Cut&Shoot: Cutting & Distributing Quantum Circuits Across Multiple NISQ Computers

Giuseppe Bisicchia*
University of Pisa, Pisa, Italy
giuseppe.bisicchia@phd.unipi.it

Alessandro Bocci*
University of Pisa, Pisa, Italy
alessandro.bocci@unipi.it

José García-Alonso
University of Extremadura, Cáceres, Spain
jgaralo@unex.es

Juan M. Murillo
University of Extremadura, Cáceres, Spain
juanmamu@unex.es

Antonio Brogi
University of Pisa, Pisa, Italy
antonio.brogi@unipi.it

***Abstract*—Currently, quantum researchers are facing the limitations of NISQ (Noisy Intermediate-Scale Quantum) computers, such as limited qubit counts and high sensitivity to external interference. In response to these challenges, two techniques have recently been proposed, among others, to enhance the execution of quantum circuits: *circuit cutting* and *shot-wise distribution*. *Circuit cutting* partitions large quantum circuits into smaller fragments that can be executed on today's NISQ devices. *Shot-wise distribution* leverages multiple heterogeneous quantum computers by distributing the *shots* of a single quantum circuit execution across various NISQ devices, improving resiliency and reducing overall execution time, in a flexible and customisable way. In this work, we investigate the synergic effect of a seamless composition of these two techniques into a unified pipeline of our own design, termed *Cut&Shoot*. This pipeline fragments a single quantum circuit into multiple independent sub-circuits, distributes the shots of each fragment across various quantum computers, and finally merges the results to reconstruct the output for the original circuit. Noisy simulation experiments seem to indicate that circuit cutting effectively reduces error rates, without the need to increase the number of shots performed. Shot-wise distribution, instead, while offering several qualitative advantages, does not significantly affect error rates, with negligible time overhead.**

***Index Terms*—Quantum Computing, Quantum Software Engineering, Quantum Cloud Computing, Distributed Quantum Computing, Shot-Wise Distribution, Circuit Cutting**

I. Introduction

Quantum computing is advancing rapidly, yet it remains an emerging field [1], [2]. Consequently, much of the research focuses on developing the fundamentals of the quantum technology stack – both hardware and software [3]–[5] – while contending with the limitations of *Noisy Intermediate-Scale Quantum* (NISQ) devices, such as limited qubit counts and high sensitivity to external interference [6]. To address these challenges, the nascent field of Quantum Software Engineering [7], [8] is exploring ways to enhance the efficiency and effectiveness of current and future quantum computations [9]. This paper explores the combination of these methodologies aimed at improving the execution of quantum circuits on NISQ (and potentially future) devices.

* These authors contributed equally to this work.

Circuit cutting: proposed by Peng *et al.* in 2019 [10], addresses devices' limited qubit counts by partitioning a large quantum circuit into smaller sub-circuits (or *fragments*). These fragments are executed independently on current NISQ computers, and their outputs are merged to reconstruct the output of the original large circuit.

Shot-wise distribution: introduced by Bisicchia *et al.* in 2023 [11], [12], enhances the resilience and flexibility of quantum computations by distributing the shots of a single quantum circuit execution across multiple quantum computers or *Quantum Process Units* (QPUs). This approach also allows quantum developers to embed custom user- and scenario-specific requirements and *Quality of Service* (QoS) attributes and objectives into the quantum execution process.

The objective of this work is to investigate which are the potential advantages of *seamlessly* combining *circuit cutting* and *shot-wise distribution*. To this end, we prototyped a software pipeline, termed *Cut&Shoot*, that *synergically* integrates the process of cutting a circuit into multiple fragments with an independent shot-wise distribution of the shots of these fragments across multiple NISQ computers.

This combination retains the qualitative benefits of both techniques:

1) ability to execute circuits *larger* than available QPUs,
2) increased *resilience* to QPU *failures*, and
3) enhanced *flexibility* and *customisability* in the execution process of quantum circuits.

To experiment with our proposal, we conducted noisy simulation experiments to investigate the impact of *Cut&Shoot* classical computing time overhead (due to classical pre- and post-processing) and execution error rates.

