical operations on low-error qubits and route cross-chip communication over high-quality links. Evaluated on realistic multi-chip topologies inspired by IBM superconducting hardware, DisMap achieves up to 21.9% higher circuit fidelity and up to 92.6% lower execution cost compared to state-of-the-art partitioning and mapping baselines, while maintaining practical compilation times across a wide range of circuit benchmarks.**

***Index Terms*—Distributed quantum compilation, Calibration-aware compilation, Multi-chip quantum systems**

I. Introduction

Current quantum processors remain in the Noisy Intermediate-Scale Quantum (NISQ) era, with available devices providing only hundreds to a few thousand physical qubits [1], far below the scale required for fault-tolerant quantum computing [2], [3]. This gap limits the practical deployment of quantum applications in domains such as cryptography, finance, and computational chemistry [4]–[7]. A widely studied path toward larger-scale systems is *distributed quantum computing* (DQC), in which multiple quantum processing units (QPUs) are interconnected and coordinated as a single coherent computational system [8], [9].

While DQC offers a hardware roadmap for scaling [10], it introduces new requirements for the quantum software stack. Once a quantum program spans multiple chips, the compiler must decide how to assign qubits to modules, and select inter-chip communication resources. These decisions are tightly coupled, yet most existing compilation flows are still designed primarily for single-chip execution or rely on simplified assumptions about inter-chip communication. As a result, current software stacks provide only limited support for noise-aware optimization across heterogeneous multi-chip systems.

Inter-chip connectivity can be realized through several classes of physical links, including short-range coherent couplers and waveguides that support coherent inter-chip gates [11], as in IBM's Flamingo architecture linking Heron R2 processors [10] (Figure 1), photonic flying-qubit links for entanglement distribution between spatially separated modules [12]–[14], microwave-to-optical transducers [15], and matter-based links that shuttle qubits between adjacent modules [16]. From a systems-software perspective, these technologies present a common characteristic: inter-chip operations are heterogeneous and calibration-sensitive. This suggests that compilation strategies may benefit from accounting for communication quality explicitly rather than assuming uniform performance across all inter-chip paths.

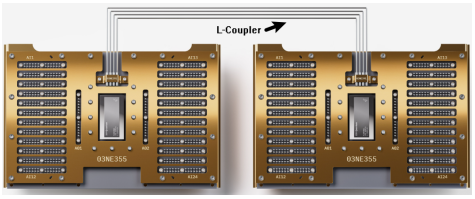


Fig. 1: IBM Flamingo architecture [10], linking two Heron R2 processors via an L-Coupler with approximately 3.5% inter-chip gate error.

This heterogeneity creates a clear opportunity for compiler optimization but also exposes a significant gap in existing distributed compilation methods. Prior work has explored modular compilation flows [17] and reinforcement-learning-based routing policies [18]. However, these approaches often focus on specific stages of the workflow or utilize simplified models of communication noise. Existing tools frequently treat inter-chip links as having fixed or uniform penalties, which may not capture the dynamic nature of hardware calibration data. This can lead to the selection of circuit partitions that are structurally sound but do not fully align with the current physical state of the heterogeneous hardware.

To address this gap, we propose DisMap, a calibration-aware distributed compilation framework for multi-chip quan-

Zefan Du, Miguel Palma, Wenqi Wei, Juntao Chen and Ying Mao are with the Department of Computer and Information Science, Fordham University, New York City. E-mail: {zdu19, mip2, wwei23, jchen504, ymao41}@fordham.edu

Zijian Mo is with BAE Systems, USA. E-mail: zijian.mo@baesystems.us

Rajkumar Buyya is with University of Melbourne, Australia. E-mail: rbuyya@unimelb.edu.au

tum systems. DisMap treats distributed compilation as a joint software optimization problem over circuit partitioning, inter-chip link selection, and physical qubit mapping. The key insight is to expose hardware heterogeneity directly to the compiler through a hierarchical cost model that captures gate error, link success probability, communication latency, and idle-time decoherence. Guided by this model, DisMap searches for circuit partitions and qubit mappings that minimize expensive cross-chip interactions, prioritize reliable communication paths, and align logical circuit structure with physical multi-chip topology.

This paper makes the following contributions:

- **A calibration-aware distributed compilation framework.** We present DisMap, a distributed quantum compiler that jointly optimizes circuit partitioning and qubit mapping for heterogeneous multi-chip systems, treating these stages as a coupled rather than independent problem.
- **A hierarchical cost model for distributed execution.** We formulate a unified optimization objective that incorporates intra-chip gate errors, inter-chip link success probabilities, communication latency, and idle-time decoherence, enabling the compiler to reason directly about the true cost of distributed execution.
- **An adaptive partitioning and mapping strategy.** Using the proposed cost model, DisMap selects high-quality communication links, avoids unreliable inter-chip paths, and aligns logical qubit groups with the physical multi-chip topology to minimize routing overhead and accumulated error.
- **An open implementation and empirical evaluation.** We implement DisMap in Qiskit and evaluate it using real IBM calibration data across homogeneous and heterogeneous multi-chip configurations. Results show up to **21.9%** higher circuit fidelity and up to **92.6%** lower execution cost compared to representative baselines, while maintaining practical compilation times.

This work positions distributed quantum compilation as a software systems problem shaped by hardware variability. By integrating calibration-aware cost modeling with joint partitioning and mapping, DisMap provides a practical compilation methodology for emerging multi-QPU platforms and advances the software toolchain needed for scalable distributed quantum computing.

II. Background and Motivation

This section reviews the hardware and software context for distributed quantum execution, covering distributed quantum computing, circuit partitioning, and compiler-level qubit mapping. We then motivate the need for calibration-aware distributed compilation.

A. Distributed Quantum Computing

Gate-based quantum computers execute algorithms through sequences of single- and two-qubit gates [19]. In superconducting platforms, these operations are typically implemented on monolithic chips with sparse coupling topologies such as heavy-hex lattices, where each physical qubit has only a small number of neighbors connected through tunable couplers [20]. Although this design supports high-fidelity local control, it limits scalability because chip area, wiring complexity, yield, and control overhead become increasingly difficult to manage as the number of qubits grows.

Distributed quantum computing (DQC) addresses this limitation by interconnecting multiple smaller chips and coordinating them as a larger computational system [8], [9]. Recent hardware progress has demonstrated the feasibility of such modular architectures. For example, IBM's Flamingo architecture links Heron R2 processors through meter-scale couplers with reported inter-chip error around 3.5% [10], while chiplet-style systems are also being explored by Rigetti [21]. These interconnects provide a practical near-term path toward larger quantum systems without requiring a single monolithic processor.

From a software perspective, however, DQC introduces a new compilation layer. A compiler for a multi-chip system must reason not only about local device connectivity, but also about when and how to use inter-chip communication resources. Unlike intra-chip gates, inter-chip operations are typically slower, noisier, and more heterogeneous. Their cost depends on link fidelity, latency, congestion, and idle-time decoherence. As a result, DQC is not only a hardware architecture problem, but also a compiler and systems-software problem.

B. Circuit Partitioning and Circuit Cutting

Circuit partitioning is a natural strategy for executing large circuits on hardware with limited qubit capacity. A circuit can be decomposed into smaller subcircuits that are assigned to different execution regions or different chips. One widely studied realization of this idea is *circuit cutting*, which uses wire cuts or gate cuts to divide a large quantum circuit into smaller fragments [22]. These fragments can then be executed separately on smaller devices.

Circuit cutting is attractive because it allows circuits exceeding a single device's qubit capacity to be executed in parts, and it can reduce local circuit depth, which may lower the accumulated effect of gate error, decoherence, and crosstalk. However, circuit cutting alone is not a complete solution for DQC. Standalone cutting introduces a classical reconstruction bottleneck: recovering the full output distribution from cut fragments generally incurs exponential overhead in the number of cuts [23]. More importantly, circuit cutting does not directly address the placement of partition boundaries relative to inter-chip link quality. By identifying natural cut points in the circuit, cutting can nonetheless guide the compiler toward communication-light partitions—revealing which cross-chip interactions are unavoidable and allowing the compiler to route those interactions over the most reliable available links. Realizing this benefit requires integrating cutting decisions with inter-chip topology awareness rather than treating them as a standalone pre-processing step.

