Scalable Circuit Cutting: A Framework for Combined Gate and Wire Cuts Using Gate Groups

Fiona Jiali Fröhler*, Yannick Stade*, Christian Ufrecht, Daniel D. Scherer‡, and Robert Wille*§
*Chair for Design Automation, Technical University of Munich, Munich, Germany
‡Fraunhofer IIS, Fraunhofer Institute for Integrated Circuits IIS, Nuremberg, Germany
§MQSC, Garching near Munich, Germany
{fiona.froehler, yannick.stade, robert.wille}@tum.de, christian.ufrecht@gmx.de, daniel.scherer2@iis.fraunhofer.de
www.cda.cit.tum.de/research/quantum

***Abstract*—Quantum circuit cutting enables the execution of large circuits on devices with a limited number of qubits by partitioning circuits into independent subcircuits. However, this introduces a sampling overhead, which grows exponentially with the number of cuts, rendering the choice of cut placements critical for practical circuit cutting. Determining optimal cut placements remains computationally challenging, particularly as circuits grow in size. Additionally, most existing circuit cutting frameworks treat gate and wire cuts independently. Moreover, approaches combining both cutting, however, do not take advantage of joint cutting, i. e., identifying common gate groups and cutting them jointly to reduce overhead. This work presents a unified framework that, for the first time, combines gate and wire cutting with joint cutting in a scalable manner. To this end, we propose a novel gate-group-aware technique to further reduce sampling overhead. By formulating the cut placement problem as a scalable graph partitioning task, the proposed method efficiently identifies high-quality cut placements for large circuits, also providing diagnostic feedback on whether circuits are suitable for cutting.**

***Index Terms*—quantum computing, quantum circuit cutting, distributed quantum computing**

I. Introduction

Quantum algorithms such as Shor's factoring algorithm [1] and Grover's search [2] offer the promise of significant computational advantages over classical methods. However, realizing these speedups on current quantum hardware remains a significant challenge. Today's quantum processors are constrained by a limited number of qubits, high error rates, and restricted connectivity—characteristics that collectively define the *Noisy Intermediate-Scale Quantum* (NISQ) era [3]. These limitations make it impractical to execute large, complex quantum circuits on a single device.

To overcome this issue, *circuit cutting*—also called *circuit knitting*—has emerged as a practical strategy for decomposing large quantum circuits into smaller subcircuits that can be executed independently with only classical communication channels between devices [4]–[6]. This approach has been experimentally realized on real hardware [7]–[10] and does not require quantum communication channels, unlike *Distributed Quantum Computing* (DQC), making it attractive for near-term systems.

However, circuit cutting comes with a fundamental trade-off: the sampling overhead grows exponentially with the number of cuts, typically limiting practical application to around five to eight cuts [11]. This makes *cut placement*—the process of identifying where to cut a circuit—critical, as high-quality placements directly reduces the required number of cuts.

The two main strategies for cut placement are *gate cuts* and *wire cuts*. Most prior frameworks focus on one strategy in isolation, potentially missing opportunities for more flexible and efficient cut locations. Furthermore, recent work has shown that jointly cutting multiple gates together as a group can substantially reduce the sampling overhead compared to individual cuts [12]—yet this technique is rarely combined with traditional cutting strategies.

This work presents a unified framework that combines gate and wire cutting with joint cutting within a single method. By leveraging all three techniques, the approach achieves better scalability and enables circuit cutting across a broader class of circuits. The framework uses graph partitioning heuristics to efficiently identify promising cut locations, naturally balancing solution quality and computational speed, thereby making circuit cutting more practical and suitable for integration into larger quantum software stacks such as the MQSS [13].

II. Preliminaries

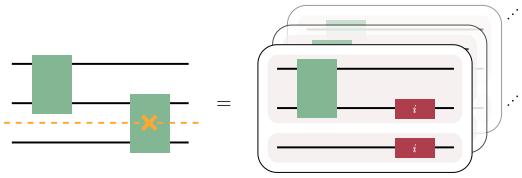
To keep this paper self-contained, this section provides the necessary background for the circuit cutting problem addressed in this work. It reviews the conceptual foundations of quantum circuit cutting and existing approaches to cut placement.

A. Quantum Circuit Cutting

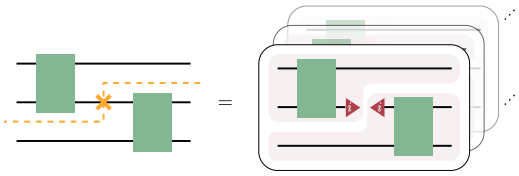
Quantum circuit cutting [4]–[6] enables the execution of large quantum circuits that do not fit as a whole on available hardware with a limited number of qubits. To this end, it partitions circuits into smaller subcircuits, each executable on the available hardware.

Circuit cutting can be realized through two approaches. *Gate cuts* [4], [6], [12], [14]–[18] remove multi-qubit gates and replace them with local gates as shown in Fig. 1a. *Wire cuts* [5], [18]–[22] break the flow of quantum information along a qubit wire as illustrated in Fig. 1b. Both approaches cut the circuit into smaller subcircuits, each satisfying the hardware's qubit count. Formally, this is described by a quasi-probability decomposition of quantum channels [23], [24].