In the rest of this paper, we first recall the essence of the circuit cutting and shot-wise methodologies (Sect. II). Then, we present our *Cut&Shoot* pipeline (Sect. III). Next, we describe the experimental setup and protocol we designed to experiment with our proposal (Sect. IV) and discuss the results of the experimentation (Sect. V). Finally, we draw conclusions and mention some potential future research directions (Sect. VI).

II. Background

A. Circuit Cutting

Quantum circuit cutting is a technique [10] designed to overcome the limitations of NISQ devices in executing circuits with a qubit size higher than their qubit counts. It allows quantum developers to partition large quantum circuits into smaller, more manageable sub-circuits (*fragments*), suitable for current QPUs, at the cost of classical pre- and post-processing overhead. The results of these fragment executions are then combined to reconstruct the original circuit's outcome.

The process involves identifying points in the quantum circuit that can be cut, based on the circuit's structure and the interactions between qubits. Searching for optimal cut points is crucial, as it determines the number of fragments generated and the complexity of the post-processing phase. Current search strategies aim to select cuts that minimise the time needed to reconstruct the original circuit's results.

Once the cut points are identified, the circuit is divided into smaller fragments, possibly introducing *ancilla qubits* at the cut points to handle the connections between fragments and ensure accurate quantum correlations. Then, multiple *fragment variations* are generated, initialising the ancilla qubits at multiple different specific states to enable the classical post-processing to sew together the various fragments. The sewing procedure involves exponential classical time complexity in the number of fragments generated. Circuit cutting offers several advantages. By reducing the number of qubits in the executed circuits, it makes it feasible to run larger circuits on NISQ devices with limited qubit counts. Additionally, it can reduce the single fragments' depth, which may reduce the impact of noise and errors in quantum computations.

In the literature, circuit cutting has been investigated from various perspectives. We mention below some key works that have proposed circuit cutting techniques implemented in open-source software. Tang *et al.* [13] introduced *CutQC*, a circuit cutting method that employs *Mixed Integer Programming* to model circuits and find optimal cuts to minimise the post-processing time needed to reconstruct the results. Lowe *et al.* [14] proposed a circuit cutting method that relies on graph partitioning to determine optimal cut points in the circuit, minimising the post-processing overhead. Their method was assessed on *Quantum Approximate Optimization Algorithm* (QAOA) circuits for the Max-Cut problem on graphs. Piveteau and Sutter [15] employed quasiprobability simulations of non-local gates with operations that act locally on the fragments. This technique is at the core of the *Circuit Knitting Toolbox*, a circuit cutting Python library included in the IBM Qiskit ecosystem. Few studies have investigated the execution of quantum circuit fragments across different QPUs. To the best of our knowledge, none have specifically addressed the distribution of *shots* of these fragments on multiple QPUs, especially in environments lacking quantum channels among the QPUs. For instance, in [16], the authors leverage non-maximally entangled qubit pairs to partition and distribute whole circuit fragments. Similarly, [17] presents a circuit-cutting technique that facilitates the distribution of quantum computations at the level of entire fragments, but not at the granularity of individual shots.

B. Shot-Wise Distribution

Shot-wise distribution is a methodology [11] for partitioning the quantum computation workload of a single circuit execution across multiple NISQ devices. The rationale is to distribute the shots of a quantum circuit execution across multiple devices, with each device executing a fraction of the total shots. The results of these executions are then combined to obtain the final result of the circuit. Shot-wise distribution is based on the observation that each shot is an independent sample of the circuit's output distribution.

After selecting the set of NISQ devices and the number of shots for the circuit, the first step in shot-wise distribution is to decide how to *split* the shots among the NISQ devices. The split can be done uniformly, with each device receiving the same number of shots, or based on user-defined policies. The circuit is then executed on the different NISQ devices with the allocated shots. Finally, the results of the executions are combined in a *merge* step using classical post-processing techniques to obtain the final result. The merge step can also be performed in various ways, such as a weighted combination of the results or according to a user-defined policy.

Shot-wise distribution increases the resilience of quantum computation, as the failure of a device does not compromise the entire computation. It also speeds up the average execution time of circuits by enabling the parallelisation of the shots on multiple different machines. On the other hand, shot-wise distribution introduces a small classical computational overhead in both the pre-processing phase to split the shots and the post-processing phase to merge the results.