For distributed quantum systems, partitioning remains essential, but the objective changes. Instead of cutting solely to fit capacity, the compiler should partition the circuit in a way that balances chip utilization, reduces expensive cross-chip interactions, and aligns communication-heavy subcircuits with high-quality inter-chip links. This makes circuit partitioning a software optimization problem tightly coupled to physical topology and runtime cost.

C. Compiler and Qubit Mapping

Qubit mapping assigns the logical qubits of a circuit to the physical qubits of the target hardware, and it is one of the most important compiler decisions affecting execution fidelity. Because hardware connectivity is sparse, logical two-qubit interactions often cannot be executed directly on the mapped qubits. To satisfy coupling constraints, the compiler inserts routing operations such as SWAPs, which increase both circuit depth and accumulated noise.

Finding an optimal qubit mapping is NP-complete [24], so practical compilers rely on heuristic strategies that trade off quality and compilation time. Existing mapping methods consider factors such as connectivity, gate fidelity, and crosstalk in order to place critical operations on high-quality qubits and reduce routing overhead [25]–[27]. These methods have been effective for monolithic devices, where communication cost is largely determined by a single-chip coupling graph.

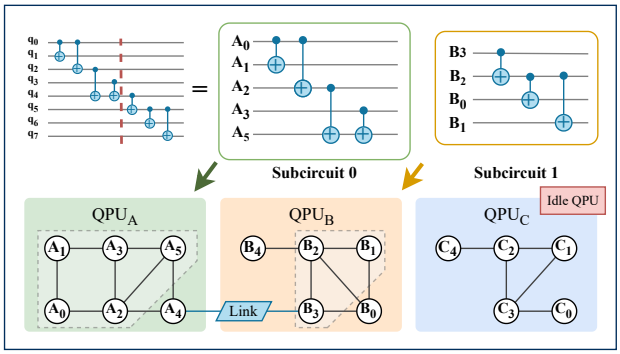
In multi-chip systems, however, the compiler must solve a more complex problem. Mapping decisions interact directly with partition boundaries and inter-chip communication. A poor assignment may increase the number of remote operations, overload weak links, or place critical subcircuits on noisy chip regions. Therefore, distributed compilation requires more than a direct extension of single-chip mapping: it requires joint reasoning over partitioning, chip assignment, and link-aware routing under dynamic calibration conditions.

D. Motivation

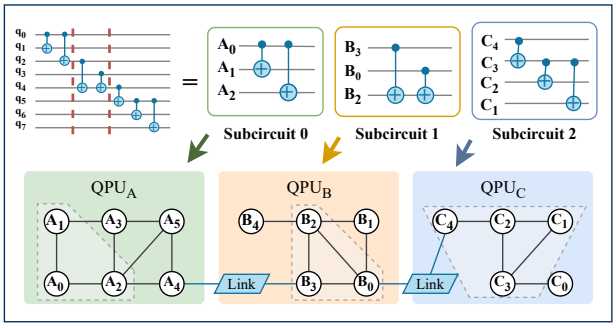
Circuit cutting and DQC provide complementary mechanisms for scaling beyond a single chip. Circuit cutting reduces circuit size to fit limited hardware, while DQC increases effective capacity by distributing computation across multiple connected QPUs. In principle, combining these two ideas enables larger quantum programs to execute without requiring one monolithic processor.

In practice, however, existing software support remains limited. Cutting-based approaches often incur exponential classical reconstruction overhead and typically assume fixed per-chip capacities. At the same time, many distributed compilation methods treat inter-chip communication with static or simplified cost models, even though real links differ in fidelity, latency, and reliability. As a result, current workflows may satisfy capacity constraints yet still produce poor system-level execution plans.

Figure 2 illustrates this challenge. A naive two-way partition over three available QPUs may fit the circuit within chip capacities, but it can leave one QPU idle and force more



(a) Two-way partitioning leaves one QPU idle and concentrates cross-chip traffic on fewer links.



(b) Three-way partitioning achieves full QPU utilization, shorter subcircuit depths, and reduced inter-chip traffic.

Fig. 2: Distributed compilation of an 8-qubit circuit across three chips (capacities $\{6, 5, 5\}$ qubits). (a) A naive two-way partition leaves one chip idle and stresses the inter-chip links. (b) An adaptive three-way partition utilizes all chips, shortens subcircuit depth, and avoids high-noise qubits, resulting in higher fidelity.

communication through costly or unreliable links. By contrast, a more adaptive three-way partition can better utilize all chips, shorten subcircuit depth, and route interactions through higher-quality resources. The result is improved fidelity and lower execution cost.

These observations motivate a calibration-aware software co-design for distributed quantum compilation. Rather than treating partitioning, chip assignment, and qubit mapping as independent stages, the compiler should optimize them jointly using explicit knowledge of hardware topology, link quality, latency, and decoherence. This is the design principle behind DisMap, which combines a hierarchical cost model with adaptive partitioning and link-aware mapping to improve the fidelity and efficiency of multi-chip quantum execution.

III. Related Work

A. Qubit Mapping and Compilation

Qubit mapping and routing for monolithic NISQ devices have been extensively studied. Early work formulated placement and routing as combinatorial optimization problems, including depth-optimal placement for arbitrary coupling topologies [28]. Since exact compilation is NP-hard, practical compilers rely on heuristics; representative examples include

SABRE, which introduced a bidirectional lookahead search for scalable qubit routing [29], and hardware-aware variants that incorporate gate duration, coherence times, or fidelity into the mapping objective [30]. Learning-based approaches have also emerged, including reinforcement-learning methods for mapping and routing [31] and attention-based graph neural network methods for larger architectures [32]. These works establish the algorithmic foundation for compiler optimization on a single coupling graph, but they do not explicitly model heterogeneous inter-chip communication or multi-chip topology.

B. Distributed Quantum Computing and Modular Architectures

Distributed quantum computing has become a major direction for scaling beyond monolithic processors. Surveys and roadmaps describe the transition to modular and networked quantum systems, highlighting the roles of entanglement distribution and compiler coordination [8], [33], [34]. On the hardware side, recent demonstrations confirm the feasibility of interconnected quantum modules, including chiplet-style superconducting systems and optical-network-linked processors [10], [13]. From a software perspective, DQC introduces a new compiler layer: quantum programs must be partitioned, placed, and routed across multiple QPUs while explicitly accounting for non-uniform communication cost and topology-dependent link quality.

C. Communication-Aware DQC Compilation

A significant line of work treats inter-QPU communication as the primary software concern in DQC. AutoComm identifies recurring communication patterns in distributed quantum programs and co-optimizes burst communication with circuit structure [35]. QuComm exposes collective communication primitives in distributed programs and optimizes them through dedicated compiler support [36]. MECH proposes a multi-entry communication highway for superconducting chiplet systems, trading ancilla resources for communication concurrency [37]. SwitchQNet studies communication scheduling for data-center-scale DQC with switch-based interconnects [38]. These works establish that communication should be treated as a first-class software concern, but they primarily target protocol-level scheduling, communication structure, or architectural concurrency rather than jointly solving circuit partitioning, physical qubit mapping, and calibration-aware link selection.

D. Distributed Compilation and Multi-QPU Mapping

Several recent compilers address the full DQC compilation problem. Ferrari et al. propose a modular compiler framework for distributed quantum computing that accounts for both device and network constraints [17]. Promponas et al. formulate DQC compilation as a reinforcement-learning problem, optimizing routing under stochastic entanglement generation [39]. Route-Forcing provides scalable mapping for large and modular architectures with low-depth routing heuristics [40], while follow-up work introduces Hungarian Qubit Assignment (HQA) and lower bounds on non-local communication [41]. Zhou et al. develop annealing- and clustering-based methods for heterogeneous DQC platforms [42], and LarQucut addresses large distributed circuits via joint cutting and mapping [43]. These works reflect rapid progress toward practical multi-QPU compilation, but most optimize routing, partitioning, or communication as separate stages.

