Quantum Machine Learning with Circuit Cutting

Saasha Joshi
Dept. of Computer Science
University of Victoria
Victoria, British Columbia, Canada
saashajoshi08@gmail.com

Angadh Singh
Dept. of Computer Science
University of Victoria
Victoria, British Columbia, Canada
angadh.singh1@gmail.com

***Abstract*—Quantum Machine Learning (QML) techniques, including variational Quantum Tensor Networks (QTN), pose a huge implementation challenge regarding qubit requirements. An approach to circumvent this issue is to perform circuit cutting that segments large quantum circuits into multiple smaller sub-circuits that can be trained easily on a quantum device [1]. This project lays down the workflow for training a variational QTN circuit that implements circuit cutting, specifically gate cutting, to perform data classification. The workflow is built with the help of the Qiskit SDK with dependence on the Circuit Knitting Toolbox [3] for circuit cutting procedures. Additionally, a significant amount of modifications are made to algorithms like SamplerQNN, a part of the Qiskit Machine Learning package, to accommodate the training of multiple sub-circuits with Qiskit Aer's Sampler runtime primitive. The dataset used for classification is the diabetes dataset from the National Institute of Diabetes and Digestive and Kidney Diseases publicly available on Kaggle.**

The project can be found on GitHub here.

***Index Terms*—Quantum Machine Learning, Circuit Cutting, Qiskit, Quantum Tensor Networks, Classification**

I. Introduction

Quantum Machine Learning (QML) represents a frontier in the utilization of near-term quantum computers, offering transformative potential across a spectrum of applications, notably in problem domains such as classification [8] and generative modeling [4]. The inherent complexity and computational demands of executing QML circuits, however, pose significant challenges. These challenges stem from the intensive computational resources and time required to run large QML circuits [1]. In addition, machine learning (ML) models like Quantum Tensor Networks (QTNs) present formidable challenges in terms of the quantum bit (qubit) resources required for implementation.

One viable strategy to overcome these hurdles involves employing circuit cutting techniques. These techniques allow for the division of expansive quantum circuits into smaller, more tractable sub-circuits. Such segmentation enables a more straightforward and efficient training process on quantum devices, as evidenced by previous studies [1]. This approach not only addresses the qubit resource challenge but also simplifies the overall training process.

Building upon this foundation, we propose a comprehensive workflow that integrates circuit cutting techniques into the training of QML models. This workflow leverages the Qiskit (SDK) [9] and incorporates the Circuit Knitting Toolbox [3], facilitating the division of large QML circuits into smaller, trainable sub-circuits. These sub-circuits are then trained according to two distinct workflows, as illustrated in Figure 4 and Figure 5. The training outcomes, form quasi-probability distributions, are subsequently utilized to reconstruct the expectation value of the original, unsegmented circuit.

To support our computational needs, we employ a Sampler primitive from Qiskit Aer, which enables access to graphics processing units (GPUs) provided by NVIDIA power-up. Additionally, we leverage local simulators, alongside SD1 and TN1 simulators, provided through AWS services. Further details on the deployment of these computational resources, including the specific roles of AWS and NVIDIA in facilitating our project, are provided [here]

II. Circuit Cutting

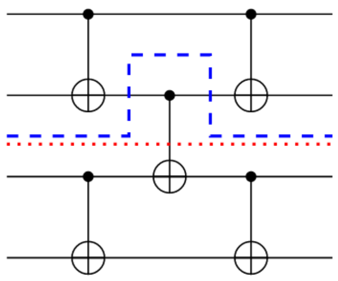


Fig. 1. Circuit Cutting with gate cuts (red) and wire cuts (blue) [5]

Circuit cutting is the method to partition a circuit into multiple smaller sub-circuits that can be executed independently of each other. This cutting procedure can be implemented through two distinctive approaches - gate cutting, shown via a red dotted line in Figure 1, or wire cutting, shown by a blue dashed line in Figure 1.