Bisicchia *et al.* proposed [12] a shot-wise distribution methodology designed to abstract the complexity of managing compilers, QPUs, and their cloud providers. Their focus is on the customizability of the split and merge policies (e.g., execution and waiting time, cost, expected fidelity, energy footprint, and legal regulatory compliance). The methodology is implemented in a Python tool termed *Quantum Broker*.

III. *Cut&Shoot*: How To

Cut&Shoot is designed to be modular and flexible. It allows the customisation of the cutting tool, the fragments' initial shot allocation policy, and the split and merge policies for shot-wise distribution. The pipeline is also agnostic to the programming language used to specify quantum circuits and the cutting tool.

Cut&Shoot comprises four main steps (Figure 1):

1) **Cut:** The original circuit is divided into smaller fragments using the chosen *cutting tool*. The tool outputs all possible fragment variations, which are then treated as standalone quantum circuits with their respective shot allocation determined by a custom *allocation policy*.
2) **Split:** The shots for each fragment variation are distributed across the target NISQ devices according to the selected *split policy*. The shots of each fragment

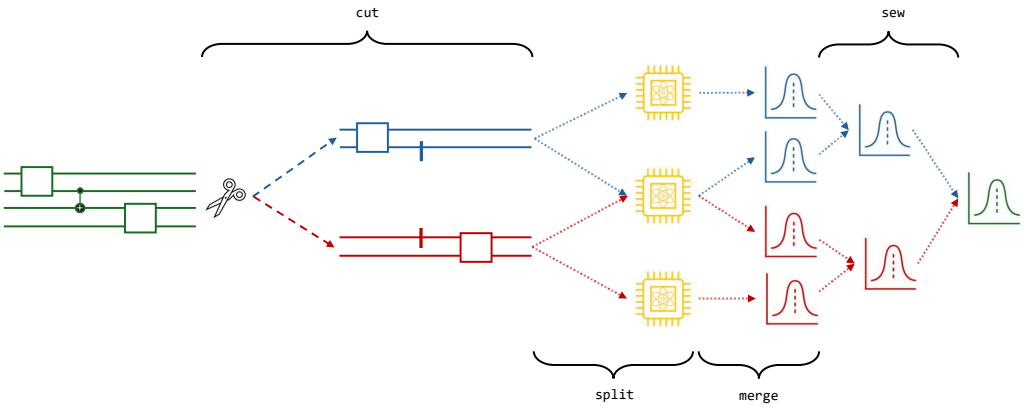


Fig. 1. The *Cut&Shoot* pipeline.

variation are executed independently and concurrently on the target QPUs.

3) **Merge:** The execution results from the NISQ devices are merged to obtain the probability distribution for each fragment through the user-specified *merge policy*, which may differ from the *split policy*.
4) **Sew:** Finally, the probability distributions of the fragments are combined to reconstruct the original circuit's probability distribution.

Cut&Shoot is a Python open-source software tool[1]. For the *Cut* and *Sew* steps, we leverage the circuit cutting implementation from [14], part of the PennyLane Python Library[2]. Due to the modularity of our pipeline, the cutting tool can be easily replaced with any other cutting tool. We chose this particular tool because it integrates seamlessly into the pipeline and is well-documented. The PennyLane cutting tool automatically detects cut points in circuits based on the desired number of qubits per fragment and the total number of fragments. It outputs all possible fragment variations, ready for execution.

IV. Experimental Setup

To experiment with our proposal and measure the performance of the synergistic combination of circuit cutting and shot-wise distribution offered by our *Cut&Shoot* pipeline, we performed a series of noisy simulation experiments[3]. These experiments varied the initial shot allocation policies, the number of QPUs used to distribute the quantum workload and involved circuits of different qubit sizes. Specifically, we conducted the simulations on a virtual machine with 8 Intel(R) Xeon(R) Gold 5218 CPUs (2.30GHz) and 32 GB RAM, running Ubuntu 22.04.3 LTS. We used *Python3.11* and the *Fake Backends* provided by *Qiskit*[4] for our simulations.