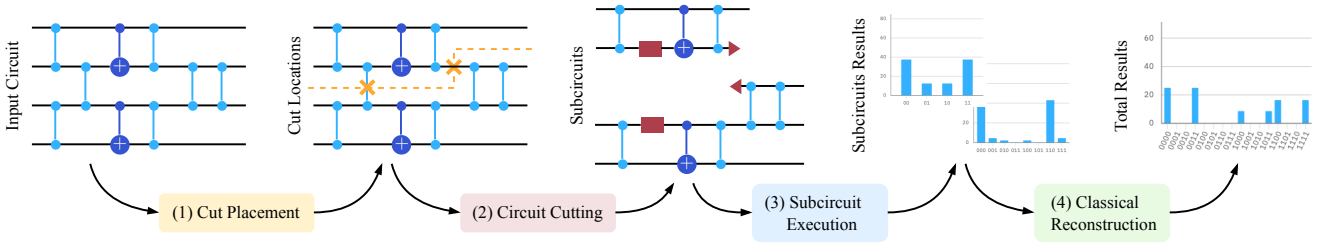
However, this gained flexibility comes at a price: every subcircuit must be executed multiple times to be able to reconstruct the results of the entire circuit from the executions of the subcircuits. This so-called *sampling overhead* scales exponentially with the number of cuts. Specifically, the overhead for estimation of the expectation value of an observable scales as κ^{2n} for n independent cuts, where κ is defined through



(a) Example of gate cutting, where multi-qubit gates are replaced with a classical quasi-probability decomposition over single-qubit gates labelled by index i.



(b) Example of a wire cut, where the flow of quantum information is interrupted along a qubit wire by measuring and preparing the qubit state.



(c) Schematic workflow of a general circuit cutting procedure. This work focuses on the first step—cut placement—highlighted in yellow: identifying a high-quality set of cut locations to partition the circuit.

Figure 1. Different kinds of quantum circuit cuts and the circuit cutting workflow.

the quasi-probability decomposition. For gate cuts, $\kappa_g = 3$ for CNOT gates [4], $\kappa_g = 1 + 2|\sin(\theta)|$ for two-qubit rotation gates [6], [12] and $\kappa_g = 7$ for swap gates [12], [14]. For wire cuts, the optimal overhead is $\kappa_w = 4$ (without classical communication) [5], i. e., for one wire cut, each subcircuit needs $4^2 = 16$ times more samples.

The resulting circuit cutting workflow is depicted in Fig. 1c and consists of four main phases: (1) *cut placement* to determine an optimal cutting strategy that minimizes sampling overhead while ensuring subcircuits fit on available hardware, (2) *circuit cutting* where the individual subcircuits are constructed based on the cut locations and the chosen cutting strategy, (3) *subcircuit execution* where each subcircuit is executed multiple times independently, and (4) *classical reconstruction* to recombine outputs via post-processing.

B. Combined Cutting

The exponential scaling of the sampling overhead limits practicality: typically only five to eight cuts remain feasible before overhead becomes prohibitive [11]. Consequently, minimizing the number of cuts is essential for practical circuit cutting. To achieve this, considering gate and wire cuts simultaneously can yield subcircuits with significantly fewer total cuts, directly reducing sampling overhead.

Example 1. *Fig. 2 depicts three different cut placements for the same circuit, in which single-qubit gates were omitted. The optimal gate cut of the given circuit depicted in Fig. 2a yields $\kappa = 27$, and the optimal wire cut shown in Fig. 2b yields $\kappa = 64$. However, a unified strategy, as illustrated in Fig. 2c, achieves $\kappa = 12$ by leveraging the strengths of both approaches.*

As Exp. 1 demonstrates, the individual strategies alone can be suboptimal depending on the circuit topology. Consequently, a comprehensive strategy should consider both gate and wire cuts simultaneously.

C. Joint Cutting

Beyond considering both cut types, further research explored *joint cutting*, which involves cutting multiple gates together or multiple wires together as a group, rather than independently. This can further reduce sampling overhead [12], [16]–[20], [22]. For gates, this advantage can only be exploited for certain compositions, which we refer to as *gate groups*. These gate groups are illustrated in Fig. 3 and classified by their structure:

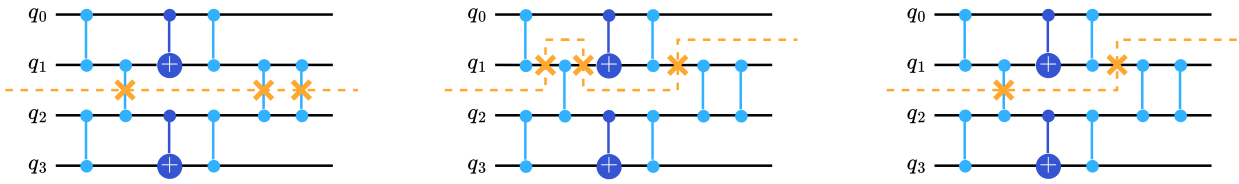
- **Adjacent controls** (see Fig. 3a): Two or more two-qubit gates that share a control qubit and are adjacent along the control line, form a so-called *cascade* on the controls. For CNOT gates, the sampling overhead for cutting the gate group at any position scales with $\kappa = 3$.
- **Adjacent targets** (see Fig. 3b): Similarly, two-qubit gates that share a target qubit and are adjacent along the target, form a *cascade* on the targets, with $\kappa = 3$ for any cut position. However, the gates must be of the same type.
- **Parallel gates** (see Fig. 3c): n_g two-qubit gates act in parallel on disjoint sets of qubits. Here, joint cutting can reduce the overhead below the product of individual cuts, with the overhead for jointly cutting n_g CNOT gates scaling with $\kappa = 2^{n_g+1} - 1$ instead of $\kappa = 3^{n_g}$ [16]–[18].
- **Other groupings** (see Fig. 3d): More complex constellations, such as gates adjacent on the same qubit but not being a cascade, or different gates being adjacent on their target, can also form gate groups with potentially reduced overhead.

