STMC: Small-Tile Multiple-Copy Compilation for Reliable Measurement-Based Quantum Computing

Rongchao Dong, Zewei Mo, Yingheng Li, Aditya Pawar, Jun Yang, Youtao Zhang, Xulong Tang
University of Pittsburgh
Pittsburgh, Pennsylvania, USA
{rongchaodong, zewei.mo, yil392, adp110, juy9, youtao, tax6}@pitt.edu

2025 IEEE/ACM International Conference On Computer Aided Design (ICCAD) | 979-8-3315-1560-7/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICCAD66269.2025.11240760

***Abstract*—Measurement-based Quantum Computing (MBQC) achieves universal quantum computing by applying measurements on the photonic architectures. While it has many advantages, such as long qubit decoherence time and strong scalability, the success rate of MBQC execution is constrained by imperfect photon control, measurement, and fusion operations. Both fusion failure and photon loss necessitate the re-execution of the entire quantum circuit, leading to significant overhead in terms of additional execution time and increased consumption of resource state layers. Recent studies mainly focus on mitigating fusion failures and little attention has been paid to photon loss. In this paper, we propose STMC (Small-Tile Multiple-Copy) compilation framework to reduce the re-execution overhead caused by both the fusion failure and photon loss. Specifically, STMC first transforms a quantum circuit into a fusion graph and partitions the fusion graph into subgraphs. Then, STMC generates compact subgraph mappings that are appropriate for the size of a subportion in the resource state layer, referred to as a tile. Finally, STMC employs multiple copies of each subgraph when mapping to tiles, duplicates the execution of tiles in parallel, and finishes the whole circuit execution in order. The experimental results demonstrate that STMC achieves an average execution time speedup of 65.68× for successfully executing the circuit under a 75% fusion success rate, compared to prior work. Additionally, STMC reduces the number of resource state layers by three orders of magnitude and decreases the number of resource states by an average of 36.40×.**

***Index Terms*—Quantum Computing**

I. Introduction

Measurement-based Quantum Computing (MBQC) has emerged as a promising quantum computing paradigm [1], [2] in photonic-based architectures [3]. To accomplish one-qubit and two-qubit quantum gate operations, MBQC performs measurement operations with different bases and angles on the entangled photons. Compared to alternative quantum computing architectures, such as superconducting [4], MBQC demonstrates extended photon decoherence time [5] and high scalability in photon number [2], [6]. Despite many advantages, the current quantum circuit execution of MBQC suffers from a low success rate due to the following two main factors.

- **Imperfect photon quality.** Each photon may be lost independently in many scenarios during the generation, entanglement, transmission, and measurement [7]–[14]. For example, photons could be attenuated and lost during the transmission in optical fiber [9]. If photon loss occurs in the execution of a quantum circuit, the entire circuit execution fails and has to be re-executed from the beginning.
- **Imperfect fusion operation.** Fusion operations are used to construct a large graph state for MBQC execution. However, fusion operation also suffers a low success probability of 75%, as reported in [15]. Fusion failures could also lead to unsuccessful circuit execution that requires re-execution.

Significant efforts have been spent in recent studies to improve MBQC photonic quantum computing [16]–[22]. Among them, two representative compilation techniques that are *OneQ* [16] and *OnePerc* [17]. OneQ first proposes the transformation of quantum circuits into fusion graphs and demonstrates successful circuit execution with optimized circuit depth, assuming no fusion failures. OnePerc leverages the percolation threshold theory [23] to guarantee successful qubit mapping and execution under fusion failures. However, none of them consider photon loss. Consequently, the entire quantum circuit has to be re-executed when photon loss happens, leading to longer execution time and requiring a significant amount of resources.

In this paper, we propose STMC (Small-Tile Multiple-Copy) compilation framework for MBQC. To the best of our knowledge, STMC is the first MBQC compilation framework to reduce the re-execution overhead caused by both fusion failure and photon loss. STMC achieves this reduction by partitioning a fusion graph into subgraphs and executing multiple copies of each subgraph in parallel. To further reduce the re-execution overhead, STMC adopts a compact subgraph mapping strategy. We summarize our contributions as follows:

- We observe that a fusion graph of a quantum circuit can be partitioned into subgraphs and executed in different resource state layers with the delayed photon feature in photonic hardwares. Moreover, redundantly executing multiple copies of a subgraph can significantly reduce the re-execution overhead caused by photon losses and fusion failures.
- It is non-trivial to conduct an effective and efficient subgraph to resource state layers mapping, and a "compact" mapping is in favor of further reducing the re-execution overhead. To this end, we formulate the subgraph mapping problem as an Integer Linear Programming (ILP) model, enabling the generation of compact subgraph mappings.
- We evaluate STMC using four representative quantum applications with different logical qubit counts. The experimental results show that STMC speeds up the average execution time by 65.68×, reduces the number of used resource state layers by three orders of magnitude, and decreases the number of

Authorized licensed use limited to: University of Pittsburgh. Downloaded on May 14,2026 at 05:37:17 UTC from IEEE Xplore. Restrictions apply.

used resource states by an average of 36.40× under 75% fusion success rate.

II. Background and Motivation

A. Photonic Hardware Architecture

Measurement-based Quantum Computing (MBQC) enables universal quantum computing on photonic architectures. To illustrate its key components, Figure 1 depicts the main structure of current photonic hardware architectures. The photonic architecture is composed of multiple resource state layers, which collectively form a 3D resource state graph over time. Each resource state is a small set of entangled photons, and is produced by the resource state generator (RSG) [24]. The set of resource states generated in each epoch is referred to as the resource state layer (RSL). Currently, a physical RSG can faithfully produce a resource state that consists of a few entangled photons, for example, three to six photons [24]. Photons within a resource state in the current RSL can be delayed to one or more subsequent RSLs via a photonic buffer known as a *delay line* [24]. As illustrated by the blue-dashed delay line in Figure 1, photons can be delayed to arrive at the same spatial location in a later RSL.

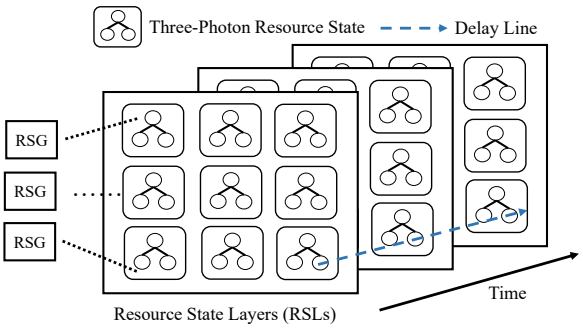


Fig. 1. The structure of the current photonic hardware architectures.

The execution of a quantum circuit on the aforementioned photonic hardware architecture can be achieved by *fusion* operations. Fusion is a two-photon entangling operation used to connect adjacent resource states by performing the interferometric photon measurements on two photons from each resource state with beam splitters and photon detectors [24]. During fusion, two photons are directed onto a beam splitter, where polarization-based interference occurs. Photon detectors placed at the output ports will monitor the interference outcomes. A successful photon fusion event is *heralded* by the detection of exactly one photon at a specific output port. If no photons or multiple photons are detected, the fusion attempt fails. Notably, a failed fusion event typically leads to the deterioration of the original photon, making it unusable for subsequent computational tasks.