We utilised a total of 8 fake backends (Brisbane, Kyoto, Osaka, Sherbrooke, Cusco, Kawasaki, Kyiv, and Quebec). The quantum circuits were QAOA circuits for the Max-Cut problem[5], with the target graph structures as discussed in [14]. We used a total of 15 circuits of different sizes, as illustrated in Table I. The circuits were randomly generated through the algorithms described in [14] and filtered to select those effectively simulable in our environment. During all the experiments no optimisations (both during compilation and execution), nor error correction or error mitigation techniques were performed, to avoid biases in our results.

TABLE I
Main features of circuits used in the experiments.

Size (in #Qubits)	#Circuits	#Gates	Depth
5	5	27	12
10	5	57-72	15-21
15	5	99-105	21-36

Experimental Protocol

For each circuit, we computed the theoretical expected value (*ground truth*) and performed the shots on each noisy simulated QPU, both with and without circuit cutting using the proposed pipeline[6]. In addition to the theoretical expected value, we collected three different baselines for each circuit:

1) Running the whole circuit on each single QPU.
2) Applying circuit cutting and executing all fragments on the same QPU (for each QPU).
3) Running the whole circuit but distributing the shots across all the QPU subsets.

Finally, we compared these baselines with the experiments where we cut the circuit and distributed the shots of the fragments by applying *Cut&Shoot* on all possible QPU subsets.

Regarding the number of shots to execute, we experimented with two different allocation policies. For both policies, we set the number of shots for the entire circuit at 10,000, as higher values may exceed the maximum limits set by quantum providers. The first policy, termed *shots multiplier*, allocates 10,000 shots to *each fragment variation* before applying the

[1]https://github.com/di-unipi-socc/Cut-Shoot

[2]PennyLane v0.37: https://pennylane.ai/

[3]The raw data, as well as the code for performing the experiments, collection and analysis of those data, is open-source and freely available at: https://github.com/di-unipi-socc/Cut-Shoot/tree/main/experiments

[4]qiskit v1.0.2, qiskit-aer v0.14.2, qiskit-ibm-provider v0.10.0, and qiskit-ibm-runtime v0.23.0.

[5]For our experimentation, we texted the circuits with the *Pauli Z* observable for all the qubits.

[6]In our experiments, we set the circuit cutting tool with the following parameters: number of qubits per fragment in the interval $[2, q-1]$, where q is the maximum number of qubits in the circuit, and the number of fragments between $[2, (q/2)+1]$.

shot-wise distribution. For instance, if a circuit is cut into 2 fragments and each fragment has 2 variations, each variation has allocated 10,000 shots, resulting in a total of 40,000 shots for the experiment run. When circuit cutting is not applied, the whole circuit is executed for 10,000 shots. The second policy, termed *shots divider*, allocates 10,000 shots to *each experiment run* before applying the shot-wise distribution. When circuit cutting is applied, the 10,000 shots are equally distributed to each fragment variation. If a circuit is cut into 2 fragments and each fragment has 2 variations, each variation has allocated 2,500 shots, maintaining the total number of shots at 10,000.

The rationale behind the *shots multiplier* policy is to ensure that each circuit fragment is executed with the same number of shots as the entire original circuit to achieve the same level of accuracy. This approach results in a total number of shots proportional to the number of fragment variations. In contrast, the *shot divider* policy maintains a constant overall number of shots but may reduce the accuracy of the results due to the division of shots among fragments. Therefore, under the *shots divider* policy, we can expect results with higher noise compared to the *shots multiplier* policy. However, this work focuses on exploring the behaviour of the *Cut&Shoot* pipeline across various scenarios, rather than determining the optimal policy, which will be addressed in future research.

In our experiments, when using shot-wise distribution, the shots allocated to each circuit or fragment follow a *fair* distribution policy. An equal number of shots is assigned to each selected QPU, which is also the policy used during the merge phase. As the main objective of this paper is to have an initial validation of our proposal, we decided to apply only a straightforward policy to avoid potential biases and influences by more sophisticated policies (e.g., which consider the noise of the QPUs). We run the experiments with and without the shot-wise distribution for each possible QPU subset, to measure the impact of this technique in the executions.