E. Positioning of This Work

DisMap is most closely related to distributed compilation and communication-aware DQC. Unlike single-QPU methods, DisMap must optimize local routing alongside chip assignment and inter-chip communication simultaneously. Unlike communication-focused frameworks (AutoComm, QuComm, MECH), DisMap addresses the full compiler problem of jointly partitioning circuits and aligning them with physical topology under heterogeneous hardware.

The works most closely adjacent to DisMap are the RL-based compiler [39] and the modular compiler [17]. The RL approach learns inter-QPU routing policies under stochastic entanglement generation, but optimizes routing as a standalone stage and does not jointly solve circuit partitioning, link selection, and qubit mapping under dynamic calibration data. Modular compiler address both network and device constraints in a modular framework, but assume uniform inter-chip link costs. Route-Forcing [40] focuses on fast, low-depth mapping heuristics for modular architectures and does not perform calibration-aware link scoring. Besides, annealing- and clustering-based methods [42] for heterogeneous DQC lack integration with real-time calibration data.

DisMap differentiates from all of the above by performing *calibration-aware joint optimization* over (i) inter-chip link selection, (ii) circuit partitioning, and (iii) distributed qubit mapping in a single unified cost model that captures gate fidelity, link success probability, communication latency, and idle-time decoherence. This coupling is the central design principle of DisMap and the primary source of its empirical improvements over baselines.

IV. DisMap Solution Design

DisMap is a software framework for distributed quantum compilation on heterogeneous multi-chip systems. Its goal is to translate a logical quantum program into a complete distributed execution plan by jointly resolving three coupled compiler decisions: which inter-chip links to activate, how to partition the circuit into subcircuits, and how to map each subcircuit onto physical qubits. Unlike monolithic compilation, these decisions cannot be made independently: link fidelity, latency, and success rate vary significantly across the system, so partition boundaries, qubit placement, and communication routing are all interdependent. DisMap addresses this challenge through a calibration-aware design that exposes hardware heterogeneity directly to the compiler cost model and joint optimization loop.

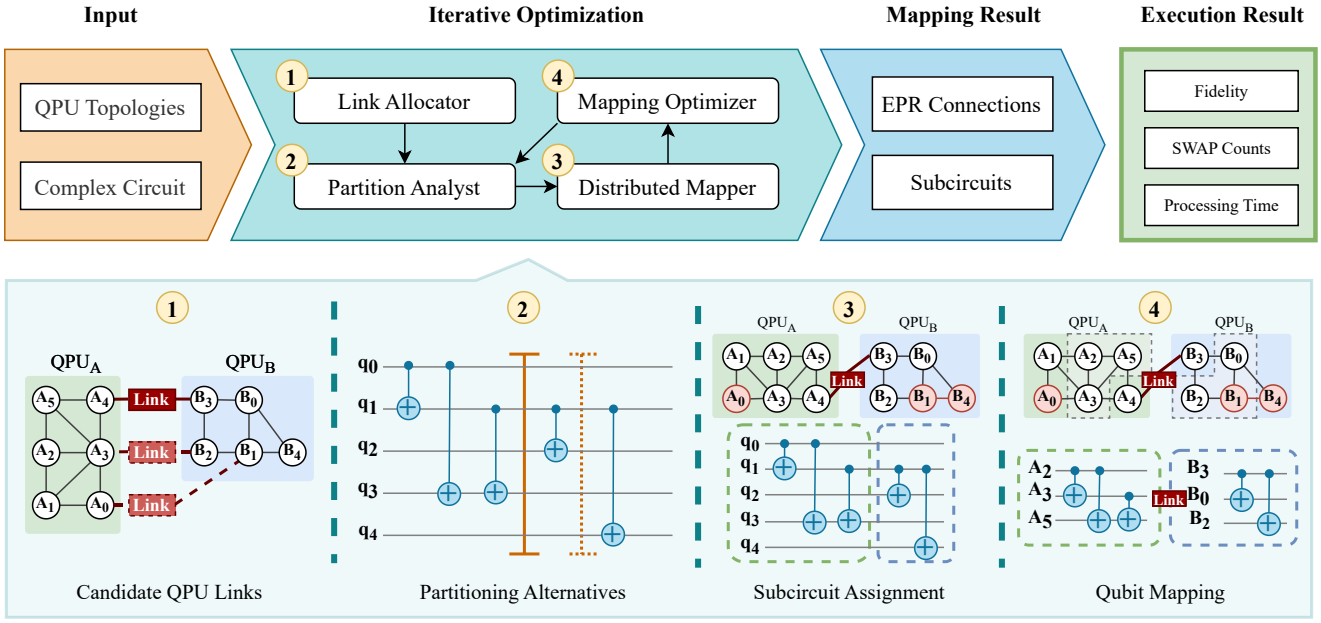


Fig. 3: **DisMap compilation workflow.** *Inputs:* Per-chip hardware topologies, calibration noise data, and a logical quantum circuit. *Step 1 (Link Allocator):* Score and select among candidate inter-chip links to construct the system topology (ST). *Step 2 (Partition Analyst):* Generate circuit partitioning candidates while avoiding high-error qubits (shown as red nodes in ST). *Step 3 (Distributed Mapper):* Assign subcircuits to chips and map them onto low-noise qubit regions (e.g., subcircuits 0 and 1 placed on Node 0 and Node 1, respectively). *Step 4 (Link-Aligned Mapping Optimizer):* Refine qubit mappings to minimize the hierarchical cost objective. The pipeline iterates over candidate link configurations until the lowest-cost distributed plan is found; subcircuits are then executed on their assigned chips and results are combined.

A. System Model

We model a distributed quantum processor as a set of chips $W = \{w_1, \ldots, w_n\}$. Each chip w_i is characterized by a hardware coupling topology T_i, qubit capacity Q_i, and a calibration-derived noise profile N_i. For each gate g executable on chip w_i, the calibration data expose a gate error estimate $\varepsilon(g)$ and gate latency $\tau(g)$. These quantities serve as software-visible inputs to the compiler, allowing DisMap to reason about execution cost beyond pure connectivity.

Inter-chip communication is represented by a set of physical links $L = \{\ell_1, \ldots, \ell_m\}$, corresponding to direct couplers or entanglement-assisted channels. For each chip pair (w_i, w_j), we denote by $\mathcal{L}_{ij} \subseteq L$ the candidate set of available links. For every link $\ell \in L$, the compiler is given:

- baseline link error $\varepsilon_{\text{lnk,base}}(\ell)$,
- baseline link latency $\mathcal{L}(\ell)$ (quantum + classical round-trip),
- per-attempt success rate $SR(\ell)$.

Given a logical circuit C, the compiler seeks three coupled outputs: a circuit partition $S = \{C_1, \ldots, C_k\}$, a link-selection strategy over available inter-chip resources, and a distributed physical mapping M. Together, these define a complete executable software plan for the target multi-chip system. The objective is to minimize total execution cost while preserving high end-to-end fidelity, by placing computation on reliable local resources and routing cross-chip interactions over the most suitable links.

B. Hierarchical Cost Formulation

DisMap formulates distributed compilation as a hierarchical optimization problem over two coupled cost terms: local intra-chip execution cost and inter-chip communication cost. Let circuit C be partitioned into k subcircuits $S = \{s_1, \ldots, s_k\}$, with each subcircuit s_i assigned to chip w_i under mapping M. The total compiler objective is

$$J(C; S, M) = \sum_{i=1}^{k} \mathcal{C}_{\text{local}}(s_i) + \mathcal{C}_{\text{inter}}(S), \tag{1}$$

where $\mathcal{C}_{\text{local}}$ captures intra-chip routing overhead and decoherence, and $\mathcal{C}_{\text{inter}}$ captures cross-chip communication cost under the selected links. This decomposition reflects the software structure of the problem: the compiler must simultaneously optimize local transpilation quality and global distributed communication cost.