A gate cutting procedure involves the decomposition of a unitary gate into multiple quantum channels that can be run on a specific quantum device. This procedure helps in the reduction of the circuit depth or width, according to the problem statement. On the other hand, a wire cutting procedure partitions the circuit along the qubit wire in time. The resultant subcircuits of a wire cut, however, have a few extra qubits added. Both techniques reconstruct the final expectation value of the original circuit by sampling the quasi-distribution of the subcircuits and performing classical post-processing.

The Circuit Knitting Toolbox [3] by Qiskit provides implementation strategies for both gate and wire cutting procedures.

III. Methodology

A. Data Preparation

In this project, we aim to perform the classification on a diabetes dataset by the National Institute of Diabetes and Digestive and Kidney Diseases publicly available on Kaggle [6]. The dataset consists of 8 feature inputs that predict the chances of diabetes in a patient.

Diabetes Dataset							
Glucose	Blood Pressure	BMI	Insulin	Diabetes Pedigree Function	Pregnancies	Skin Thickness	Age

Fig. 2. Caption

Since the dataset consists of classical data, we perform data embedding to represent this data as quantum states. We use the angle embedding technique to represent the data features on an 8-qubit quantum circuit, as shown in Figure 3. The data is normalized in the range $[-\pi, \pi]$.

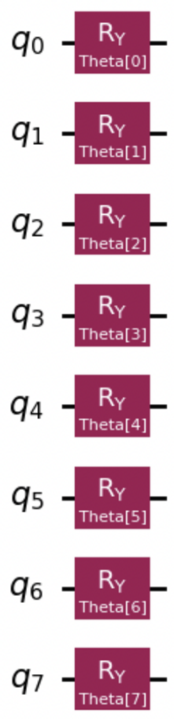


Fig. 3. Angle Embedding Circuit for Diabetes Dataset

For this project, the dataset is split into train and test sets, distributed in a 70:30 ratio. The training set encompasses 537 data samples, while the test set comprises 231 samples.

B. Training Workflow

The training of a QML model integrated with circuit cutting techniques can be performed using two possible workflows, as depicted in Figures 4 and 5.

a) Workflow A: In workflow A, sub-circuits generated after a circuit cutting undergo training using a subset of input data features. Evaluation is subsequently performed with respect to the original training labels. Following this, the circuits undergo a tuning stage, wherein variable parameters are updated based on the loss function computed in the evaluation stage. Upon obtaining optimal parameter values from the optimizer, the quasi-probability distributions derived from the sub-circuits are combined to reconstruct the expectation value of the original circuit.

This workflow facilitates an independent and parallel evaluation of sub-circuits over multiple iterations. The concurrent training can be performed with the help of existing Batching techniques in the Qiskit Runtime primitives [2].

b) Workflow B: Workflow B, also proposed in [7], involves training the sub-circuits to reconstruct the original expectation values after each training iteration. This is unlike *Workflow A*, where the reconstruction stage occurs after multiple iterations are performed on each sub-circuit. Subsequently, an optimization step is performed on the reconstructed expectation value to tune the variable parameters within the sub-circuits. The updated parameters are then utilized to finalize the training process. The ultimately reconstructed expectation value is further used for validation and testing purposes.

This workflow facilitates concurrent training over one training iteration at a time. This is unlike *Workflow A* that allows for the parallel execution of sub-circuits over multiple iterations. Following the completion of each training iteration, the sub-circuits undergo classical post-processing steps to reconstruct the original expectation value.

For this project, we have opted to adopt the Workflow A structure. This design decision aligns with considerations related to the constraints imposed by the current Qiskit stack, which currently lacks support for training sub-circuits using the existing *SamplerQNN* primitive. Additionally, this decision is influenced by the time limitations inherent in the project timeline.