As the red circle shown in Figure 2, fusion merges two neighboring resource states to form a larger graph state, enabling the mapping of larger circuits onto RSLs. And as shown by the blue line in Figure 2, a photon can be delayed into the next layer and fuse with other resource states, allowing the computation to continue across several layers. We exploit the hardware properties above in this paper so that we have the flexibility to choose to delay photons in a resource state to the following RSLs or to perform a fusion operation with photons in the other resource states in an RSL.

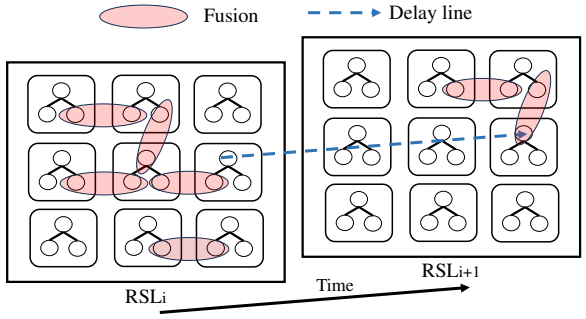


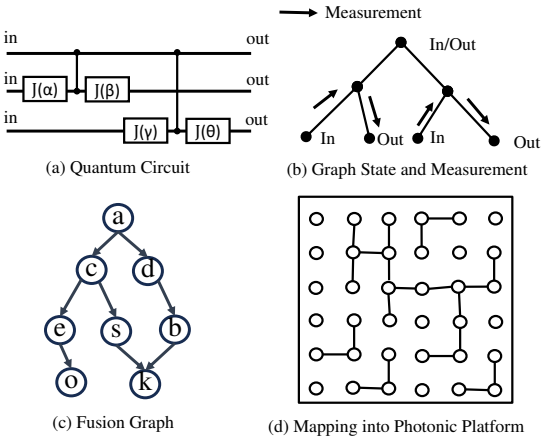
Fig. 2. Photons can fuse with others or delay into the next RSL.

B. Measurement-based Quantum Computing

Given a quantum circuit, MBQC first converts it into a graph state $G = (V, E)$, where each vertex is a photon initialized as the $|+\rangle$ state, and each edge stands for a Controlled-Z (CZ) gate applied to two connected vertices. The graph state can be defined as

$$|G\rangle = \prod_{(a,b)\in E} U^{\{a,b\}} |+\rangle^{\otimes V}$$

where $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and operator $U^{\{a,b\}}$ stands for the CZ gate between photons a and b. Figure 3(a) presents a simple quantum circuit and Figure 3(b) gives its corresponding graph state, where vertices are initialized photons, and the connected edges between them are the CZ gates. The directed edges in Figure 3(b) indicate the measurement order to carry out the quantum circuit correctly. The measurement basis and angle of photons (i.e., α angle on the X-Y plane) are attached to the vertices, so we can also obtain the measurement pattern from the graph state.



(a) Quantum Circuit (b) Graph State and Measurement (c) Fusion Graph (d) Mapping into Photonic Platform

Fig. 3. Graph transformation of a quantum circuit in MBQC.

Figure 3(c) shows an example of a fusion graph, where each vertex represents a resource state and each edge corresponds to a fusion operation. The fusion graph is transformed from graph state, and the transformation is based on three specific patterns introduced in OneQ [16]: degree increment, line extension, and graph connection. After the transformation, the fusion graph becomes a Directed Acyclic Graph (DAG).

Authorized licensed use limited to: University of Pittsburgh. Downloaded on May 14,2026 at 05:37:17 UTC from IEEE Xplore. Restrictions apply.

Finally, the fusion graph will be mapped into the RSLs. Figure 3(d) shows a possible mapping on an RSL, where each circle is a resource state, and lines between circles represent the fusion operations. The mapping process assigns each vertex in the fusion graph to a resource state, while each edge is mapped to a sequence of fusion operations forming a valid path. A single fusion graph may yield multiple valid mapping solutions, each consuming a different number of fusions and resource states, as edge mapping can follow different routing paths with varying fusion and resource state overhead. After mapping, the circuit execution starts, and the measurements, fusions, and delay operations are carried out on the RSLs. The final result is obtained following the completion of the last operation on the RSLs and is sent to the classical processor.

C. Fusion Failure and Photon Loss

In photonic architecture, there are two major errors affecting the successful execution of a quantum circuit: i) fusion failure and ii) photon loss. The former causes unsuccessful graph state generation, whereas the latter leads to quantum information loss even if the graph state is successfully generated. According to prior work, the most practical fusion success rate is 75% [15]. In contrast, photon loss is independent to each photon and can arise in many stages of photon lifetime before the measurement on it, such as photon generation in the RSG [25], buffered in the delay lines [26], and detected by the photon detectors [27], [28]. In MBQC, both fusion failures and photon losses typically necessitate the re-execution of the entire quantum circuit from the beginning [2], [17]. They can significantly impact circuit execution time, as it may require multiple re-executions to successfully complete the circuit.

Prior research efforts have predominantly concentrated on mitigating fusion failures during graph state generation. For instance, the state-of-the-art photonic quantum compiler, OnePerc [17] exploits the percolation-based approach to handling the randomness introduced by fusion failures. It does so by using percolation [23] theory to map a relative small fusion graph to multiple large RSLs. However, OnePerc cannot handle photon loss, as photon loss can only be detected at the time of its measurement during the execution. By that point, the percolated fusion graph has already been established, leaving no opportunity to handle photon loss unless the circuit is re-executed entirely. In this paper, we leverage redundant execution to reduce the re-execution overhead, significantly improving the circuit execution time.

D. Opportunities and Challenges

Opportunity-I: Improving circuit execution success rates via subgraph-based redundant execution. As OnePerc maps the entire quantum circuit onto the RSLs, it incurs significant re-execution overhead once photon loss occurs. To reduce the re-execution overhead, we can partition the RSL into multiple tiles and redundantly execute multiple copies of one circuit on them. This redundancy ensures that photon loss in one copy does not halt the execution of others, thereby reducing the probability that photon losses will simultaneously occur across all copies. Additionally, since these redundant copies are independent, they can be executed in parallel. As the RSL is partitioned into multiple tiles, a single tile may not be able to accommodate the entire fusion graph. Therefore, the fusion graph is also required to be partitioned into multiple subgraphs. By leveraging the delay property of photonic architectures, we are able to partition the fusion graph into subgraphs and delay photons to connect subgraphs to complete the whole circuit execution. Corresponding to the concept of RSLs, subgraph-based redundant execution means that we execute each subgraph's multiple copies within one RSL in parallel, and we need several RSLs to finish the whole circuit based on the number of subgraphs.