1) Local (Intra-chip) Cost: For subcircuit s_i assigned to chip w_i, the compiler inserts intra-chip SWAP operations to satisfy the chip's coupling topology. Let $S_{\text{intra}}(s_i)$ denote the set of SWAPs induced by local routing. Each $SWAP(p \leftrightarrow q)$ is decomposed into three two-qubit interactions: $SWAP(p \leftrightarrow q) = \{\text{CX}(p,q), \text{CX}(q,p), \text{CX}(p,q)\}$. The total local routing cost is

$$\mathcal{C}_{\text{local}}(S) = \sum_{s_i \in S} \sum_{swap \in S_{\text{intra}}(s_i)} R(swap), \tag{2}$$

with calibration-derived components defined as follows. The decoherence rate of qubit x is $\gamma_x = \frac{1}{2T_{1,x}} + \frac{1}{2T_{2,x}}$; the duration of a single SWAP operation is $\Delta t_{swap} = \tau(swap)$, obtained directly from calibration; and the per-SWAP cost is

$$R(swap) = 3\varepsilon_c + \Big(1 - e^{-(\gamma_p+\gamma_q)\Delta t_{swap}/2}\Big), \tag{3}$$

where ε_c is the calibrated CNOT error for physical qubits p and q. The first term penalizes the three CX operations comprising the SWAP; the second term penalizes the decoherence accumulated on both qubits during that single SWAP's gate time.

This term makes routing overhead a calibration-aware quantity. Rather than penalizing all SWAPs equally, DisMap weighs each SWAP by its hardware-specific gate error and the decoherence accumulated on the participating qubits, so that local mapping decisions better reflect actual execution quality.

2) Inter-chip Communication Cost: Inter-chip operations are executed between qubit $q_i \in Q_i$ on chip w_i and qubit $q_j \in Q_j$ on chip w_j. For each chip pair (w_i, w_j), DisMap selects among the candidate links in $\mathcal{L}_{ij}$. Each communication operation introduces link-level gate error, expected latency, and idle-time decoherence on both participating qubits. The aggregate inter-chip cost over all selected links is

$$\mathcal{C}_{\text{inter}} = \sum_{\ell \in \mathcal{L}_{ij}} \Big[\lambda_f\, \varepsilon(q_i, q_j) + \lambda_t\, \mathbb{E}[\tau(q_i, q_j)] + \lambda_d\Big(1 - e^{-(\gamma_{q_i}+\gamma_{q_j})\Delta t_{(q_i,q_j)}/2}\Big)\Big] / SR(\ell), \tag{4}$$

where $\varepsilon(q_i, q_j)$ is the calibrated inter-chip operation error for qubit pair (q_i, q_j) over link ℓ; $\mathbb{E}[\tau(q_i, q_j)]$ is the expected communication latency, which depends on the link type (e.g., direct coupler versus entanglement distribution with stochastic attempts); $\Delta t_{(q_i,q_j)}$ is the communication duration during which both qubits accumulate decoherence; and $\gamma_{q_i}, \gamma_{q_j}$ are the corresponding qubit decoherence rates.

The division by $SR(\ell)$ is motivated by the stochastic nature of some inter-chip links: for an entanglement-based link with per-attempt success rate $SR(\ell)$, the number of attempts until success follows a geometric distribution with mean $1/SR(\ell)$, so the effective communication cost scales inversely with success rate. For direct-coupler links (where $SR(\ell) \approx 1$), this factor has negligible effect. The summation over $\ell \in \mathcal{L}_{ij}$ aggregates the cost contributed by each inter-chip operation assigned to the candidate link set for that chip pair, enabling the compiler to distribute traffic across multiple links.

The weights λ_f, λ_t, and λ_d balance the three cost components. To ensure commensurate contributions, each weight is set as the reciprocal of the corresponding hardware-typical scale: $\lambda_f = 1/\bar{\varepsilon}$, $\lambda_t = 1/\bar{\tau}$, and $\lambda_d = 1/\bar{d}$, where $\bar{\varepsilon}$, $\bar{\tau}$, and $\bar{d}$ are the median inter-chip error, latency, and decoherence penalty observed in the calibration dataset, respectively. This normalization places all three terms on a dimensionless, hardware-relevant scale.

This formulation treats cross-chip communication as a first-class compiler cost. Rather than imposing a uniform penalty at chip boundaries, DisMap exposes link heterogeneity directly in the optimization objective, enabling the compiler to prioritize lower-error and lower-latency routes, distribute traffic across multiple candidate links, and avoid unreliable communication paths when generating distributed execution plans.

C. System Overview

DisMap is a calibration-aware compiler pipeline for heterogeneous multi-chip superconducting systems. Given chip topologies T, qubit capacities, calibrated noise metrics, and inter-chip link specifications with success rates SR, the compiler determines: (i) which inter-chip links to activate, (ii) how to partition the logical circuit C into subcircuits, and (iii) how to map each subcircuit onto physical qubits to minimize the total objective in (1).

Figure 3 summarizes the workflow. DisMap first constructs a software-level system topology ST by merging all chip topologies T with candidate inter-chip links L. On this unified abstraction, the compiler explores partitioning alternatives and distributed qubit mappings that jointly minimize local SWAP overhead and inter-chip communication cost. The pipeline proceeds in three stages:

- **Adaptive link selection:** for each chip pair (w_i, w_j), DisMap scores candidate links $\ell \in \mathcal{L}_{ij}$ using calibrated error, latency, and success rate, and injects the selected links into ST;
- **Cost-aware circuit partitioning:** given ST, the compiler identifies partitions that minimize both cross-chip communication overhead and local routing cost, co-locating strongly interacting qubits on the same chip and assigning unavoidable cross-chip interactions to the highest-quality available links;
- **Distributed subcircuit mapping:** each subcircuit s_i is transpiled onto its assigned chip using a noise-aware pass that respects inter-chip boundaries and preferentially places critical operations on high-fidelity physical qubits.

The compiler iterates over candidate link configurations and retains the distributed execution plan with the lowest total cost.

D. Algorithmic Framework

DisMap is implemented as a three-module compiler workflow: **ConST**, **TopoCutter**, and **SubMapper** (Algorithm 1), wrapped inside the iterative optimization loop in Algorithm 2. Together, these modules translate a logical circuit into a complete distributed execution plan that minimizes the objective in (1).

ConST $(T, \mathcal{EP})$ constructs the global system topology ST as the graph union of all per-chip topologies T_i, augmented by inserting an edge for each inter-chip link endpoint pair in $\mathcal{EP}$. This produces a unified software-level coupling graph that the downstream modules can reason about as a single device.

TopoCutter (C, ST) determines the circuit partition. It sweeps over candidate subcircuit sizes sq from $SQ_{\min} = |C|$ (the circuit qubit count, i.e., the minimum possible subcircuit size) to $SQ_{\max} = |ST|$ (the total system capacity). For each size, SelectQubits(ST, sq) restricts each chip to the sq

Algorithm 1 DisMap Core Module Definitions (T: per-chip topologies; $\mathcal{EP}$: inter-chip link endpoint pairs)

Function ConST($T,\ \mathcal{EP}$):
  $ST \leftarrow \bigcup_{w_i \in W} T_i$ *// graph union of per-chip topologies*
for $(q_k^{w_i},\ q_l^{w_j})$ **in** $\mathcal{EP}$ **do**
  $ST \leftarrow ST \cup \{\text{edge}(q_k^{w_i},\ q_l^{w_j})\}$ *// add inter-chip link edge*
end for
Return ST