C. Quantum Tensor Network and Circuit Cutting

Initially, a variational tree tensor network (TTN) circuit is constructed with qubits equal to the number of data features, in our case it makes a total of 8 qubits. A circuit cutting process is initiated that partitions the original TTN circuit into sub-circuits based on a predefined partitioning scheme - gate or wire cutting. In our case, we utilize gate cutting to split the TTN at specific gate locations, creating sub-circuits that can be executed independently. In the context of diabetes data analysis, where the relationships between features (such as glucose levels, BMI, age, etc.) may involve complex, nonlinear interactions, gate cutting can allow for a nuanced manipulation of these relationships in smaller, more manageable quantum operations (Figure 6).

For simplicity, we cut the QTN circuit such that each of the sub-circuits consists of half of the qubits in the original circuit.

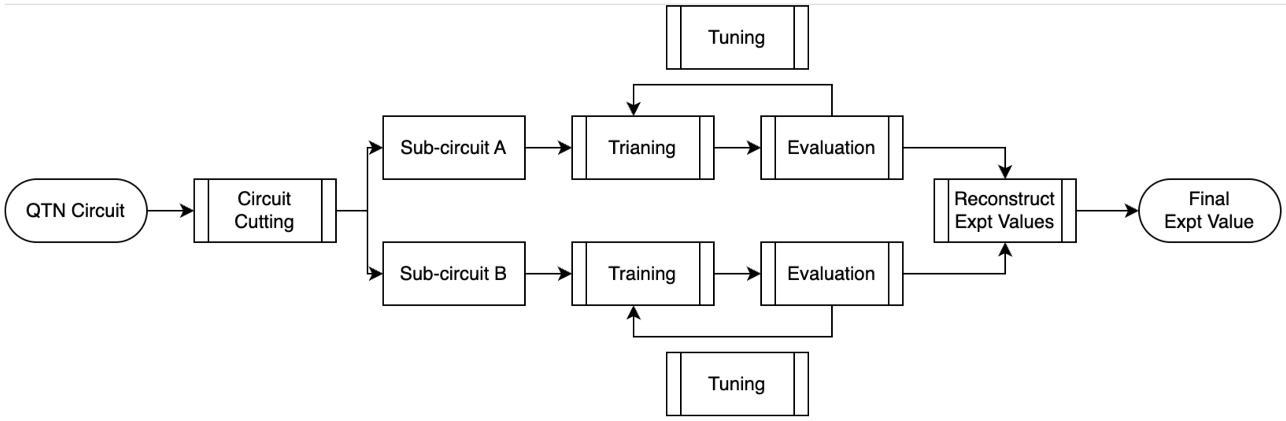


Fig. 4. QML model workflow (A) with Circuit Cutting

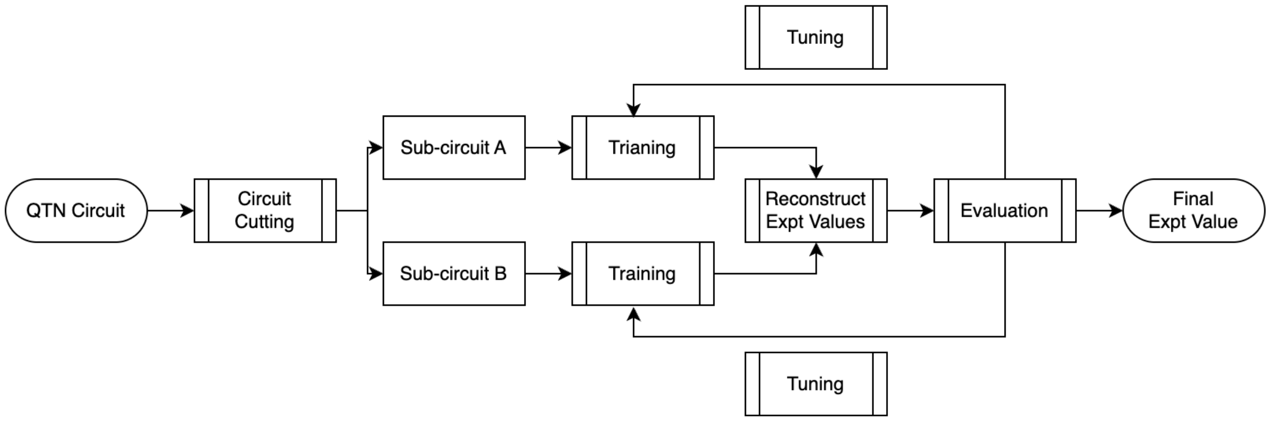


Fig. 5. QML model workflow (B) with Circuit Cutting

This results in sub-circuits having 7 and 8 trainable parameters, respectively. The sub-circuits are then composed of the embedding circuits that contain 4 feature inputs respectively (Figure 7).

Since we are using the gate cutting procedure to build the sub-circuits, each sub-circuit generates multiple different sub-experiments consisting of various single-qubit basis rotation gates at the location of the cuts. For one cut, a total of 6 sub-experiments are generated per sub-circuit. Hence, we have a total of 12 sub-experiments (sub-circuits) to train at the end of the circuit cutting procedure (Figure 8).

D. Training with CustomSamplerQNN