Opportunity-II: Reducing photon count and fusion operations via compact subgraph mapping. Since both photon losses and fusion failures are independent events, involving a greater number of them during circuit execution inherently increases the probability of execution failure, leading to re-execution. Thus, our second goal is to minimize the total photon usage and the number of fusion operations by employing more compact mappings of subgraphs onto RSLs. Achieving such compact mappings, however, is non-trivial and presents significant challenges. Firstly, as quantum circuits become more complex with a larger number of qubits, the search space for identifying optimal compact mappings expands exponentially, complicating the mapping process. Secondly, the mapping process must satisfy various constraints, such as fusions only occur between neighboring photons and each resource state has a limiting connection degree (i.e., the maximum number of fusions it can operate).

III. Design

In this section, we present the design of the STMC (Small-Tile Multiple-Copy) compilation framework, which addresses the aforementioned challenges and leverages key opportunities to reduce the re-execution overhead.

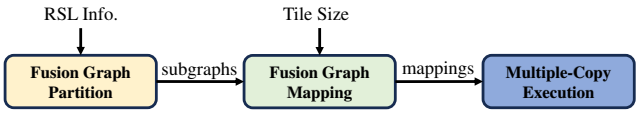


Fig. 4. STMC compilation framework.

As shown in Figure 4, a quantum circuit is first transformed into a fusion graph and partitioned into subgraphs, each of which can be executed within a subset of the RSL, referred to as tiles. To derive mappings for fusion subgraphs, STMC then employs an ILP model to generate compact mappings for each subgraph to a tile. Finally, for each subgraph, STMC performs multiple-copy execution and repeats the failed tiles until the entire circuit is successfully executed.

A. Fusion Graph Partition

As mentioned in Section II-B, we first transform the circuit into a fusion graph using the transformation process proposed in OneQ [16]. As discussed in **Opportunity-I**, we deploy multiple identical subgraph mappings across multiple tiles within a single RSL. Thus, we first need to determine the

Authorized licensed use limited to: University of Pittsburgh. Downloaded on May 14,2026 at 05:37:17 UTC from IEEE Xplore. Restrictions apply.

optimal number of tiles for RSL. In an RSL, increasing the number of tiles leads to smaller tile sizes, requiring the fusion graph to be partitioned into more subgraphs. This increases the number of RSLs due to longer sequential execution. Conversely, using fewer tiles results in larger tile sizes but limits the number of parallel executions. We performed an empirical evaluation to determine the optimal number of tiles, as detailed in Section IV-D1. Once the number of tiles in the RSL is determined, the tile size can also be computed easily.

We then partition the fusion graph into subgraphs based on the calculated tile size. Specifically, we first constrain the number of nodes in each subgraph to be at most half the tile size. This is because, in a DAG, the maximum number of edges is bounded by $n(n-1)/2$ [29], where n is the number of nodes in the DAG. Limiting the node count ensures sufficient space is reserved for edges within the tile. Among the execution of sequential subgraphs, certain resource states (DAG nodes) must be delayed through delay lines into the next RSL to enable the execution of subsequent subgraphs.

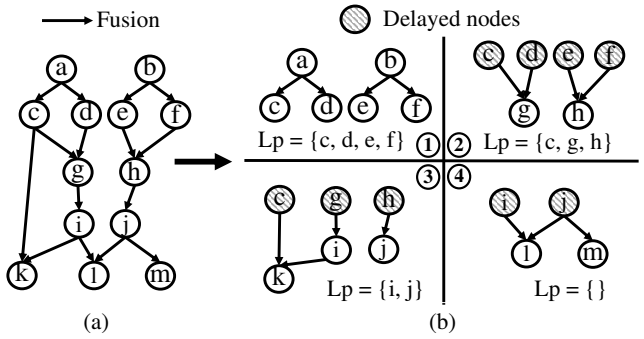


Fig. 5. An example of partitioning a fusion graph into multiple subgraphs.

Figure 5 illustrates an example of partitioning a fusion graph into four subgraphs. Figure 5(a) shows the fusion graph, while Figure 5(b) demonstrates the partitioning process. Assuming a tile size of 12, each subgraph can contain at most 6 nodes. We begin by performing a topological sort to determine the node ordering of the DAG (e.g., the alphabetical order shown in Figure 5(a)), where the photon measurements follow this order [17]. To ensure a node can be executed before its successor nodes, we assign nodes to subgraphs in a greedy manner, starting with the first empty subgraph, labeled ① in Figure 5(b). Since we need to enable the following subgraph's execution, we must identify nodes that need to be delayed into the next subgraph. To this end, we traverse the remaining graph after each partition and maintain a set Lp that records the nodes to be delayed in the subsequent subgraphs. For example, in subgraph ①, $Lp = \{c, d, e, f\}$. These nodes are added to the second subgraph ②, and after the greedy assignment, Lp is updated to $\{c, g, h\}$. This process continues until the entire fusion graph is partitioned into four subgraphs, as shown in Figure 5(b). After the fusion graph partition, we then need to generate the mappings for all subgraphs.

B. Fusion Graph Mapping

Recall **Opportunity-II** in Section II-D, our goal is to generate a compact mapping for each subgraph to minimize both the mapped photon and fusion amount. To this end, we leverage the Integer Linear Programming (ILP) model to solve this problem efficiently.

1) Why ILP: We aim to map the DAG of each subgraph onto a tile such that each DAG node is assigned to a resource state, and each DAG edge is routed through adjacent resource states, subject to a maximum degree constraint M per resource state (i.e., the maximum number of fusions it can perform). To support connectivity beyond this limit, intermediate resource states may be used for routing, which incurs additional edge capacity overhead. The objective is to minimize both the number of resource states and fusions, yielding a compact mapping. A similar mapping problem has been proven to be NP-hard [30]. To address it efficiently, we formulate it as an ILP problem. ILP is well-suited for this task as it enables precise modeling of discrete decisions (e.g., node placement and edge routing) and accommodates complex constraints such as resource state capacity and multi-path routing. Furthermore, since each subgraph and tile size is relatively small, the search space remains computationally tractable.

2) Variables in ILP: Let V denote the set of DAG nodes, E the set of DAG edges, and $G = (P, Q)$ represent a tile with $N \times N$ resource states, where P is the set of resource states positions and Q is the set of edges connecting adjacent resource states. Let M be the maximum degree allowed per resource state. We begin by introducing the binary variables used in the ILP formulation:

- $X_{i,p} \in \{0, 1\}$ for each $i \in V$, $p \in P$: indicates whether DAG node i is mapped to resource states position p. $X_{i,p} = 1$ if node i is placed at p; otherwise, $X_{i,p} = 0$.
- $Y_{e,(u,v)} \in \{0, 1\}$ for each $e \in E$, $(u, v) \in Q$: indicates whether DAG edge e is routed through the fusion (u, v). $Y_{e,(u,v)} = 1$ if edge e uses (u, v) in its routing path; otherwise, $Y_{e,(u,v)} = 0$.
- $D_{i,p} \in \{0, 1\}$ for each $i \in V$, $p \in P$: indicates whether DAG node i on resource state position p delays its remaining photons into the following RSLs. $D_{i,p} = 1$ if node i is delayed; otherwise, $D_{i,p} = 0$.
- $Z_p \in \{0, 1\}$ for each $p \in P$: indicates whether resource state position p is used (i.e., it hosts a node or is involved in routing). $Z_p = 1$ if used; otherwise, $Z_p = 0$.