Function TopoCutter($C,\ ST$):
  $SQ_{\min} \leftarrow |C|$ *// min width = circuit qubit count*
  $SQ_{\max} \leftarrow |ST|$ *// max width = total system capacity*
  $S_{\text{best}} \leftarrow \emptyset;\ n_{\text{cut}}^{\text{best}} \leftarrow \infty$
for sq **in** $[SQ_{\min},\ SQ_{\max}]$ **do**
  // sweep candidate subcircuit sizes
  $ST_{\text{curr}} \leftarrow \text{SelectQubits}(ST,\ sq)$ *// keep sq lowest-noise qubits per chip*
  $S_{\text{cand}} \leftarrow \text{CircuitCut}(C,\ ST_{\text{curr}})$ *// topology-guided cutting [44]*
  if $|\text{InterCuts}(S_{\text{cand}})| < n_{\text{cut}}^{\text{best}}$ **then**
    $S_{\text{best}} \leftarrow S_{\text{cand}};\ n_{\text{cut}}^{\text{best}} \leftarrow |\text{InterCuts}(S_{\text{cand}})|$
  end if
end for
Return S_{best} *// partition with fewest inter-chip cuts*

Function SubMapper($S,\ ST$):
  $MS \leftarrow [\,]$
for s_i **in** S **do**
  $ms_i \leftarrow \text{Transpiler}(s_i,\ ST)$ *// noise-aware transpile onto assigned chip region*
  $\mathcal{C}_{\text{local}}(s_i) \leftarrow$ Eq. (2) *// per-SWAP intra-chip cost from calibration*
  $MS.\text{append}(ms_i)$
end for
Return $MS,\ \mathcal{C}_{\text{local}}$

lowest-noise available qubits, and $\text{CircuitCut}(C, ST_{\text{curr}})$ applies topology-guided Qiskit circuit cutting [44] to partition C into subcircuits that fit within those qubit groups. TopoCutter returns the partition that minimizes the number of inter-chip cuts, co-locating strongly interacting qubits on the same chip wherever the topology allows.

SubMapper (S, ST) maps each subcircuit s_i onto its assigned chip region in ST using a noise-aware transpilation pass that prefers low-error qubits and short routing paths to the link endpoint. The per-SWAP cost from (2) is evaluated for each mapped subcircuit.

Algorithm 2 drives the calibration-aware outer search. Candidate link sets are generated by scoring all boundary qubit pairs by combined link error and inverse success rate, then forming subsets of the top-k pairs at each chip boundary. For each candidate configuration, the three modules are invoked, the total cost (1) is evaluated, and the lowest-cost distributed execution plan is retained. This joint enumeration over link selection, circuit partitioning, and distributed qubit mapping is a heuristic exhaustive search; while it does not carry worst-case optimality guarantees, the search space is bounded by the number of link candidates (typically small: IBM Flamingo exposes one or two L-Coupler links per chip pair), ensuring practical compilation times as reported in Section V-B2.

Algorithm 2 DisMap Iterative Optimization over Link Configurations

Input: Circuit C, chips W, topologies T, noise profile N
// Step 1: generate candidate link sets
for each chip pair (w_i, w_j) **do**
  rank boundary qubit pairs $(q_k^{w_i}, q_l^{w_j})$ by $\varepsilon_{\text{lnk}} + 1/SR$ (ascending)
  retain top-k pairs as candidates *// k=2 in practice*
end for
$L \leftarrow$ all size-$\leq k$ subsets from the ranked pairs across all chip boundaries
$\mathcal{J}_{\text{best}} \leftarrow \infty;\ MS_{\text{best}} \leftarrow \emptyset$
// Step 2: evaluate each link configuration
for each $\mathcal{EP}_i \in L$ **do**
  $ST \leftarrow \text{ConST}(T,\ \mathcal{EP}_i)$
  $S \leftarrow \text{TopoCutter}(C,\ ST)$
  $MS,\ \mathcal{C}_{\text{local}} \leftarrow \text{SubMapper}(S,\ ST)$
  $\mathcal{C}_{\text{inter}} \leftarrow$ Eq. (4) evaluated on $(S,\ \mathcal{EP}_i,\ N)$
  $\mathcal{J}_{\text{curr}} \leftarrow \sum_i \mathcal{C}_{\text{local}}(s_i) + \mathcal{C}_{\text{inter}}$ *// Eq. (1)*
  if $\mathcal{J}_{\text{curr}} < \mathcal{J}_{\text{best}}$ **then**
    $\mathcal{J}_{\text{best}} \leftarrow \mathcal{J}_{\text{curr}};\ MS_{\text{best}} \leftarrow MS;\ ST_{\text{best}} \leftarrow ST$
  end if
end for
Execute MS_{best} on ST_{best}; obtain R and $\mathcal{F}_{\text{total}}$
Return R, $\mathcal{J}_{\text{best}}$, $\mathcal{F}_{\text{total}}$

V. Performance Evaluation

We evaluate DisMap on circuit fidelity, inter-chip operation count, intra-chip SWAP count, total cost, and compilation time across multi-chip systems with heterogeneous qubit configurations.

A. Implementation and Experimental Settings

DisMap is implemented in Python with Qiskit 1.2 [45], using IBM-Q noisy emulators [46] with real device calibration data on heavy-hex topologies: AlmadenV2 (20 qubits) and Auckland (27 qubits). Since physical chip-to-chip hardware is not yet broadly accessible for the heterogeneous configurations studied here, we emulate inter-chip links by scaling two-qubit gate error by $3.5\times$, doubling T_1/T_2 decoherence durations, and assigning link success rates of 0.90–0.95, consistent with IBM's reported Flamingo inter-chip error of approximately 3.5% [10]. This emulation captures the key asymmetry between intra- and inter-chip costs present in real modular hardware; the primary limitation is that it cannot reproduce link-specific congestion or correlated error that may appear in physical multi-chip systems. Each link configuration is parameterized independently to capture distinct noise and fidelity characteristics.

TABLE I: Representative noise parameters used in emulation. Intra-chip values are real IBM calibration medians

Parameter	Inter-chip	Inter-chip link
2Q gate error	0.3%–0.7%	3.5% [10]
Operation time (ns)	50–140	235 [10]
Success Rate SR	—	0.90–0.95

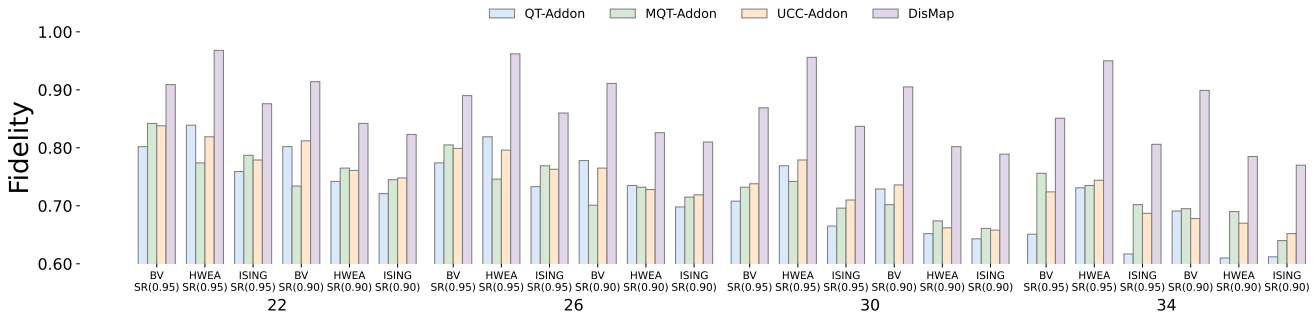


Fig. 4: Fidelity comparison on a two-chip AlmadenV2(20)×2 system for BV, HWEA, and ISING circuits (22–34 qubits) under two inter-chip link success-rate (SR) settings (0.90 and 0.95). Methods compared: QT-Addon, MQT-Addon, UCC-Addon, and DisMap.

Table I summarizes the representative per-chip and inter-chip noise parameters used in our evaluation. Median two-qubit gate errors are approximately 0.3%–0.7% (intra-chip) for single processor like AlmadenV2 and Auckland based on real IBM calibration snapshots; emulated inter-chip links are assigned gate error, operation time and success rate SR.