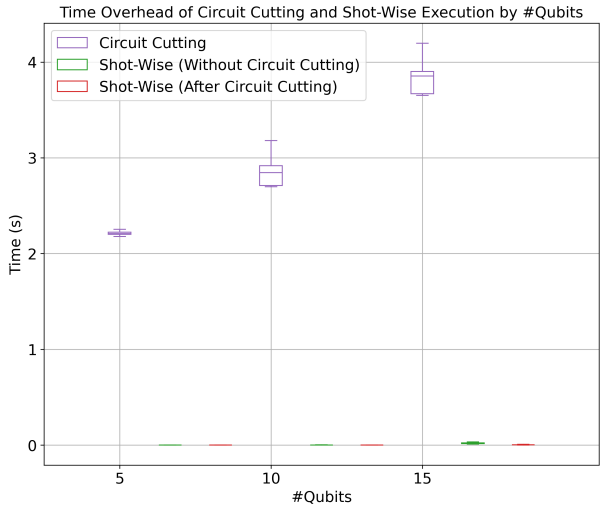
V. Experimental Results & Discussion

In this work, we focused our attention on analysing the classical computational time overhead of the pipeline and the errors of the quantum computations.

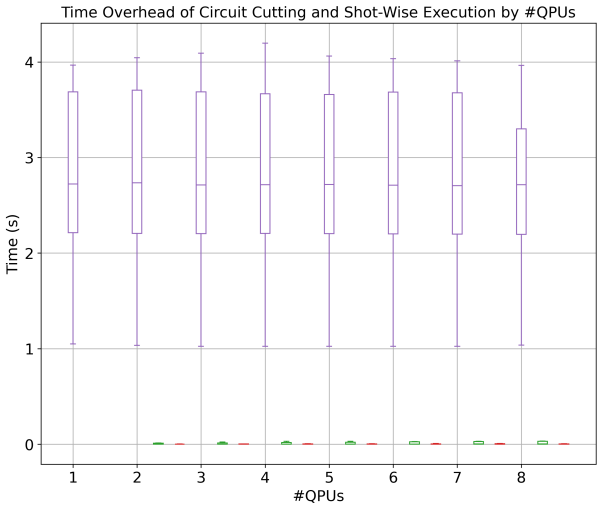
Classical Computational Time Overhead - We measured the time overhead for classical pre- and post-processing for the circuit cutting (cut and sew phases of Fig. 1) and the shot-wise distribution (split and merge phases of Fig. 1). We investigated the time overhead from two perspectives (Fig. 2): by varying the number of qubits and by varying the number of QPUs used. We observed that in both cases:

- the time required for circuit cutting is *significantly higher* than shot-wise distribution (order of seconds versus milliseconds),
- circuit cutting time increases with the number of qubits and is *unaffected* by the number of QPUs, and
- shot-wise distribution time is always almost *negligible* compared to the circuit cutting time.

Note that the results above are independent of the shot allocation policy applied (*shots divider* or *shot multiplier*).



(a)



(b)

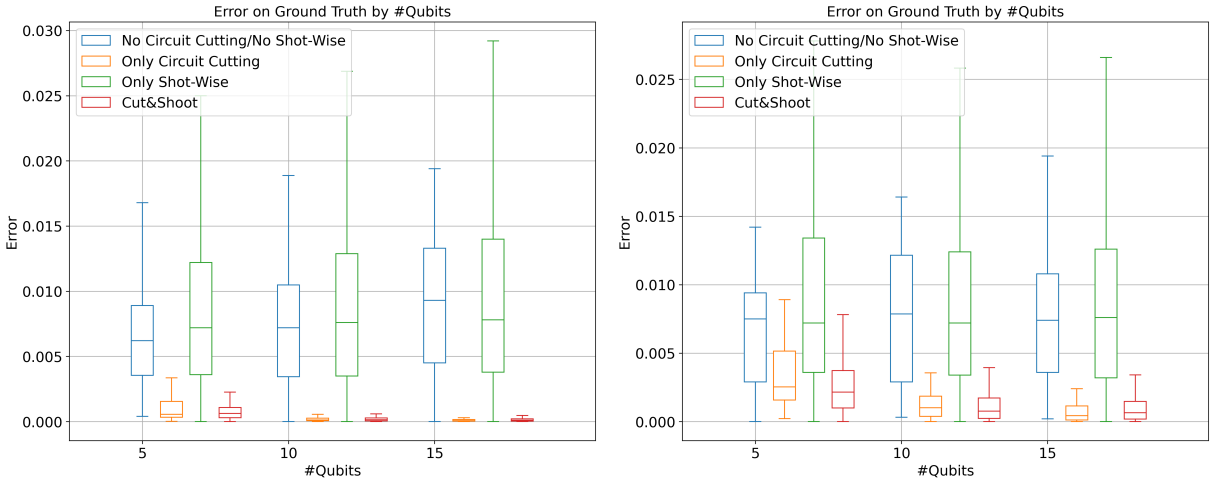
Fig. 2. Time overhead for circuit cutting and shot-wise distribution.