3) Constraints: To ensure a valid mapping, we impose the following constraints.

During mapping, each DAG node should be placed exactly once. This constraint is enforced by the constraint 1. This guarantees that every node in the DAG is mapped to one and only one resource state position.

$$\sum_{p \in P} X_{i,p} = 1, \quad \forall i \in V \tag{1}$$

Furthermore, each resource state must host at most one DAG node to prevent overlaps. This is enforced by the following constraint 2.

$$\sum_{i \in V} X_{i,p} \leq 1, \quad \forall p \in P \tag{2}$$

Authorized licensed use limited to: University of Pittsburgh. Downloaded on May 14,2026 at 05:37:17 UTC from IEEE Xplore. Restrictions apply.

Since delayed nodes will appear at the exact same positions in the following RSLs as they did in the previous one, we must reserve these positions and prevent any new occupations. This constraint is enforced by Equation 3, where t represents the current RSL and $t-1$ denotes the previous RSL.

$$\sum_{j \in V} X_{j,p}^{(t)} \leq 1 - \sum_{i \in V} D_{i,p}^{(t-1)} \quad \forall p \in P \tag{3}$$

To map each DAG edge onto the tile, we must ensure that the routing forms a valid path from the source DAG node to the destination DAG node. Additionally, the routing path must preserve the measurement order dictated by the DAG. This requirement is enforced by the following constraint 4.

$$\sum_{(p,q) \in Q} y_{e,(p,q)} - \sum_{(q,p) \in Q} y_{e,(q,p)} = \begin{cases} 1 & \text{if } x_{i,p} = 1 \\ -1 & \text{if } x_{j,p} = 1 \\ 0 & \text{otherwise} \end{cases}, \tag{4}$$

$$\forall e = (i \rightarrow j) \in E, \ \forall p \in P \tag{5}$$

To enforce the degree limit on each resource state, we apply the following constraint 6.

$$\sum_{e \in E} \sum_{\substack{(p,q) \in Q \\ \text{or } (q,p) \in Q}} y_{e,(p,q)} \leq M \quad \forall p \in P \tag{6}$$

Finally, to ensure consistency, if any DAG node is placed or any path goes through the resource state position p, then $Z_p = 1$. We have the following constraint 7 and 8 to make sure the correct relations between Z_p, $X_{i,p}$ and $Y_{e,(u,v)}$.

$$z_p \geq x_{i,p} \quad \forall i \in V, \ \forall p \in P \tag{7}$$

$$z_p \geq y_{e,(p,q)} \quad \text{and} \quad z_q \geq y_{e,(p,q)} \quad \forall e \in E, \ \forall (p,q) \in Q \tag{8}$$

The above constraints ensure the correctness of the mapping solution for a single subgraph onto a single tile by enforcing unique node placement, delay node reservation, valid routing paths, and resource state capacity limitation.

4) Objective Function: We define the objective function to minimize the total number of resource states and fusions utilized by the mapping. This promotes compactness in the mapping solution, which in turn reduces the likelihood of photon loss and fusion failure during execution. The objective function, together with the previously defined constraints, forms the complete ILP model.

$$L \equiv \sum_{p \in P} z_p + \sum_{e \in E} \sum_{(u,v) \in Q} y_{e,(u,v)} \tag{9}$$

$$\begin{aligned} &minimize \ \ objective \ \ L(Eq.\ 9) \\ &s.t. \ constraint \ Eqs.\ (1-8) \end{aligned} \tag{10}$$

In the implementation, we use Gurobi [31] as our ILP solver to solve the proposed model. Gurobi efficiently computes near-optimal solutions by leveraging advanced optimization techniques such as branch-and-bound, cutting planes, and pruning. As a result, our mapping problem can be solved efficiently, even with the combinatorial complexity inherent in the constraints.

C. Multiple-Copy Execution

We finally propose the multiple-copy execution process to reduce the re-execution overhead caused by photon loss and fusion failure, as detailed in Algorithm 1. That is, we deploy multiple copies of the same subgraph mapping into one RSL, and enable parallel execution of all copies. Since each tile is executed independently, once a tile succeeds, the required photons are delayed into the following RSLs to allow the execution to proceed with the next subgraph.

Algorithm 1: Multiple-Copy Execution Method

```
Input: Mappings M = {M_1, M_2, ..., M_n}
foreach subgraph M_i ∈ M do
    map to Tiles T;
    foreach Tile T_j in RSL L_k in parallel do
        foreach Fusion operation F_m in T_j do
            if fusion F_m is successful then
                Execute fusion F_m;
            else
                Re-execute on RSL L_{k+1};
        if photon loss occurs within a tile then
            Re-execute on RSL L_{k+1};
        if Tile execution successful then
            if Delay photons successful then
                Start M_{i+1} on RSL L_{k+1};
            else
                Re-execute on RSL L_{k+1};
```

Figure 6 illustrates the execution process for the example fusion graph shown in Figure 5. Figure 6(a) depicts the same partitioned subgraphs derived from the earlier partitioning step, and Figure 6(b) presents a mapping solution produced by the mapping process. On the right side of Figure 6, the temporal execution timeline is shown, demonstrating how the execution progresses step by step as time advances.

Assuming we have four RSLs (i.e., RSL_1 to RSL_4). For each RSL, we assume we can partition it into four tiles (i.e., tile_1 to tile_4). Starting from RSL_1, the first subgraph mapping is duplicated into four copies and mapped to the four tiles in RSL_1. Assuming tile_1 and tile_2 execute successfully without any errors, whereas tile_3 encounters a fusion failure and tile_4 has a photon loss, as the red line and the red circle shown in tile_3 and tile_4. Then in RSL_2, the second subgraph mapping can be scheduled to tile_1 and tile_2, whereas the first subgraph needs to re-execute in tile_3 and tile_4. Similarly, for RSL_3, the third subgraph can be scheduled to tile_1, but the first subgraph has to re-execute in tile_2 due to the photon loss happened in RSL_2. Finally, assuming the tile_1 completes the execution in RSL_4, the entire execution will be completed and stopped at RSL_4. The final result will be retrieved and transferred into classical processors and the remaining unfinished executions in other tiles will also be ceased.

By combining redundancy and parallelism execution, STMC enhances overall execution speedup and mitigates the effects of fusion failures and photon losses during the execution.

Authorized licensed use limited to: University of Pittsburgh. Downloaded on May 14,2026 at 05:37:17 UTC from IEEE Xplore. Restrictions apply.