Circuit benchmarks range from 22–100 qubits and include: Bernstein–Vazirani (BV) [47] for shallow, structured circuits; Ripple-carry Adder (Adder) [48] for structured circuits with CNOT/Toffoli chains; Hardware Efficient Ansatz (HWEA) [49] for deep, densely entangling VQE-style workloads; Supremacy circuits [50] and Random circuits at fixed depth and width using Qiskit built-in utilities [51]. Fidelity analysis (Fig. 4) focuses on the 22–34 qubit range of the small two-chip system to isolate the effect of inter-chip link quality; cost, SWAP-count, and compilation-time results (Fig. 5 and Table II) cover the full 22–310 qubit range across all four multi-chip configurations. We report three metrics:

- **Fidelity** (higher is better): estimated post-compilation execution quality, computed from hardware-calibrated gate error rates combined with an exponential decoherence penalty for qubit idle time.
- **Cost** (lower is better): the total compiler objective in Eq. (1), capturing both inter- and intra-chip operation overhead.
- **Compilation time** (lower is better): wall-clock time for the complete compilation pipeline, including link selection, circuit partitioning, and qubit mapping.

As baselines, we compare against IBM's Qiskit transpiler (QT) [45], the Munich Quantum Toolkit (MQT) [26], and the Unitary Compiler Collection (UCC) [27]. Since none of these tools natively supports multi-chip compilation, we augment each with Qiskit Addon-Cutting [44], setting per-chip qubit capacity to match the hardware chip sizes. This pairing reflects the closest analog to a distributed execution strategy available for these single-chip tools and provides a conservative lower bound on what existing methods can achieve when extended to multi-chip settings.

Regarding comparison with dedicated DQC compilers: the RL-based framework of Promponas et al. [39] (open source at `CompilerDQC`) targets an abstract graph-based network model with explicit entanglement generation protocols and does not interface with IBM Fake Provider calibration data or heavy-hex topologies; adapting it to our evaluation setting would require non-trivial hardware translation. The CloudQC framework [52] similarly assumes a fixed-connectivity chiplet model and does not expose the dynamic link-selection or per-qubit noise parameterization used here. Ferrari et al. [17] assume uniform link costs. These architectural mismatches make direct metric comparison unreliable, as differences in results would conflate hardware assumptions with algorithmic quality; we therefore use the augmented single-chip baselines as a standardized comparison point and position DisMap qualitatively against dedicated DQC compilers in Section III.

B. Experimental Results

DisMap achieves consistently high fidelity and low cost across all circuits and chip configurations; the baselines exhibit substantially higher and more variable overhead.

1) Fidelity: As shown in Fig. 4, DisMap achieves the highest fidelity across all circuits and success-rate (SR) settings. Gains are most pronounced at SR = 0.95. For HWEA(34), fidelity improves from 73.1% (QT-Addon) to 95.0% (DisMap), a 21.9% improvement attributable to reducing inter-chip operations from 4 to 1. BV(34) increases from 76.9% to 95.6% with a corresponding cost reduction ($23 \rightarrow 18$). Even at SR = 0.90, DisMap maintains double-digit improvements by simultaneously minimizing inter-chip operations and placing subcircuits on low-noise physical qubits.

These fidelity gains are consistent with the design objectives of DisMap's hardware-aware compilation pipeline. The primary mechanism is adaptive partitioning: strongly interacting logical qubits are co-located on the same chip, reducing the number of expensive inter-chip operations (e.g., $4 \rightarrow 1$ for HWEA(34)). When cross-chip interactions are unavoidable, the link-aligned mapping routes them over the most reliable links, as selected by the hierarchical cost model. Physical qubits with low two-qubit error rates and long coherence times are preferred for critical operations. Together, these strategies reduce accumulated gate error and decoherence,

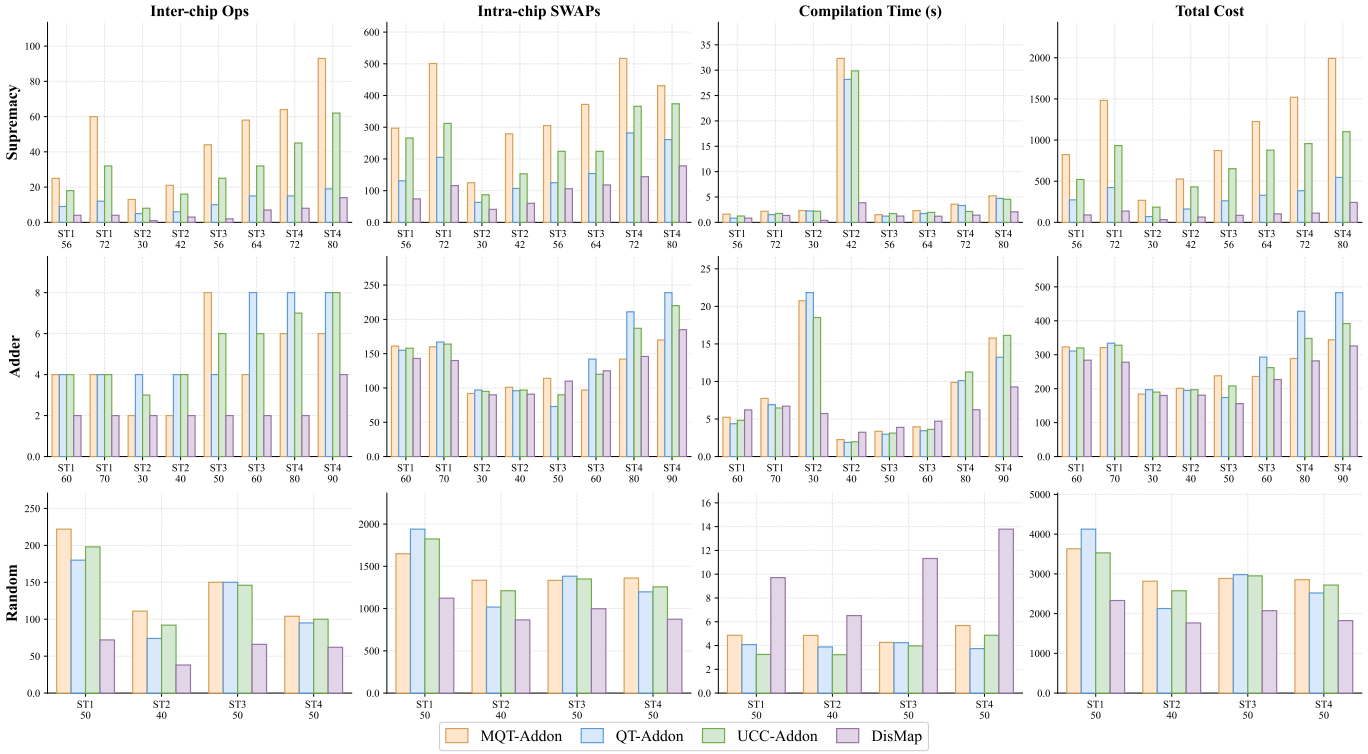


Fig. 5: Inter- and intra-chip SWAP counts, compilation time (s), and total cost for Adder, Supremacy, and Random circuits across four multi-chip topology configurations. ST1 = Auckland(27)×3; ST2 = AlmadenV2(20)+Auckland(27); ST3 = AlmadenV2(20)×2+Auckland(27); ST4 = AlmadenV2(20)×2+Auckland(27)×2.

consistent with the observed fidelity improvements as circuit size increases.