Similarly, wires can be jointly cut to reduce sampling overhead below the product of individual wire cuts, however this requires classical communication between subcircuits [19], [20], [22].

D. Existing Approaches

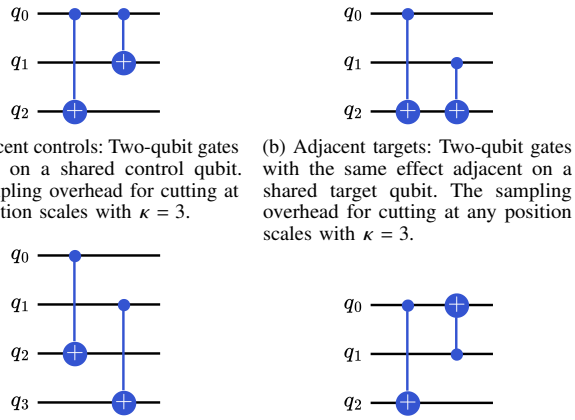
Most existing approaches to cut placement focus on either gate cuts or wire cuts separately. Wire cutting has established optimal methods, such as CutQC [25], as well as more recent heuristics [26]–[28]. Gate cut placement is less studied, but can leverage techniques from Distributed Quantum Computing (DQC) [29]–[34].

However, as demonstrated in Exp. 1, separate consideration of gate and wire cuts prevents finding optimal decompositions for circuits that would benefit from combined strategies. So far, only one method solves the cut placement problem optimally for combined gate and wire cuts [11]. The corresponding



(a) Optimal gate cut, $\kappa = 3 \cdot 3 \cdot 3 = 27$ (b) Optimal wire cut, $\kappa = 4 \cdot 4 \cdot 4 = 64$ (c) Optimal combined cut, $\kappa = 3 \cdot 4 = 12$

Figure 2. Example illustrating the advantage of combined cutting. While pure gate or wire cuts may be optimal within their own categories, a unified strategy can identify mixed solutions that significantly reduce total sampling overhead.



(a) Adjacent controls: Two-qubit gates adjacent on a shared control qubit. The sampling overhead for cutting at any position scales with $\kappa = 3$.

(b) Adjacent targets: Two-qubit gates with the same effect adjacent on a shared target qubit. The sampling overhead for cutting at any position scales with $\kappa = 3$.

(c) Parallel gates: Two-qubit gates acting in parallel on disjoint qubits. Joint cutting can reduce the overhead for cutting n_g gates to $\kappa = 2^{n_g+1} - 1$, e. g., for two gates $\kappa = 7$ instead of $\kappa = 9$.

(d) Other groupings: More complex constellations, e. g. gates adjacent on the same qubit but not a cascade. E. g. jointly cutting both gates here yields $\kappa = 7$, whereas individually cutting them yields $\kappa = 9$.

Figure 3. Examples of different types of gate groups and their associated sampling overheads for various cut placements.

SMT-based approach, while exact, is limited to small circuits (up to 1 hour for 40-qubit circuits with 51 gates).

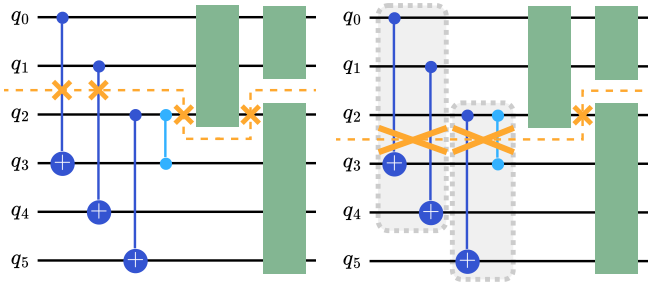
Recent heuristics address this gap with varied strategies. Pawar et al. [35] combine an ILP-based cut finder with qubit reuse, demonstrating improvements over CutQC [25] while facing scalability challenges. The Qiskit circuit cutting module [36] and Nakamura et al. [37] target k-way partitioning instead of bi-partitioning; the latter, though claiming superior performance over Qiskit, may produce vertical cuts that satisfy hardware constraints but incur unnecessary sampling overhead. For DQC, Burt et al. [38] demonstrate strong performance on structured circuits using genetic algorithms with graph partitioning. Other DQC approaches [39]–[41] are slower or too domain-specific for direct application to circuit cutting.

Furthermore, a key limitation across existing approaches concerns gate groups: they only handle cascades, i. e., the cases shown in Fig. 3a and Fig. 3b, missing the efficiency gains achievable through general joint cutting [30], [38].

III. Considered Problem

In this work, we focus on *cut placement*—the first step of a circuit cutting workflow as shown in Fig. 1c. As outlined in the previous section, existing methods typically optimize either gate or wire cuts individually and do not consider gate groups for joint cutting. Although combining gate and wire cutting with joint cutting can significantly reduce sampling overhead compared to individual cuts.