Error rates - We compute the error by comparing each circuit's execution to the theoretical expected value. We consider four scenarios with varying numbers of QPUs: by applying neither/either/both circuit cutting and shot-wise distribution. When shot-wise distribution is not employed, execution occurs on a single machine but for all considered QPUs. When shot-wise distribution is employed, instead, circuits are executed across different machine combinations and we analysed the whole powerset of the 8 considered QPUs.

We show the results from two different perspectives, by varying the number of qubits of the circuits and by varying the number of QPUs employed by the shot-wise distribution process, for both shot allocation policies discussed in Sect. IV (*shots divider* and *shots multiplier*).

The results of the first perspective are depicted in Fig. 3:

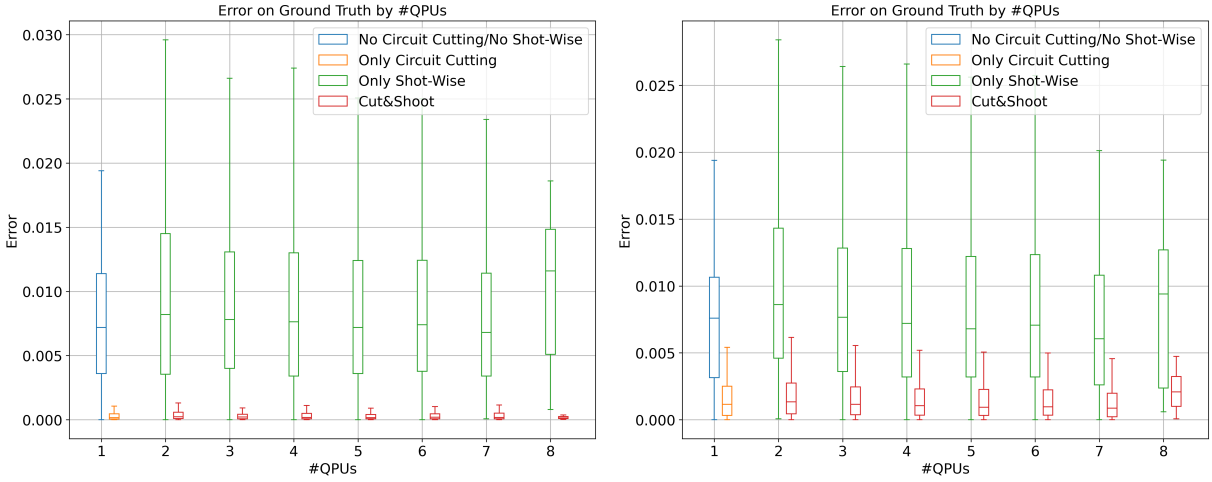
- as expected, the plain execution of the whole circuit and shot-wise distribution alone have an error that increases with the number of qubits,
- circuit cutting alone and our pipeline *consistently and sig-*



(a) *Shots multiplier* policy.

(b) *Shots divider* policy.

Fig. 3. Error on the ground truth by circuit size.



(a) *Shots multiplier* policy.

(b) *Shots divider* policy.