2) Cost Analysis: Fig 5 shows that DisMap achieves the lowest total cost across all evaluated circuits and multi-chip configurations, both homogeneous and heterogeneous. The improvements derive from reductions in both inter-chip operations and intra-chip SWAP overhead.

a) Inter-Chip Operation Reduction: The largest inter-chip operation reductions occur on heterogeneous multi-chip configurations. For the Supremacy(56) circuit on ST3, DisMap reduces the inter-chip operation count from 44 (MQT) to 2, a 95.5% reduction. These gains arise because DisMap places partition boundaries to minimize the number of cross-chip interactions, routing only the unavoidable ones through the highest-quality available links.

b) Intra-Chip SWAP Reduction: On the homogeneous three-chip system (ST1), DisMap reduces the intra-chip SWAP count for Supremacy(56) from 297 (QT) to 74, a 75.1% reduction. The total cost correspondingly drops from 823 to 90 (89.1% reduction). This improvement follows directly from the optimal inter-link selection: once partition boundaries are aligned with the best links, each subcircuit is transpiled onto physical qubits with low gate error and a short routing path to the link endpoint, minimizing local SWAP overhead in accordance with Eq. (2).

c) Total Cost: DisMap achieves substantial cost reductions across both system types. On the four-chip heterogeneous system (ST4), the cost of Supremacy(72) drops from 1520 (MQT) to 112, a 92.6% reduction. On the homogeneous three-chip system (ST1), the cost of Random(50) decreases from 4126 (QT) to 2330, a 43.5% reduction. Both results are driven by the joint minimization of inter- and intra-chip costs in the hierarchical objective.

3) Compilation Time: Fig 5 also reports compilation time. For structured, shallow circuits such as Adder(30) on ST2, DisMap completes link selection, partitioning, and mapping in 5.73 s, compared to 20.75 s (MQT), 21.83 s (QT), and 18.51 s (UCC). On dense circuits such as Supremacy(64) on ST3, DisMap runs in 1.21 s versus 2.32 s for MQT, while simultaneously producing substantially fewer SWAPs. The compilation time advantage comes from two sources: the hardware-aware link search front-loads topology evaluation once per configuration, and the resulting smaller, better-aligned subcircuits transpile more quickly on each individual chip. Overall, DisMap achieves faster or comparable compilation times while consistently producing higher-quality distributed execution plans.

4) Scalability to Larger Systems: To assess scalability, we evaluate DisMap on two IBM Heron-r2 chips (ibm_marrakesh + ibm_fez, 155 + 155 qubits) using Random circuits of sizes 200 and 300 at depth 10, averaged over 10 runs (Table II). As system and circuit size grow, inter-chip routing complexity and intra-chip SWAP overhead increase

TABLE II: Scalability on large two-chip Heron-r2 systems (ibm_marrakesh + ibm_fez, 155 + 155 qubits). Random circuits at depth = 10, averaged over 10 runs.

Circuit	Compiler	Inter	Intra	Time(s)	Cost
Random 200	MQT	581	4156	122.3	11822
	QT	455	3978	139.4	10216
	UCC	533	4035	127.6	13135
	DisMap	290	3320	95.6	7288
Random 300	MQT	912	6473	158.7	19383
	QT	793	5992	166.8	17678
	UCC	844	6110	162.4	18516
	DisMap	665	4555	117.3	9925

substantially for all baselines; DisMap nonetheless continues to provide significant cost reductions.

For the 200-qubit circuit, DisMap completes compilation in 95.6 s and reduces total cost from 13,135 (UCC) to 7,288, a 44.5% improvement over the strongest baseline. For the 300-qubit circuit, DisMap finishes in 117.3 s and lowers cost from 19,383 (MQT) to 9,925, a 48.8% reduction. These results confirm that the calibration-aware link search consistently identifies high-reliability inter-chip links and low-noise qubit embeddings, yielding substantial reductions in both intra-chip SWAP count and total execution cost as systems scale.

VI. Conclusion

We presented DisMap, a calibration-aware distributed compilation framework for multi-chip quantum systems. Unlike prior DQC compilers that rely on fixed or homogeneous inter-chip link models, DisMap exposes heterogeneous link quality directly to the compiler through a unified hierarchical cost model informed by per-chip calibration data. This design enables joint optimization of link selection, circuit partitioning, and distributed qubit mapping—decisions that prior approaches typically handle in isolation or with simplified communication assumptions.

Evaluated on heterogeneous multi-chip topologies across a diverse benchmark suite, DisMap achieves up to 95.5% fewer inter-chip operations, up to 92.6% lower total execution cost, and up to 21.9% higher circuit fidelity compared to representative baselines. These results suggest that treating inter-chip communication quality as a first-class software concern in the compiler, and performing calibration-aware joint optimization, is a promising direction toward reliable, high-fidelity quantum software for emerging multi-QPU platforms.

The current evaluation relies on emulated inter-chip links parameterized from IBM calibration data, as physical chip-to-chip hardware is not yet broadly accessible. Future work will validate DisMap on real distributed hardware as it becomes available, extend the cost model to entanglement-based long-distance links with dynamic generation rates, and investigate tighter integration with quantum error correction for fault-tolerant distributed execution.

References

[1] M. AbuGhanem, "Ibm quantum computers: evolution, performance, and future directions," *The Journal of Supercomputing*, vol. 81, no. 5, Apr. 2025. [Online]. Available: http://dx.doi.org/10.1007/s11227-025-07047-7

[2] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, p. 79, 2018. [Online]. Available: https://doi.org/10.22331/q-2018-08-06-79

[3] C. Gidney, "How to factor 2048 bit rsa integers with less than a million noisy qubits," 2025. [Online]. Available: https://arxiv.org/abs/2505.15917

[4] P. W. Shor, "Fault-tolerant quantum computation," in *Proceedings of 37th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 1996, pp. 56–65.

[5] N. Ahmed, L. Zhang, and A. Gangopadhyay, "A survey of post-quantum cryptography support in cryptographic libraries," *arXiv preprint arXiv:2508.16078*, 2025.

[6] S. Patel, P. Jayakumar, T.-C. Yen, and A. F. Izmaylov, "Quantum measurement for quantum chemistry on a quantum computer," *arXiv preprint arXiv:2501.14968*, 2025.

[7] Y. Zhang *et al.*, "Quantum algorithms for quantum molecular systems: A survey," *WIREs Computational Molecular Science*, 2025.

[8] M. Caleffi, M. Amoretti, D. Ferrari, D. Cuomo, J. Illiano, A. Manzalini, and A. S. Cacciapuoti, "Distributed quantum computing: A survey," *Computer Networks*, vol. 254, p. 110672, 2024.

[9] D. Barral, F. J. Cardama, G. Díaz-Camacho, D. Faílde, I. F. Llovo, M. Mussa-Juane, J. Vázquez-Pérez, J. Villasuso, C. Piñeiro, N. Costas, J. C. Pichel, T. F. Pena, and A. Gómez, "Review of distributed quantum computing: From single qpu to high performance quantum computing," *Computer Science Review*, vol. 57, p. 100747, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013725000231

[10] "Ibm quantum delivers on 2022 100x100 performance challenge — IBM Quantum Computing Blog — ibm.com," https://www.ibm.com/quantum/blog/qdc-2024, [Accessed 28-02-2025].

[11] J. Xu, X. Deng, W. Zheng, W. Yan, T. Zhang, Z. Zhang, W. Huang, X. Xia, X. Liao, Y. Zhang, J. Zhao, S. Li, X. Tan, D. Lan, and Y. Yu, "Tunable hybrid-mode coupler enabling strong interactions between transmons at centimeter-scale distance," 2025. [Online]. Available: https://arxiv.org/abs/2506.14128

[12] C. Monroe, R. Raussendorf, A. Ruthven, K. R. Brown, P. Maunz, L.-M. Duan, and J. Kim, "Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects," *Physical Review A*, vol. 89, no. 2, p. 022317, 2014.

[13] D. Main *et al.*, "Distributed quantum computing across an optical network link," *Nature*, 2025. [Online]. Available: https://www.nature.com/articles/s41586-024-08404-x

[14] M. Weaver, G. Arnold, H. Weaver, S. Gröblacher, and R. Stockill, "Scalable quantum computing with optical links," *arXiv preprint arXiv:2505.00542*, 2025.

[15] H. Zhao, W. D. Chen, A. Kejriwal, and M. Mirhosseini, "Quantum-enabled microwave-to-optical transduction via silicon nanomechanics," *Nature Nanotechnology*, pp. 1–7, 2025.