Example 2. *Consider the 6-qubit circuit shown in Fig. 4. In Fig. 4a, applying individual gate and wire cuts yields a sampling overhead of* $\kappa = 3 \cdot 3 \cdot 4 \cdot 4 = 144$. *However, the circuit contains gate groups: two parallel gates on qubits* q_0, q_3 *and* q_1, q_4, *and a cascade on control qubit* q_2. *By cutting these gates jointly rather than individually, the overhead in Fig. 4b is reduced to* $\kappa = (2^{2+1} - 1) \cdot 3 \cdot 4 = 84$. *This corresponds to a 41% reduction compared to individual cuts.*



(a) Optimal combined cut using individual cuts, $\kappa = 3 \cdot 3 \cdot 4 \cdot 4 = 144$ (b) Optimal combined cut considering gate groups, $\kappa = (2^{2+1} - 1) \cdot 3 \cdot 4 = 84$

Figure 4. Example of combined gate, wire, and joint cuts.

The example demonstrates that the existing methods leave huge potential for combining gate and wire cutting with joint cutting untapped. Consequently, we target the following problem statement:

- **Input:** A quantum circuit C consisting of a set of qubits Q and gates, along with possible gate groups.
- **Constraints:** A hardware-imposed limit $q_{\max}$, representing the maximum number of qubits available per subcircuit. The solution must ensure $|Q_i| \leq q_{\max}$ for each subcircuit C_i.
- **Output:** A set of *cut locations* that partition the circuit into independent subcircuits respecting the qubit constraint. Cuts can be individual gate cuts, wire cuts, or joint cuts of groups of gates.
- **Objective:** Minimizing *sampling overhead* $S = \kappa_g^{2n_g} \cdot \kappa_w^{2n_w} \cdot S_{group}$, where n_g and n_w are the numbers of gate and wire cuts, κ_g and κ_w result from the chosen cutting strategy, and S_{group} is the overhead from joint cutting.

This work uses minimal κ-values that do not require classical communication between subcircuits.

Determining the optimal solution to the problem above can be formulated as a balanced graph partitioning problem, which is known to be NP-hard [42]. This NP-hardness creates a fundamental barrier: exact methods cannot scale to practical qubit counts. Aside from scalability, another critical gap remains—none of the existing methods for circuit cutting address this problem within a unified framework.

IV. Proposed Solution

This work addresses this deficiency by proposing a novel scalable approach that combines gate and wire cuts and considers gate groups for joint cutting, allowing circuits with hundreds of qubits to benefit from synergies between these techniques. To this end, we propose a scalable heuristic-based algorithm grounded in graph partitioning techniques that efficiently approximates the optimal solution while scaling to practical circuit sizes.

It follows a *multilevel partitioning strategy* that progressively refines the cut placement. The first level identifies a good gate cut (ideally close to optimal) by only focusing on the circuit's spatial structure. The second level then refines this solution by replacing gate cuts with wire cuts, where this reduces the overall sampling overhead. Furthermore, both steps are extended to further reduce overhead by identifying gate groups and cutting them jointly rather than individually. Each stage is designed to maintain or improve upon prior solutions, with detailed algorithms presented later in Sec. V.

The first stage considers only gate cuts, simplifying the initial problem. We construct an *interaction graph* (see Fig. 5) where each qubit is a node and edges represent two-qubit operations between qubits. The weight of each edge reflects how frequently two qubits interact. By finding a balanced partition of this graph, we identify which interactions are expensive to cut and should be kept within the same partition, and which can be sacrificed.

Input: A quantum circuit with single- or two-qubit gates
Output: A partition of qubits into two groups, and the number of gates cut by this partition

Example 3. *Consider the interaction graph in Fig. 5 of the 4-qubit circuit shown in Fig. 2. The interaction graph formalizes which qubits communicate and how intensely, revealing a chain of equally strong connections (q_0–q_1–q_2–q_3). By inspecting the graph, we identify the edge between q_1 and q_2 as an ideal cutting location for a balanced cut. Partitioning as $\{q_0, q_1\}$ vs. $\{q_2, q_3\}$ cuts only one edge (weight 3), minimizing the gates affected and thus the sampling overhead.*

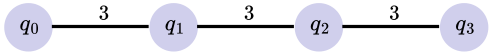


Figure 5. Mapping circuit in Fig. 2 to an interaction graph to find gate cuts.

The second stage refines the partition by considering both gate cuts and wire cuts simultaneously. After identifying good gate cut locations, we construct a *circuit topology graph* (see Fig. 6) that exposes the temporal structure of the circuit. As depicted in Fig. 6, the circuit topology graph not only shows the number of interactions between the qubits but also the time and order of the interactions. This allows us to selectively cut some qubit wires (in addition to or instead of certain gates) if doing so reduces overall overhead. The key challenge is ensuring that cuts remain valid for the given qubit bounds.

Input: Quantum circuit and initial partition from the first stage
Output: Refined partition using both gate and wire cuts

Example 4. *Consider the circuit topology graph in Fig. 6 of the 4-qubit circuit shown in Fig. 2. Then, starting with the gate cut identified in the first stage, we can explore beneficial wire cuts, e. g., cutting a wire instead of two gates. This stage should ideally result in the combined cut shown in Fig. 2c that achieves the lowest sampling overhead.*

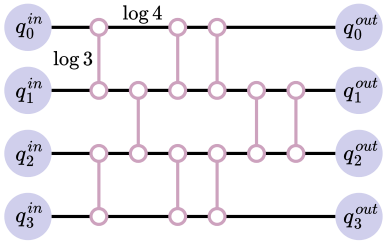


Figure 6. Mapping circuit in Fig. 2 to a circuit topology graph to find gate and wire cuts.

This two-level approach guarantees that we never perform worse than pure gate cutting alone, while capturing synergies between gate and wire cuts.