Fig. 4. Error on the ground truth by number of QPUs employed by the shot-wise distribution.

nificantly reduce the error rate, especially as the number of qubits increases, and

- the error rate observed with the *shots multiplier* policy is consistently lower compared to the *shots divider* policy. This behaviour can be explained by the higher number of shots performed with the *shots multiplier* policy.

The error behaviour of circuit cutting and our pipeline is somewhat unexpected because neither of our shot allocation policies increases the overall number of shots with the number of qubits, as it should be to maintain the same level of accuracy while the number of output states (exponential in the number of qubits) increases. Consequently, we would expect a loss in accuracy as the number of qubits grows. Indeed, with the *shots multiplier* policy, the overall number of shots increases proportionally to the number of fragment variations, which in our experiments ranged from 2 to 8 (averaging 7) independently from the number of qubits. The *shots divider* policy keeps the number of shots fixed at 10,000. Therefore, despite expectations, circuit cutting and our pipeline significantly reduce error rates as the number of qubits increases.

The results of the second perspective, by varying the number of QPUs, are depicted in Fig. 4:

- circuit cutting reduces significantly the error rate for each subset of QPUs considered,
- applying shot-wise distribution does not significantly increase the error rate, whether used alone or within our pipeline.

As previously observed, both policies exhibit similar behaviour, with the *shots multiplier* policy resulting in lower error rates compared to the *shots divider* policy, due to the higher number of shots and the consequent increased accuracy. However, for the purposes of this work, note that the relative error behaviour of the different approaches remains consistent.

Discussion **-** Our experiments provide further evidence [13], [18] that circuit cutting reduces error rates and that combining it with shot-wise distribution does not significantly compromise these benefits. The time overhead introduced by shot-wise distribution is negligible compared to circuit cutting.

Furthermore, the shot-wise distribution methodology enhances our *Cut&Shoot* pipeline by increasing resilience to QPU failures and offering higher flexibility and customisability in the execution process. Moreover, all our findings were also validated through the *Student's t-test*, confirming the statistical relevance of our results. The assessment performed with the two different initial shot allocation policies further helps and strengthens our initial findings. Indeed, while the absolute error differs, the relative behaviour is always the same, bringing always to the same conclusions. It is also interesting to notice, especially for the *shots divider* policy (but the same holds with a few adjustments also for the *shots multiplier* policy) that the same number of shots is performed independently from the size of the circuit considered. However, increasing the size of the circuit, the number of output states increases exponentially. Therefore, our *"resolution power"* decreases if we keep fixed the number of shots, and as a consequence, we expect the noise to increase. This is indeed true when circuit cutting is not applied. However, when we apply circuit cutting this behaviour seems reversed. In fact, despite the increase in the size of the circuits and, consequently the exponential increase of the state spaces and the theoretical loss in accuracy, the errors when circuit cutting is applied decrease with the number of qubits. A possible explanation for this behaviour is that by increasing the size of the circuits, more fragments with small qubits are generated, which are therefore subjected to less noise. As a final remark, the shot-wise distribution methodology can incorporate distribution policies that consider QPU noise, potentially offering further noise reduction when integrated with circuit cutting. This hypothesis requires future experimental validation.

VI. Concluding remarks

In this paper, we investigated the potential advantages of seamlessly and synergically combining the *circuit cutting* [10] and *shot-wise distribution* [12] quantum methodologies through a pipeline we designed and developed, termed *Cut&Shoot*. To evaluate the performance of our approach, we conducted a series of noisy simulation experiments. The results seem to indicate that employing our pipeline leads to:

1) *negligible time overhead* of the shot-wise classical pre- and post-processing compared to applying only circuit cutting, and
2) *lower error rates* when circuit-cutting is applied (both with and without shot-wise). Additionally, applying the shot-wise distribution does not significantly affect the error rates compared to executing all shots on a single NISQ device.

Moreover, it is worth noting that the pipeline may reduce the overall execution time for a circuit, achieved through the parallelisation facilitated by the shot-wise distribution methodology and the decreased number of shots enabled by circuit cutting. However, as this initial work involved only simulation assessments, we plan to conduct more experiments on real NISQ computers to validate all our findings and assess potential time savings. Finally, the *Cut&Shoot* pipeline retains the benefits of both approaches in a synergistic manner:

1) enabling the execution of circuits larger than the available QPUs,
2) increasing resilience to QPU failures, and
3) enhancing the flexibility and customisability of the quantum circuit distribution and execution process.