[16] M. Akhtar, F. Bonus, F. Lebrun-Gallagher, N. Johnson, M. Siegele-Brown, S. Hong, S. Hile, S. Kulmiya, S. Weidt, and W. Hensinger, "A high-fidelity quantum matter-link between ion-trap microchip modules," *Nature communications*, vol. 14, no. 1, p. 531, 2023.

[17] D. Ferrari, S. Carretta, and M. Amoretti, "A modular quantum compilation framework for distributed quantum computing," *IEEE Transactions on Quantum Engineering*, vol. 4, pp. 1–13, 2023.

[18] P. Promponas, A. Mudvari, L. D. Chiesa, P. Polakos, L. Samuel, and L. Tassiulas, "Compiler for distributed quantum computing: A reinforcement learning approach," in *ICC 2025 - IEEE International Conference on Communications*, 2025, pp. 4615–4621.

[19] D. E. Deutsch, "Quantum computational networks," *Proceedings of the royal society of London. A. mathematical and physical sciences*, vol. 425, no. 1868, pp. 73–90, 1989.

[20] N. P. De Leon, K. M. Itoh, D. Kim, K. K. Mehta, T. E. Northup, H. Paik, B. Palmer, N. Samarth, S. Sangtawesin, and D. W. Steuerman, "Materials challenges and opportunities for quantum computing hardware," *Science*, vol. 372, no. 6539, p. eabb2823, 2021.

[21] https://investors.rigetti.com/static-files/fbac3801-223f-4f0f-a207-47d25084a1d7, [Accessed 20-08-2025].

[22] T. Peng, A. W. Harrow, M. Ozols, and X. Wu, "Simulating large quantum circuits on a small quantum computer," *Physical Review Letters*, vol. 125, p. 150504, 2020.

[23] S. Bravyi, G. Smith, and J. A. Smolin, "Trading classical and quantum computational resources," *Physical Review X*, vol. 6, no. 2, p. 021043, 2016.

[24] M. Y. Siraichi, V. F. d. Santos, C. Collange, and F. M. Q. Pereira, "Qubit allocation," in *Proceedings of the 2018 international symposium on code generation and optimization*, 2018, pp. 113–125.

[25] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, "Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers," in *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019, pp. 1015–1029.

[26] R. Wille, L. Burgholzer, S. Hillmich, and N. Quetschlich, "Mqt qmap: Optimal qubit mapping with noise adaptation," 2023. [Online]. Available: https://github.com/cda-tum/mqt-qmap

[27] "GitHub - unitaryfoundation/ucc: Unitary Compiler Collection — github.com," https://github.com/unitaryfoundation/ucc, [Accessed 15-09-2025].

[28] D. Bhattacharjee and A. Chattopadhyay, "Depth-optimal quantum circuit placement for arbitrary topologies," in *arXiv preprint arXiv:1703.08540*, 2017. [Online]. Available: https://arxiv.org/abs/1703.08540

[29] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for NISQ-era quantum devices," in *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19, 2019, pp. 1001–1014.

[30] H. Deng *et al.*, "CODAR: A contextual duration-aware qubit mapping for various NISQ devices," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020. [Online]. Available: https://arxiv.org/abs/2002.10915

[31] C.-Y. Huang, C.-H. Lien, and W.-K. Mak, "Reinforcement learning and dear framework for solving the qubit mapping problem," in *Proceedings of the 41st IEEE/ACM international conference on computer-aided design*, 2022, pp. 1–9.

[32] S. Sang *et al.*, "Toward scalable quantum compilation for modular architecture: Qubit mapping and reuse via deep reinforcement learning," *arXiv preprint arXiv:2506.09323*, 2025. [Online]. Available: https://arxiv.org/abs/2506.09323

[33] R. V. Meter and S. J. Devitt, "The path to scalable distributed quantum computing," *Computer*, vol. 49, no. 9, pp. 31–42, 2016.

[34] D. Barral, F. J. Cardama, G. Díaz-Camacho, D. Faílde, I. F. Llovo, M. Mussa-Juane, J. Vázquez-Pérez, J. Villasuso, C. Piñeiro, N. Costas *et al.*, "Review of distributed quantum computing: From single qpu to high performance quantum computing," *Computer Science Review*, vol. 57, p. 100747, 2025.

[35] A. Wu, Y. Ding, and A. Li, "AutoComm: A framework for enabling efficient communication in distributed quantum programs," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 1027–1041.

[36] ——, "QuComm: Optimizing collective communication for distributed quantum computing," in *Proceedings of the 56th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023, pp. 479–493.

[37] H. Zhang *et al.*, "MECH: Multi-entry communication highway for superconducting quantum chiplets," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2024. [Online]. Available: https://arxiv.org/abs/2305.05149

[38] ——, "SwitchQNet: Optimizing distributed quantum computing for quantum data centers with switch networks," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA)*, 2025.

[39] P. Promponas, A. Mudvari, L. D. Chiesa, P. Polakos, L. Samuel, and L. Tassiulas, "Compiler for distributed quantum computing: A reinforcement learning approach," *arXiv preprint arXiv:2404.17077*, 2024. [Online]. Available: https://arxiv.org/abs/2404.17077

[40] P. Escofet, A. Gonzalvo, E. Alarcón, C. G. Almudéver, and S. Abadal, "Route-forcing: Scalable quantum circuit mapping for scalable quantum computing architectures," in *IEEE International Conference on Quantum Computing and Engineering (QCE)*, 2024. [Online]. Available: https://arxiv.org/abs/2407.17306

[41] P. Escofet *et al.*, "Revisiting the mapping of quantum circuits," *arXiv preprint arXiv:2403.17205*, 2024. [Online]. Available: https://arxiv.org/abs/2403.17205

[42] R. Zhou *et al.*, "Optimizing compilation for distributed quantum computing via clustering and annealing," *arXiv preprint arXiv:2508.15267*, 2025. [Online]. Available: https://arxiv.org/abs/2508.15267

[43] X. Dou, L. Liu, Z. Wang, and P. Li, "Larqucut: A new cutting and mapping approach for large-sized quantum circuits in distributed quantum computing (dqc) environments," *ACM Transactions on Architecture and Code Optimization*, vol. 22, no. 3, pp. 1–24, 2025.

[44] "Qiskit addon: circuit cutting 0.10.0 — qiskit.github.io," https://qiskit.github.io/qiskit-addon-cutting/index.html, [Accessed 10-09-2025].

[45] G. Aleksandrowicz, T. Alexander, P. Barkoutsos, L. Bello, Y. Ben-Haim, D. Bucher, F. J. Cabrera-Hernández, J. Carballo-Franquis, A. Chen *et al.*, "Qiskit: An open-source framework for quantum computing." *Zenodo*, 2019, qiskit version accessed: 2025-04-13. [Online]. Available: https://zenodo.org/records/2562111

[46] "fake_provider (latest version) — IBM Quantum Documentation — docs.quantum.ibm.com," https://docs.quantum.ibm.com/api/qiskit/providers_fake_provider, [Accessed 10-09-2025].

[47] E. Bernstein and U. Vazirani, "Quantum complexity theory," in *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*. ACM, 1993, pp. 11–20.

[48] W. Methachawalit and P. Chongstitvatana, "Adder circuit on ibm universal quantum computers," in *2020 17th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, 2020, pp. 92–95.

[49] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta, "Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets," *Nature*, vol. 549, no. 7671, pp. 242–246, 2017.

[50] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell *et al.*, "Quantum supremacy using a programmable superconducting processor," *nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[51] "random_circuit (v0.29) — IBM Quantum Documentation — quantum.cloud.ibm.com," https://quantum.cloud.ibm.com/docs/en/api/qiskit/circuit_random, [Accessed 13-09-2025].

[52] P. Escofet, A. Gonzalvo, E. Alarcón, C. G. Almudéver, and S. Abadal, "Route-forcing: Scalable quantum circuit mapping for scalable quantum computing architectures," in *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 01, 2024, pp. 909–920.