While the presented approach already constitutes a valid solution to the problem, some resulting cuts can be suboptimal. To this end, we utilize gate groups. By cutting gate groups—rather than individual gates—the sampling overhead can be further reduced. Gate groups are incorporated as partition-dependent cost functions that compute sampling overhead based on the qubit partition, rather than using fixed costs, which would be the case if they were modeled with hyperedges.

Gate groups are incorporated by augmenting the graph with partition-dependent cost functions that accurately reflect the overhead reduction when cutting gates jointly rather than individually. This allows both stages to identify cuts that exploit these grouping opportunities.

Example 5. *Revisiting the 6-qubit circuit in Fig. 4, the circuit topology graph alone identifies cuts that yield sampling overhead $\kappa = 144$ (as shown in the problem section). By recognizing the parallel gates and cascade structure as exploitable groups, the algorithm can identify the same cuts but account for the efficiency gained by joint cutting, yielding the improved solution with overhead $\kappa = 84$ referenced in Fig. 4b.*

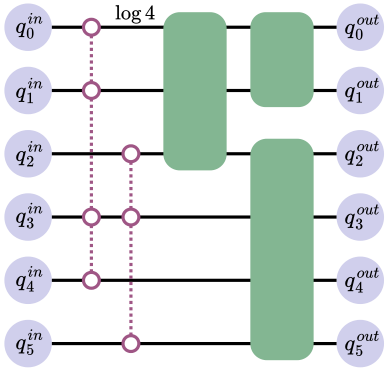


Figure 7. Mapping circuit in Fig. 4 to a circuit topology graph to find gate and wire cuts while considering gate groups.

V. Implementation Details

This section describes the implementation of the multi-level circuit cutting approach resulting from the ideas sketched above.

A. Gate Cutting via Interaction Graph

The first stage performs spatial partitioning via gate cutting by mapping the quantum circuit to an interaction graph and then applying balanced graph partitioning heuristics.

We construct an interaction graph $G = (V, E)$ as follows:

- **Vertices** (V): For each qubit q_i, create a node
- **Edges** (E): For each two-qubit gate operating on qubits q_i and q_j, add or increment the edge (q_i, q_j) in the graph
- **Edge weights**: The weight $w(i, j)$ equals the total number of two-qubit gates between qubits i and j
- **Exclusions**: Single-qubit gates are ignored; multi-qubit gates are decomposed into two-qubit gates

We partition the interaction graph using the Kernighan-Lin (KL) heuristic [43], implemented in NetworkX [44]. KL iteratively swaps pairs of vertices between partitions to minimize the total edge cut cost while maintaining balance of vertices. Since KL is sensitive to initialization, we run the algorithm multiple times with different random initial partitions and select the partition with the minimum edge cut cost.

B. Combined Cutting via Circuit Topology Graph

To incorporate temporal information about gate order, we augment the interaction graph with a circuit topology graph $G = (V_Q \cup V_G, E_W \cup E_G)$ that represents the time dimension:

- **Qubit vertices** (V_Q): For each qubit, create two vertices: q_{in} (input) and q_{out} (output)
- **Gate vertices** (V_G): For each two-qubit gate, create two gate vertices
- **Gate edges** (E_G): Connect the two vertices of each gate
- **Wire edges** (E_W): For each qubit, connect temporally consecutive vertices with an edge, representing the qubit's evolution through the circuit

To apply KL to the circuit topology graph, we assign weights and vertex sizes aligning with the underlying circuit constraints. Edge weights use logarithmic scaling to ensure linear optimization corresponds to exponential overhead reduction, i. e.,

$$w_g = \log \kappa_g, \quad w_w = \log \kappa_w \tag{1}$$

where $\kappa_g \in \{3, 1 + 2|\sin(\theta)|, 7\}$ (see Sec. II) and $\kappa_w = 4$ are the minimal sampling overheads for cutting a single two-qubit gate and wire, respectively.

We assign vertex sizes to enforce the balance constraint only on qubits: gate vertices have size 0, while qubit vertices have size 1. This reflects the physical constraint that device qubit counts are fixed, whereas the number of gates is irrelevant to the partition. We adapted the NetworkX implementation of KL to respect these vertex sizes during partitioning.

Direct partitioning of the circuit topology graph would be problematic: the graph structure is more permissive than the circuit structure. It allows graph-valid partitions that violate qubit constraints (e. g., vertical cuts separating all wires at a single time step). The two-stage strategy addresses this by initializing KL with the spatial partition from the first stage, which prevents such invalid configurations, then refining it by exploring wire cuts while staying within qubit bounds. This guarantees that the final partition respects qubit constraints, never performs worse than the first stage alone, and captures synergies between gate and wire cuts.

C. Extension: Gate Groups

In the partitioning framework, we represent gate groups (detailed in Sec. II) as partition-dependent cost functions. For a given partition of qubits, the function returns the actual sampling overhead incurred by cutting that group, which may vary depending on which qubits are separated.

To incorporate gate groups into the partitioning process, we modify the KL algorithm in two ways. First, when constructing the circuit topology or interaction graph, we remove all edges corresponding to gates within a gate group. This prevents double-counting their cost, since their contribution is instead handled through the gate group cost function during the KL gain calculations.

Second, we adjust the so-called D-values and swap gains within the KL gain calculations to account for gate groups. The D-value for a vertex v normally represents the change in cost if v moves to the opposite partition. If v participates in a gate group, we extend the D-value calculation to:

$$D[v] = \text{ExtCost}(v) - \text{IntCost}(v) + (c_{\text{current}} - c_{\text{moved}}), \tag{2}$$

where c_{current} is the current cut cost of the gate group and c_{moved} is the cost if v were moved. This ensures that changes in the gate group's cost are reflected in the gain calculation.