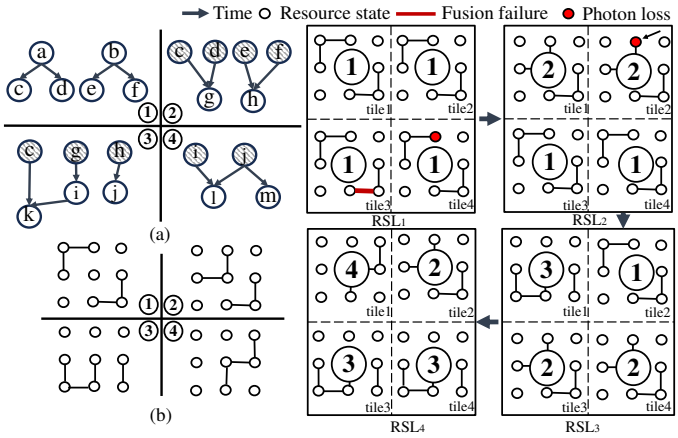


Fig. 6. Multiple-Copy execution of the partitioned subgraphs in Figure 5.

IV. Evaluation

A. Experimental Setup

1) Baseline: We select the state-of-the-art MBQC compiler OnePerc [17] as our baseline. We failed to execute most of the benchmarks using OneQ [16] due to its massive memory requirements. Since OneQ requires several orders of magnitude more fusion operations and RSLs than STMC when simulating under both fusion failure and photon loss conditions.

2) Benchmarks: We use four representative quantum computing benchmarks: Quantum Approximate Optimization Algorithm (QAOA) [32], Hardware Efficient Ansatz (HWEA) [33], Bernstein–Vazirani (BV) [34], and Variational Quantum Eigensolver (VQE) [35]. We use qiskit [36] to construct our benchmarks. Specifically, QAOA is implemented to solve the max-cut problem on randomly generated graphs. VQE is used to estimate the ground state energy of molecular Hamiltonians. HWEA is applied to variational algorithms using shallow circuits tailored to the connectivity of the hardware, commonly used for near-term quantum devices. BV is a generic representative algorithm for evaluating quantum advantage in query complexity. The fusion graph information of each benchmark is given in Table I. For each benchmark, we studied three different numbers of logical qubits: 16, 36, and 64 in our main results. We further conduct experiments on circuits with up to 100 qubits to demonstrate the scalability of our approach.

TABLE I
FUSION GRAPH INFORMATION OF BENCHMARKS.

Benchmark	Qubit	Graph Size	
		#Node	#Edge
BV	16	46	59
HWEA		224	256
QAOA		214	281
VQE		120	163
BV	36	106	139
HWEA		524	592
QAOA		514	676
VQE		280	383
BV	64	190	251
HWEA		944	1068
QAOA		934	1239
VQE		504	691

3) Hardware Setup: Table II lists the photonic architecture settings. We use the same RSL size as employed in OnePerc to ensure a fair comparison between STMC and OnePerc. STMC will partition the RSL and map onto the small tiles, while OnePerc will map onto the whole RSL directly. As shown in Table II, S_t denotes the size of each tile in the RSL and S_c is the number of tiles. We adopt the optimal tile count derived from the empirical evaluation presented in Section IV-D1. M is the overall size of the RSL adopted from OnePerc. P_f represents the fusion success rate, and P_l denotes the photon loss rate [12]. We evaluate our approach under two fusion success rate settings: 75%, which reflects practical experimental conditions as reported in [15], and 90%, which represents an idealized scenario used in [17]. This dual-setting evaluation allows us to assess both real-world robustness and the best-case performance. To further gain a comprehensive understanding of the impact of photon loss, we conduct a sensitivity analysis by varying the photon loss rate P_l, as shown in Section IV-D.

Since photon loss and fusion failure are probabilistic events that can vary across different runs, we execute each benchmark 30 times and report the average results to ensure statistical reliability and robustness of the evaluation.

TABLE II
SUMMARY OF HARDWARE SETUP FOR DIFFERENT QUBIT COUNTS.

#Qubit	S_t	S_c	M	P_f	P_l
16	4×4	576	96×96	75% [15]	0.05% [12]
36	6×6	576	144×144		
64	8×8	576	192×192		
16	4×4	144	48×48	90% [17]	0.05% [12]
36	6×6	144	72×72		
64	8×8	144	96×96		

4) Prototype Implementation: We implement both the STMC and the OnePerc in Python 3.9.12. We simulate the photonic architectures, where delay operations, and measurements are modeled based on predefined loss rates [24] to reflect realistic photonic behavior. Both STMC and OnePerc require repeated execution if photon loss caused by delay or measurement happens during the execution. The evaluation is conducted on a single node with 8 AMD EPYC 7351 CPUs with 256 GB of memory.

5) Metrics: We evaluate STMC and OnePerc using four key metrics: i) **Compilation time** – the fusion graph partitioning time and the ILP mapping time in STMC, corresponding to the mapping time in OnePerc; ii) **Execution time** – the time of the quantum circuit can be successful executed in both methods; iii) **Number of used RSLs** (denoted as #RSL) – indicating how many RSLs are consumed during execution; iv) **Number of used resource states** (denoted as #RS) – representing the total number of resource states used across all RSLs.

B. Performance Analysis

Table III and Table IV compare the compilation time and execution time between STMC and OnePerc. The final column reports the execution time speedup ratio, highlighting the performance improvement of STMC over OnePerc. From Table III and Table IV, several key observations can be made:

Authorized licensed use limited to: University of Pittsburgh. Downloaded on May 14,2026 at 05:37:17 UTC from IEEE Xplore. Restrictions apply.

1) execution time: First, STMC significantly outperforms OnePerc across most of the benchmarks, achieving substantial improvements during the execution. On average, STMC delivers a 65.68× speedup under a 75% fusion success rate and 381.70× speedup under the ideal 90% fusion success rate. This improvement is primarily attributed to the multiple-copy execution strategy, which significantly reduce the re-execution overhead caused by photon loss and fusion failure. Second, we can observe that STMC completes most benchmarks within several seconds, except for QAOA-64 in 75% rate in Table III. This is due to its higher circuit complexity as shown in Table I. Under 90% fusion success rate, QAOA-64 could finish within 6 seconds, demonstrating STMC's scalability under favorable conditions. Third, when the qubit count is small (e.g., 16), OnePerc runs faster than STMC in cases like BV-16 under 75% fusion success rate and all 16-qubit benchmarks under 90% success rate. This is because small quantum circuits produce simpler fusion graphs, and OnePerc could generate the whole circuit mapping on large RSLs and run the mapping quickly, while STMC introduces slight overhead as we partition and execute the fusion subgraphs.