In future work, we plan to:

- developing a holistic circuit cutting search strategy that considers the execution phase, potentially determining the optimal cut by also considering which QPU(s) will execute the fragments' shots,
- leveraging the modularity of our prototype to integrate different circuit cutting strategies and distribution policies (e.g., noise-aware) to feature a richer set of combinations within our pipeline, possibly *as-a-Service*, and
- expanding the pipeline to include other phases, for instance, compilation [19], advanced semantics [20], or simultaneous execution within the same QPU through multi-programming [21]–[23].

References

[1] V. Sood *et al.*, "Archives of quantum computing: research progress and challenges," *Arch. Comput. Meth. Eng.*, vol. 31, no. 1, pp. 73–91, 2024.
[2] J. M. Murillo, *et al.*, "Challenges of quantum software engineering for the next decade: The road ahead," *arXiv:2404.06825*, 2024.
[3] F. Arute *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.
[4] Y. Kim *et al.*, "Evidence for the utility of quantum computing before fault tolerance," *Nature*, vol. 618, no. 7965, pp. 500–505, 2023.
[5] X. Zhou *et al.*, "Electron charge qubit with 0.1 millisecond coherence time," *Nature Physics*, vol. 20, no. 1, pp. 116–122, 2024.
[6] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, p. 79, 2018.
[7] M. Piattini *et al.*, "The talavera manifesto for quantum software engineering and programming." in *QANSWER*, 2020, pp. 1–5.
[8] J. Zhao, "Quantum software engineering: Landscapes and horizons," *arXiv preprint arXiv:2007.07047*, 2020.
[9] G. Bisicchia *et al.*, "From quantum mechanics to quantum software engineering," *arXiv preprint arXiv:2404.19428*, 2024.
[10] T. Peng *et al.*, "Simulating large quantum circuits on a small quantum computer," *Phys. Rev. Lett.*, vol. 125, p. 150504, 2020.
[11] G. Bisicchia *et al.*, "Dispatching shots among multiple quantum computers: An architectural proposal," in *IEEE QCE*, vol. 02, 2023.
[12] ——, "Distributing quantum computations, by shots," in *ICSOC*, 2023.
[13] W. Tang *et al.*, "Cutqc: using small quantum computers for large quantum circuit evaluations," in *ASPLOS*, 2021, p. 473–486.
[14] A. Lowe *et al.*, "Fast quantum circuit cutting with randomized measurements," *Quantum*, vol. 7, p. 934, 2023.
[15] C. Piveteau *et al.*, "Circuit knitting with classical communication," *IEEE Trans. Inf. Theory*, vol. 70, no. 4, pp. 2734–2745, 2024.
[16] M. Bechtold *et al.*, "Circuit cutting with non-maximally entangled states," *arXiv preprint arXiv:2306.12084*, 2023.
[17] S. Kan *et al.*, "Scalable circuit cutting and scheduling in a resource-constrained and distributed quantum system," *arXiv:2405.04514*, 2024.
[18] M. A. o. Perlin, "Quantum circuit cutting with maximum-likelihood tomography," *npj Quantum Information*, vol. 7, no. 1, p. 64, 2021.
[19] M. Salm *et al.*, "Automating the comparison of quantum compilers for quantum circuits," in *CCIS*, vol. 1429, 2021, pp. 64–80.
[20] B. Weder *et al.*, "Analysis and rewrite of quantum workflows: Improving the execution of hybrid quantum algorithms." in *CLOSER*, 2022.
[21] P. Das *et al.*, "A case for multi-programming quantum computers," in *IEEE/ACM MICRO*, 2019, pp. 291–303.
[22] Y. Ohkura *et al.*, "Simultaneous execution of quantum circuits on current and near-future nisq systems," *IEEE TQE*, vol. 3, pp. 1–10, 2022.
[23] Romero-Alvarez *et al.*, "Quantum circuit scheduler for qpus usage optimization," *arXiv preprint arXiv:2404.01055*, 2024.