Similarly, when evaluating a swap of two vertices a and b, we adjust the swap gain to:

$$\delta = D[a] + D[b] - 2w(a, b) + (c_{\text{swapped}} - c_{\text{current}}), \tag{3}$$

where c_{swapped} is the gate group's cost if a and b are swapped. This accounts for any changes in the joint cutting cost resulting from the combined move.

Partition-dependent gate group modeling provides more accurate cost estimates than hyperedge-based approaches, which assign fixed costs regardless of partition. Since gate group costs can vary significantly depending on which qubits are separated, our approach captures these savings and enables better partitioning decisions, particularly in circuits with diverse gate group constellations.

VI. Evaluation

The proposed method is the first to combine gate and wire cutting with joint cutting in a unified framework. To demonstrate the effectiveness of the proposed method, we compared it with the state of the art in circuit cutting across various benchmarks. The following sections describe the experimental setup and eventually present the obtained results.

A. Experimental Setup and Metrics

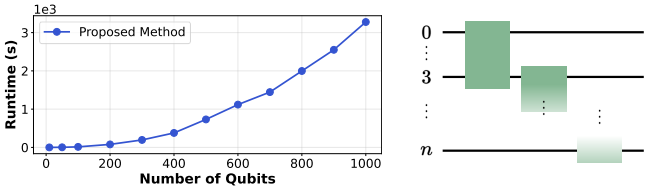
The proposed method has been implemented in Python 3.13 using NetworkX [44]. Evaluation employs the MQT Bench suite [46] (QAOA instances for max-cut problems, QFT, and Grover circuits), plus synthetically generated benchmarks: chain-structured circuits (Fig. 8b) and circuits from the problem formulation (Fig. 4). These circuits are restricted to single- or two-qubit gates and listed in Table I together with their qubit number and two-qubit gate count. For each test scenario, we collect the runtime of the cutting method and the resulting sampling overhead. Partitions are constrained to be at most 3/4 of the original qubit count, and we perform multiple runs (50 per instance) to achieve high-quality results. Scalability is tested on an AMD Ryzen Threadripper PRO 5955WX (16 cores, 130GB RAM); tool comparisons use a MacBook Pro with Apple M5 (10 cores, 24GB RAM).

Table I. COMPARISON OF STATE-OF-THE-ART ALGORITHMS VS. PROPOSED WORK WITH AND WITHOUT GATE GROUPS (GG)

Benchmark			**MIP Gate [45]**		**CutQC Wire [25]**		**Qiskit [36]**		**Prop. Work w/o GG**		**Prop. Work w/ GG**	
Name	Num. Qubits	Num. 2Q-Gates	Runtime (ms)	Sampling Overhead	Runtime (ms)	Sampling Overhead	Runtime (ms)	Sampling Overhead	Runtime (ms)	Sampling Overhead	Runtime (ms)	Sampling Overhead
QAOA	8	16	1	1.7×10^{3}	1488	6.5×10^{4}	150	1.7×10^{3}	123	1.7×10^{3}	260	2.6×10^{2}
	15	30	2	1.7×10^{3}	1581	6.5×10^{4}	1147	1.7×10^{3}	200	1.7×10^{3}	303	2.6×10^{2}
QFT	6	18	5	2.5×10^{3}	1584	NA	149	2.5×10^{3}	99	2.5×10^{3}	428	4.2×10^{1}
	12	72	14	8.0×10^{5}	49088	NA	2365	1.5×10^{7}	397	8.0×10^{5}	[Future Work]	
Grover	4	38	1	8.2×10^{6}	1595	NA	510	1.7×10^{19}	159	8.2×10^{6}	770	5.1×10^{4}
Chain (Fig. 8b)	10	36	3	5.3×10^{5}	1506	1.6×10^{1}	12	1.6×10^{1}	235	1.6×10^{1}	257	1.6×10^{1}
	20	73	4	5.3×10^{5}	1569	1.6×10^{1}	23	1.6×10^{1}	429	1.6×10^{1}	561	1.6×10^{1}
Fig. 4	6	28	2	5.3×10^{5}	1579	NA	248	2.1×10^{4}	153	2.1×10^{4}	212	7.1×10^{3}
Median			**2.5**	**5.3×10^{5}**	**1580**	**2.5×10^{4}**	**199**	**2.1×10^{3}**	**179.5**	**2.1×10^{3}**	**303**	**2.6×10^{2}**

B. Scalability

To demonstrate scalability, we evaluated the proposed method on random circuits ranging from 10 to 1000 qubits with moderate gate density. The method runs 50 times per instance, retaining the best solution. The obtained runtimes, plotted in Fig. 8a, scale moderately, making the proposed method suitable for assessing feasibility for circuit cutting across arbitrary problem instances.



(a) Scalability tests for random circuits. (b) *Chain Circuit* with n qubits.

Figure 8. Scalability results and generated benchmark circuit.

C. Comparison to Other Tools

The proposed method was evaluated against three baselines: an MIP optimal gate-cutting approach using Gurobi [45], CutQC [25] for optimal wire cutting, and Qiskit's circuit cutting implementation [36] for both cut types. The results are summarized in Table I, with the middle columns showing the state-of-the-art baselines, and the rightmost showing the results of the proposed method with and without gate groups.