2) compilation time: When comparing compilation time, STMC consistently outperforms OnePerc under 75% fusion success rates, as shown in Table III. This benefit comes from STMC's partitioning strategy, as smaller subgraphs and smaller tiles are much easier to generate mappings. Also, STMC has similar average compilation time under both 75% and 90% fusion rates. Since STMC uses a fixed mapping strategy, it will not be affected by the fusion success rate. In contrast, OnePerc compiles much faster at 90% fusion success rate than it at 75% fusion success rate. This is due to its inherent sensitivity to fusion success rate. As the results show in Table IV, it has a similar average compilation time in 90% fusion success rate to STMC. Furthermore, the compilation time increases with the number of qubits for all benchmarks in both methods. This suggests that more complex circuits (e.g., QAOA and HWEA) require additional time to generate valid mappings. However, due to its subgraph partitioning and mapping approach, STMC exhibits a stable and gradual increase in compilation time, whereas OnePerc shows an exponential growth. Additionally, STMC can further exploit parallelism to reduce compilation time even more.

C. Efficiency Analysis

Tables V and VI present the number of RSLs and resource states used under 75% and 90% fusion success rates, respectively. These results highlight the execution efficiency. As we can see, STMC significantly outperforms OnePerc in both metrics. Under 75% fusion success rate, the number of RSLs is reduced by an average of 6848.12×, and resource states by 36.40×. Under 90% fusion success rate, the reduction is 3219.79× for RSLs and 285.69× for resource states. Furthermore, by comparing two tables, we can observe that both methods benefit from improved fusion rates, requiring fewer resources overall. Finally, STMC uses slightly more or a comparable number of resource states than OnePerc, but fewer RSLs for BV-16, VQE-16, and BV-36 in both fusion rate settings. One reason is that these benchmarks are relatively simple, requiring fewer resource states in the mappings, which allows OnePerc to complete them using fewer resource states overall. But it still consumes a larger number of RSLs due to full re-executions. Another reason is due to our multiple-copy strategy, which increases total resource state consumption per RSL in exchange for higher overall execution reliability.

TABLE III
THE END-TO-END TIME (IN SECONDS) FOR BOTH ONEPERC AND STMC UNDER FUSION SUCCESS RATE OF 75%.

Benchmark	OnePerc		STMC		Execution Speedup
	Compilation	Execution	Compilation	Execution	
BV-16	9.54	**0.0007**	7.26	0.05	0.02
HWEA-16	110.12	0.34	10.91	**0.16**	**2.19**
QAOA-16	150.59	1.53	7.45	**0.19**	**8.15**
VQE-16	42.28	0.24	16.64	**0.11**	**2.23**
BV-32	28.89	0.19	38.43	**0.15**	**1.25**
HWEA-32	496.83	959.30	219.75	**0.62**	**1544.68**
QAOA-32	1129.16	1053.93	335.36	**1.77**	**614.68**
VQE-32	250.50	1040.97	121.87	**0.75**	**1388.15**
BV-64	65.95	27.59	168.75	**0.44**	**63.27**
HWEA-64	2093.41	1831.28	363.99	**7.92**	**231.33**
QAOA-64	5621.65	2132.58	721.37	**113.92**	**18.72**
VQE-64	1208.81	1665.88	306.36	**6.66**	**250.22**
Average	933.42	726.15	193.18	**11.06**	**65.68**

TABLE IV
THE END-TO-END TIME (IN SECONDS) FOR BOTH ONEPERC AND STMC UNDER FUSION SUCCESS RATE OF 90%.

Benchmark	OnePerc		STMC		Execution Speedup
	Compilation	Execution	Compilation	Execution	
BV-16	2.25	$\mathbf{8 \times 10^{-5}}$	7.24	0.01	0.01
HWEA-16	16.73	**0.004**	10.91	0.04	0.09
QAOA-16	36.56	**0.01**	7.45	0.05	0.12
VQE-16	8.80	**0.0006**	16.66	0.04	0.02
BV-32	5.98	**0.001**	38.39	0.02	0.06
HWEA-32	200.36	46.14	219.75	**0.08**	**600.04**
QAOA-32	340.60	301.24	331.36	**0.19**	**1575.97**
VQE-32	55.55	2.24	121.87	**0.10**	**22.86**
BV-64	14.54	0.05	169.32	**0.03**	**1.81**
HWEA-64	358.85	747.22	361.31	**0.19**	**3852.20**
QAOA-64	872.30	737.20	722.56	**5.34**	**138.13**
VQE-64	301.29	306.31	306.36	**0.43**	**1520.19**
Average	184.48	206.95	192.77	**0.54**	**381.70**

TABLE V
THE RESULTS OF THE #RSLS AND #RS METRICS ACROSS ALL BENCHMARKS UNDER FUSION SUCCESS RATE OF 75%.

Benchmark	OnePerc		STMC		Reduction Factor	
	#RSLs	#RS	#RSLs	#RS	#RSLs	#RS
BV-16	584	4993	30	1.2×10^5	**19.47**	0.04
HWEA-16	8.3×10^4	2.2×10^6	126	6.5×10^5	**655.94**	**3.45**
QAOA-16	3.1×10^5	1.0×10^7	144	7.5×10^5	**2125.35**	**13.63**
VQE-16	5182	2.8×10^5	87	4.4×10^5	**59.56**	0.64
BV-36	3594	6.5×10^5	57	6.2×10^5	**63.05**	**1.05**
HWEA-36	$> 9 \times 10^7$	$> 6.8 \times 10^9$	186	2.2×10^6	$\mathbf{4.9 \times 10^5}$	**3078.79**
QAOA-36	$> 9 \times 10^7$	$> 7.5 \times 10^9$	611	3.9×10^6	$\mathbf{1.5 \times 10^5}$	**1931.96**
VQE-36	$> 9 \times 10^7$	$> 7.5 \times 10^9$	270	2.1×10^6	$\mathbf{3.3 \times 10^5}$	**3550.09**
BV-64	9.7×10^5	2.3×10^8	81	1.7×10^6	$\mathbf{1.2 \times 10^4}$	**130.59**
HWEA-64	$> 9 \times 10^7$	$> 1.3 \times 10^{10}$	1146	2.6×10^7	$\mathbf{7.9 \times 10^4}$	**503.56**
QAOA-64	$> 9 \times 10^7$	$> 1.5 \times 10^{10}$	7.6×10^4	1.7×10^9	**1192.91**	**9.31**
VQE-64	$> 9 \times 10^7$	$> 1.2 \times 10^{10}$	921	1.5×10^7	$\mathbf{9.9 \times 10^4}$	**808.30**
Average	4.5×10^7	5.2×10^9	6633	1.4×10^8	**6848.12**	**36.40**

D. Sensitivity Analysis

1) Optimal number of tiles: We first perform the empirical evaluation to determine the optimal number of tiles for each quantum circuit we used, as shown in Figure 7. We evaluate

Authorized licensed use limited to: University of Pittsburgh. Downloaded on May 14,2026 at 05:37:17 UTC from IEEE Xplore. Restrictions apply.

TABLE VI
THE RESULTS OF THE #RSLS AND #RS METRICS ACROSS ALL BENCHMARKS UNDER FUSION SUCCESS RATE OF 90%.

Benchmark	OnePerc		STMC		Reduction Factor	
	#RSLs	#RS	#RSLs	#RS	#RSLs	#RS
BV-16	75	840	30	2.9×10^4	**2.50**	0.03
HWEA-16	12460	4.0×10^5	126	1.7×10^5	**98.89**	**2.34**
QAOA-16	2.2×10^4	2.5×10^5	144	1.9×10^5	**155.48**	**1.33**
VQE-16	1473	34415	87	1.1×10^5	**16.93**	0.31
BV-36	1033	98753	57	1.6×10^5	**18.12**	0.60
HWEA-36	1.0×10^6	2.9×10^8	186	6.0×10^5	**5386.20**	**490.23**
QAOA-36	8.9×10^6	1.9×10^9	441	9.6×10^5	$\mathbf{2.0 \times 10^5}$	**1965.94**
VQE-36	52931	1.7×10^7	270	5.5×10^5	**196.04**	**30.00**
BV-64	2013	2.8×10^5	81	4.3×10^5	**24.85**	0.65
HWEA-64	8.9×10^6	5.2×10^9	531	2.7×10^6	$\mathbf{1.7 \times 10^4}$	**1964.49**
QAOA-64	8.9×10^6	5.1×10^9	8673	4.9×10^7	**1036.74**	**102.89**
VQE-64	8.9×10^6	4.5×10^9	870	4.3×10^6	$\mathbf{1.0 \times 10^4}$	**1044.16**
Average	3.1×10^6	1.4×10^9	958	5.0×10^6	**3219.79**	**285.69**

the average execution time across all selected benchmarks using different numbers of tiles under different fusion success rates. The results indicate that under both fusion success rates (75% and 90%), increasing the number of tiles initially reduces the execution time across all benchmarks, demonstrating that redundant execution effectively lowers the re-execution overhead. We also observe that the optimal number of tiles is 576 for 75% and 144 for 90% fusion rate. Beyond the optimal results, the total execution time begins to increase, because excessive partitioning into too many subgraphs introduces longer execution depth, thereby extending the overall execution time. We use the optimal number of tiles as the default configuration to achieve the best performance results, as shown in Table II.

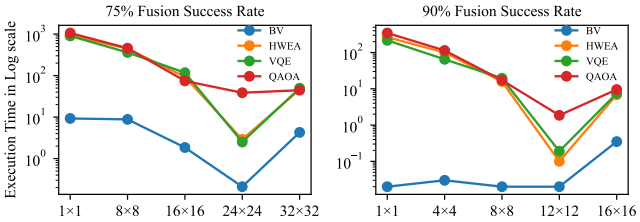


Fig. 7. Execution time comparison across different number of tiles.

2) Large Qubit Benchmarks: To further demonstrate the scalability of STMC, we evaluate the execution time on 100-qubit circuits. As shown in Figure 8, STMC consistently outperforms OnePerc, achieving a 9.43× speedup under a 75% fusion success rate and a 52.80× speedup under a 90% fusion success rate. The results also highlight that STMC can benefit more from ideal fusion success rates, indicating its strong potential for future use as fusion success rate improves.

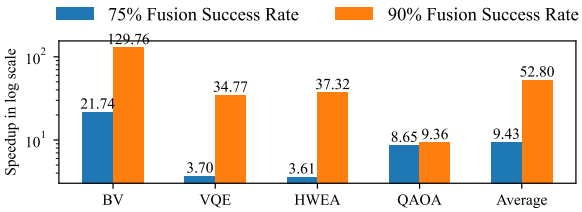


Fig. 8. Execution time speedup of 100-qubit benchmarks.

3) Photon Loss Variation: We also conduct a sensitivity study to evaluate how photon loss rate impacts execution time. We consider three photon loss settings: 0%, representing an ideal scenario with no photon loss, which highlights the efficiency of our partition-based mapping compared to OnePerc's whole-circuit-based approach; 0.1% and 0.5%, which are higher than the loss rates used in our main results, allowing us to assess the robustness of STMC under more challenging conditions. We were unable to evaluate loss rates beyond 0.5% because OnePerc requires more than 24 hours to complete a single benchmark under such conditions, making the evaluation computationally infeasible.

As shown in Figure 9, we run all benchmarks listed in Table I and report the average execution time. When the photon loss rate is 0%, OnePerc outperforms STMC, as it maps the entire circuit directly, whereas STMC's partitioning introduces additional circuit depth. However, under photon loss rates of 0.1% and 0.5%, STMC completes all benchmarks significantly faster than OnePerc. Moreover, as the loss rate increases, STMC's speedup continues to grow, highlighting its strong resilience to photon loss and excellent scalability. The exception is QAOA, where the inherent circuit complexity leads to a longer execution depth in STMC, making the improvement less compared to other benchmarks.

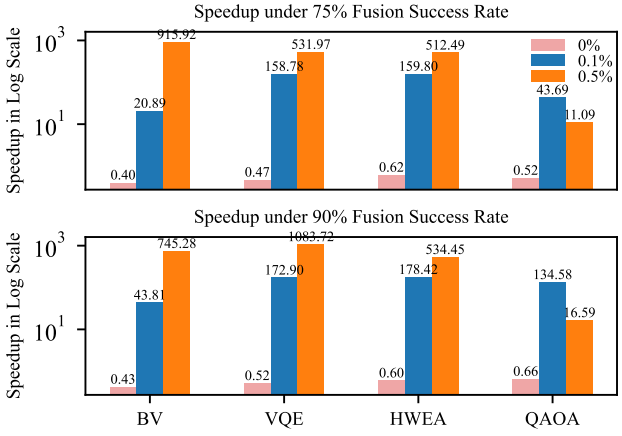


Fig. 9. Average execution time speedup in different photon loss rates.

V. CONCLUSION

In this paper, we propose the Small-Tile Multiple-Copy (STMC) compilation framework for MBQC, which explicitly addresses both fusion failure and photon loss. STMC employs redundant copies and executes compact mapping across multiple tiles in parallel, significantly reducing the re-execution overhead. Experimental results demonstrate that STMC improves the reliability of MBQC execution, achieving an average execution time speedup of 65.68×, while reducing the number of RSLs used by an average of three orders of magnitude and the total number of resource states by an average of 36.40× under a realistic 75% fusion success rate compared to the baseline.

ACKNOWLEDGMENTS

This work is supported by the National Science Foundation under award numbers #2312157, #2154973, and #2334628. We sincerely appreciate all reviewers for their valuable insights and feedback.

Authorized licensed use limited to: University of Pittsburgh. Downloaded on May 14,2026 at 05:37:17 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] H.-S. Zhong, H. Wang, Y.-H. Deng, M.-C. Chen, L.-C. Peng, Y.-H. Luo, J. Qin, D. Wu, X. Ding, Y. Hu *et al.*, "Quantum computational advantage using photons," *Science*, vol. 370, no. 6523, pp. 1460–1463, 2020.

[2] J. L. O'brien, A. Furusawa, and J. Vučković, "Photonic quantum technologies," *Nature Photonics*, vol. 3, no. 12, pp. 687–695, 2009.