By combining gate and wire cuts, the approach minimizes or reduces overhead compared to either cut type alone. This advantage is exemplified on the circuit in Fig. 4, where the method significantly reduces sampling overhead relative to the MIP baseline, underscoring the practical benefits of combined optimization. Furthermore, while CutQC often fails to find valid solutions that satisfy the maximum qubit count, the proposed method consistently identifies viable cuts, demonstrating robustness across diverse circuit types. Note that Qiskit performs k-way partitioning while the proposed method is restricted to bi-partitioning; appropriate qubit constraints ensure Qiskit returns only bi-partitions for fair comparison. The proposed method occasionally incurs higher runtimes than Qiskit on some benchmarks. Despite this trade-off, sampling overhead is comparable or superior across benchmarks, particularly for dense circuits such as QFT and Grover.

The last two columns in Table I show the results when considering Gate Groups (GG). The resulting runtime overhead is moderate and could be mitigated through caching strategies. Automatic gate-group identification remains an open problem, with existing approaches limited to canonical forms or greedy heuristics [30], [38]. Hence, group identification currently is manual rather than automatic, limiting scalability for larger circuits.

Overall, the results obtained when considering gate groups demonstrate substantial performance improvements, especially on circuits with many parallel gates. In particular, chain circuits achieve optimal sampling overhead that matches wire-cutting baselines, while dense circuits like QFT achieve an over 90% reduction in overhead. These results underscore the substantial potential of gate-group-enhanced circuit cutting and motivate future work on automation.

VII. CONCLUSION

This work presents a unified, scalable approach for circuit cutting that simultaneously optimizes gate and wire cuts, while considering joint gate cutting—addressing a critical gap in existing methods, which often handle cutting strategies in isolation and do not exploit joint cuts. By leveraging graph partitioning heuristics in a two-stage strategy, the proposed approach never performs worse than gate cutting alone while achieving overhead reduction in many cases.

The method is robust, providing valid solutions across different circuit types, and scales to 1000 qubits efficiently. Additionally, it provides diagnostic value: persistently high cut values reliably signal that a circuit is fundamentally unsuitable for cutting, enabling early go/no-go decisions in compilation pipelines. Furthermore, gate group modeling for joint cutting significantly reduces overhead compared to prior approaches. Together, these improvements make circuit cutting more practical for near-term quantum systems.

Acknowledgments

We thank S. Volkamer for valuable feedback on the manuscript. The authors acknowledge funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program under grant agreement No. 101001318, as well as the Munich Quantum Valley (MQV), which is supported by the Bavarian state government with funds from the Hightech Agenda Bayern Plus. Additionally, this work was supported by the BMFTR under grant number 13N17298 (SYNQ) and by BMIMI, BMWET, the State of Upper Austria and the State of Tyrol within the COMET module Quantum Algorithm Engineering (FFG grant no. 923923) managed by the Austrian Research Promotion Agency FFG.

During the preparation of the code and manuscript, the authors used GitHub Copilot, powered by Claude's Haiku 4.5 to improve code, spelling, grammar, clarity, and readability. Afterward, the authors reviewed and edited the content as needed. The authors take full responsibility for the final content.

References

[1] P.W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Symp. Found. of Comput. Sci.*, 1994.

[2] Lov K. Grover, "A fast quantum mechanical algorithm for database search," in *Symp. Theory of Comput.*, ACM, 1996.

[3] John Preskill, "Quantum Computing in the NISQ era and beyond," *Quantum*, 2018.

[4] Holger F. Hofmann, "How to simulate a quantum computer using negative probabilities," *J. Phys. A: Math. Theor.*, 2009.

[5] Tianyi Peng, Aram Harrow, Maris Ozols, and Xiaodi Wu, "Simulating Large Quantum Circuits on a Small Quantum Computer," *Phys. Rev. Lett.*, 2020.

[6] Kosuke Mitarai and Keisuke Fujii, "Constructing a virtual two-qubit gate by sampling single-qubit operations," *New J. Phys.*, 2021.

[7] Chong Ying et al., "Experimental Simulation of Larger Quantum Circuits with Fewer Superconducting Qubits," *Phys. Rev. Lett.*, 2023.

[8] Akhil Pratap Singh et al., "Experimental demonstration of a high-fidelity virtual two-qubit gate," *Phys. Rev. Res.*, 2024.

[9] Almudena Carrera Vazquez et al., "Combining quantum processors with real-time classical communication," *Nature*, 2024.

[10] Laura S. Herzog et al., *Improving Quantum and Classical Decomposition Methods for Vehicle Routing*, arXiv, 2024.

[11] Sebastian Brandhofer, Ilia Polian, and Kevin Krsulich, "Optimal Partitioning of Quantum Circuits Using Gate Cuts and Wire Cuts," *IEEE Trans. Quantum Eng.*, 2024.

[12] Christophe Piveteau and David Sutter, "Circuit Knitting With Classical Communication," *IEEE Trans. Inf. Theory*, 2024.

[13] Lukas Burgholzer et al., "The munich quantum software stack: Connecting end users, integrating diverse quantum technologies, accelerating hpc," in *Proceedings of the Supercomputing Asia and International Conference on High Performance Computing in Asia Pacific Region*, ACM, 2026.

[14] Kosuke Mitarai and Keisuke Fujii, "Overhead for simulating a non-local channel with local channels by quasiprobability sampling," *Quantum*, 2021.

[15] Christian Ufrecht et al., "Cutting multi-control quantum gates with ZX calculus," *Quantum*, 2023.

[16] Christian Ufrecht et al., "Optimal joint cutting of two-qubit rotation gates," *Phys. Rev. A*, 2024.