[3] L. S. Madsen, F. Laudenbach, M. F. Askarani, F. Rortais, T. Vincent, J. F. Bulmer, F. M. Miatto, L. Neuhaus, L. G. Helt, M. J. Collins *et al.*, "Quantum computational advantage with a programmable photonic processor," *Nature*, vol. 606, no. 7912, pp. 75–81, 2022.

[4] 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.

[5] W. F. Koehl, B. B. Buckley, F. J. Heremans, G. Calusine, and D. D. Awschalom, "Room temperature coherent control of defect spin qubits in silicon carbide," *Nature*, vol. 479, no. 7371, pp. 84–87, 2011.

[6] H. J. Briegel, D. E. Browne, W. Dür, R. Raussendorf, and M. Van den Nest, "Measurement-based quantum computation," *Nature Physics*, vol. 5, no. 1, pp. 19–26, 2009.

[7] S. Scheel, "Single-photon sources–an introduction," *Journal of Modern Optics*, vol. 56, no. 2-3, pp. 141–160, 2009.

[8] K. Ballantine and J. Ruostekoski, "Quantum single-photon control, storage, and entanglement generation with planar atomic arrays," *PRX Quantum*, vol. 2, no. 4, p. 040362, 2021.

[9] T.-J. Wang, C. Cao, and C. Wang, "Linear-optical implementation of hyperdistillation from photon loss," *Phys. Rev. A*, vol. 89, p. 052303, May 2014.

[10] C. M. Dawson, H. L. Haselgrove, and M. A. Nielsen, "Noise thresholds for optical quantum computers," *Phys. Rev. Lett.*, vol. 96, p. 020501, Jan 2006.

[11] R. H. Hadfield, "Single-photon detectors for optical quantum information applications," *Nature photonics*, vol. 3, no. 12, pp. 696–705, 2009.

[12] J. L. O'brien, "Optical quantum computing," *Science*, vol. 318, no. 5856, pp. 1567–1570, 2007.

[13] E. T. Campbell and S. C. Benjamin, "Measurement-based entanglement under conditions of extreme photon loss," *Physical review letters*, vol. 101, no. 13, p. 130502, 2008.

[14] E. Knill, R. Laflamme, and G. J. Milburn, "A scheme for efficient quantum computation with linear optics," *nature*, vol. 409, no. 6816, pp. 46–52, 2001.

[15] W. P. Grice, "Arbitrarily complete bell-state measurement using only linear optical elements," *Physical Review A—Atomic, Molecular, and Optical Physics*, vol. 84, no. 4, p. 042331, 2011.

[16] H. Zhang, A. Wu, Y. Wang, G. Li, H. Shapourian, A. Shabani, and Y. Ding, "Oneq: A compilation framework for photonic one-way quantum computation," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23. New York, NY, USA: Association for Computing Machinery, 2023.

[17] H. Zhang, J. Ruan, H. Shapourian, R. R. Kompella, and Y. Ding, "Oneperc: A randomness-aware compiler for photonic quantum computing," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 738–754.

[18] Y. Li, A. Pawar, M. Azari, Y. Guo, Y. Zhang, J. Yang, K. P. Seshadreesan, and X. Tang, "Orchestrating measurement-based quantum computation over photonic quantum processors," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[19] Y. Li, A. Pawar, Z. Mo, Y. Zhang, J. Yang, and X. Tang, "Fmcc: Flexible measurement-based quantum computation over cluster state," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2025, p. 111–126. [Online]. Available: https://doi.org/10.1145/3622781.3674185

[20] Z. Mo, Y. Li, A. Pawar, X. Tang, J. Yang, and Y. Zhang, "Fcm: A fusion-aware wire cutting approach for measurement-based quantum computing," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, ser. DAC '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3649329.3657346

[21] Y. Li, Y. Dai, A. Pawar, R. Dong, J. Yang, Y. Zhang, and X. Tang, "Using reinforcement learning to guide graph state generation for photonic quantum computers," 2024. [Online]. Available: https://arxiv.org/abs/2412.01038

[22] F. Zilk, K. Staudacher, T. Guggemos, K. Fürlinger, D. Kranzlmüller, and P. Walther, "A compiler for universal photonic quantum computers," in *2022 IEEE/ACM Third International Workshop on Quantum Computing Software (QCS)*. IEEE, 2022, pp. 57–67.

[23] M. Pant, D. Towsley, D. Englund, and S. Guha, "Percolation thresholds for photonic quantum computing," *Nature communications*, vol. 10, no. 1, p. 1070, 2019.

[24] S. Bartolucci, P. Birchall, H. Bombin, H. Cable, C. Dawson, M. Gimeno-Segovia, E. Johnston, K. Kieling, N. Nickerson, M. Pant *et al.*, "Fusion-based quantum computation," *Nature Communications*, vol. 14, no. 1, p. 912, 2023.

[25] Y. Zhang, D. Niu, A. Shabani, and H. Shapourian, "Quantum volume for photonic quantum processors," *Phys. Rev. Lett.*, vol. 130, p. 110602, Mar 2023.

[26] Y. Zhan and S. Sun, "Deterministic generation of loss-tolerant photonic cluster states with a single quantum emitter," *Physical Review Letters*, vol. 125, no. 22, p. 223601, 2020.

[27] T. J. Bell, L. A. Pettersson, and S. Paesani, "Optimizing graph codes for measurement-based loss tolerance," *PRX Quantum*, vol. 4, p. 020328, May 2023.

[28] T. Kilmer and S. Guha, "Boosting linear-optical bell measurement success probability with predetection squeezing and imperfect photon-number-resolving detectors," *Physical Review A*, vol. 99, no. 3, p. 032302, 2019.

[29] K. Thulasiraman and M. N. Swamy, *Graphs: theory and algorithms*. John Wiley & Sons, 2011.

[30] J. Yan, X. Lyu, R. Cheng, and Y. Lin, "Towards machine learning for placement and routing in chip design: a methodological overview," 2022. [Online]. Available: https://arxiv.org/abs/2202.13564

[31] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2024. [Online]. Available: https://www.gurobi.com

[32] E. F. *et al.*, "A quantum approximate optimization algorithm," 2014.

[33] N. Moll, P. Barkoutsos, L. S. Bishop, J. M. Chow, A. Cross, D. J. Egger, S. Filipp, A. Fuhrer, J. M. Gambetta, M. Ganzhorn *et al.*, "Quantum optimization using variational algorithms on near-term quantum devices," *Quantum Science and Technology*, vol. 3, no. 3, p. 030503, 2018.

[34] E. B. *et al.*, "Quantum complexity theory," in *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, 1993, pp. 11–20.

[35] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'brien, "A variational eigenvalue solver on a photonic quantum processor," *Nature communications*, vol. 5, no. 1, p. 4213, 2014.

[36] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson, and J. M. Gambetta, "Quantum computing with Qiskit," 2024.

Authorized licensed use limited to: University of Pittsburgh. Downloaded on May 14,2026 at 05:37:17 UTC from IEEE Xplore. Restrictions apply.