[17] Lukas Schmitt, Christophe Piveteau, and David Sutter, "Cutting circuits with multiple two-qubit unitaries," *Quantum*, 2025.

[18] Aram W. Harrow and Angus Lowe, "Optimal Quantum Circuit Cuts with Application to Clustered Hamiltonian Simulation," *PRX Quantum*, 2025.

[19] Lukas Brenner, Christophe Piveteau, and David Sutter, *Optimal wire cutting with classical communication*, arXiv, 2023.

[20] Hiroyuki Harada, Kaito Wada, and Naoki Yamamoto, "Doubly Optimal Parallel Wire Cutting without Ancilla Qubits," *PRX Quantum*, 2024.

[21] Edwin Pednault, *An alternative approach to optimal wire cutting without ancilla qubits*, arXiv, 2023.

[22] Angus Lowe et al., "Fast quantum circuit cutting with randomized measurements," *Quantum*, 2023.

[23] Hakop Pashayan, Joel J. Wallman, and Stephen D. Bartlett, "Estimating Outcome Probabilities of Quantum Circuits Using Quasiprobabilities," *Phys. Rev. Lett.*, 2015.

[24] Christophe Piveteau, David Sutter, and Stefan Woerner, "Quasiprobability decompositions with reduced sampling overhead," *npj Quantum Inf.*, 2022.

[25] Wei Tang, Teague Tomesh, Martin Suchara, Jeffrey Larson, and Margaret Martonosi, "Cutqc: Using small quantum computers for large quantum circuit evaluations," in *Int'l Conf. Archit. Support for Program. Lang. & Oper. Syst.*, ACM, 2021.

[26] Wei Tang and Margaret Martonosi, *ScaleQC: A Scalable Framework for Hybrid Computation on Quantum and Classical Processors*, arXiv, 2022.

[27] Shuwen Kan et al., "Scalable Circuit Cutting and Scheduling in a Resource-Constrained and Distributed Quantum System," in *Int'l Conf. Quantum Comput. & Eng.*, 2024.

[28] Saikat Basu, Arnav Das, Amit Saha, Amlan Chakrabarti, and Susmita Sur-Kolay, "FragQC: An efficient quantum error reduction technique using quantum circuit fragmentation," *J. Syst. Softw.*, 2024.

[29] Mariam Zomorodi-Moghadam, Mahboobeh Houshmand, and Monireh Houshmand, "Optimizing Teleportation Cost in Distributed Quantum Circuits," *Int'l J. Theor. Phys.*, 2018.

[30] Pablo Andrés-Martínez and Chris Heunen, "Automated distribution of quantum circuits via hypergraph partitioning," *Phys. Rev. A*, 2019.

[31] Omid Daei, Keivan Navi, and Mariam Zomorodi-Moghadam, "Optimized Quantum Circuit Partitioning," *Int'l J. Theor. Phys.*, 2020.

[32] Zohreh Davarzani, Mariam Zomorodi-Moghadam, Mahboobeh Houshmand, and Mostafa Nouri-baygi, "A dynamic programming approach for distributing quantum circuits by bipartite graphs," *Quantum Inf. Process.*, 2020.

[33] Zilu Chen, Zhijin Guan, Shuxian Zhao, and Xueyun Cheng, "Distributed quantum circuit partitioning and optimization based on combined spectral clustering and search tree strategies," *Chin. Phys. B*, 2025.

[34] Waldemir Cambiucci, Regina Melo Silveira, and Wilson Vicente Ruggiero, *Spatial and temporal circuit cutting with hypergraphic partitioning*, arXiv, 2025.

[35] Aditya Pawar et al., "QRCC: Evaluating Large Quantum Circuits on Small Quantum Computers through Integrated Qubit Reuse and Circuit Cutting," in *Int'l Conf. Archit. Support for Program. Lang. & Oper. Syst.*, ACM, 2025.

[36] Agata M. Brańczyk et al., *Qiskit addon: circuit cutting*, 2024.

[37] Junya Nakamura, Takahiko Satoh, and Shinichiro Sanji, *Improved sampling bounds and scalable partitioning for quantum circuit cutting beyond bipartitions*, arXiv, 2025.

[38] Felix Burt, Kuan-Cheng Chen, and Kin K. Leung, "Generalised Circuit Partitioning for Distributed Quantum Computing," in *Int'l Conf. Quantum Comput. & Eng.*, 2024.

[39] Marc Grau Davis, Joaquin Chung, Dirk Englund, and Rajkumar Kettimuthu, *Towards Distributed Quantum Computing by Qubit and Gate Graph Partitioning Techniques*, arXiv, 2023.

[40] Eesa Nikahd, Naser Mohammadzadeh, Mehdi Sedighi, and Morteza Saheb Zamani, "Automated window-based partitioning of quantum circuits," *Phys. Scr.*, 2021.

[41] Oliver Crampton et al., "A Genetic Approach to Minimising Gate and Qubit Teleportations for Multi-Processor Quantum Circuit Distribution," en, *J. Quantum Comput.*, 2025.

[42] Konstantin Andreev and Harald Räcke, "Balanced graph partitioning," in *ACM Symp. Parallel Algorithms & Archit.*, ACM, 2004.

[43] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, 1970.

[44] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Python in Science Conf.*, 2008.

[45] Gurobi Optimization, LLC, *Gurobi Optimizer Reference Manual*, 2024.

[46] Nils Quetschlich, Lukas Burgholzer, and Robert Wille, "MQT Bench: Benchmarking software and design automation tools for quantum computing," *Quantum*, 2023.