Next, a custom quantum neural network sampler (CustomSamplerQNN) is instantiated and acts as the core of the quantum neural network training process. The QTN sub-circuits are parameterized by input parameters (related to the data embedding) and variational weight parameters (related to the trainable aspects of the quantum circuits). The sampler provides the forward and backward pass methods for training the variational parameters in the QTN sub-circuits. This allows the network to learn from the data by adjusting the weight parameters. The neural network runs with two passes to evaluate the final output.

Forward Pass: The forward pass involves running the prepared QTN sub-circuits with the given input data and weight parameters. This step simulates the execution of the sub-circuits for the entire dataset, generating predictions or outputs based on the current state of the network's weights. The output of this step is a dictionary containing the results of the computations for each sub experiment, providing a detailed view of how the input data is transformed through the quantum circuits.

Backward Pass: Following the forward pass, a backward pass is conducted to compute gradients with respect to the input data and weight parameters. This is crucial for the training process, as it identifies how changes to the weight parameters affect the network's output. The gradients are used to adjust the weights in a way that minimizes the difference between the predicted outputs and the actual targets, thereby improving the model's accuracy over time.

This process is run for both training sets 'A' and 'B' such that we can evaluate the sub-circuits for each set of features encoded on each training data input.

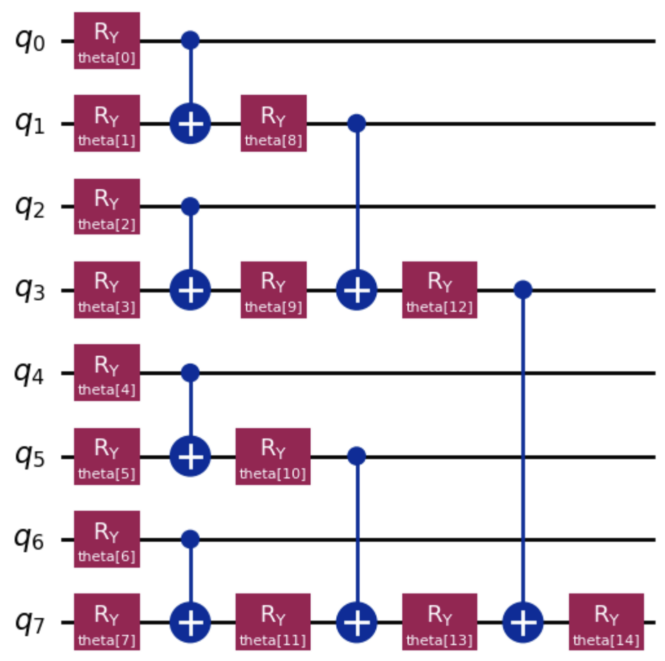


Fig. 6. Tensor Network for eight data features in diabetes data.

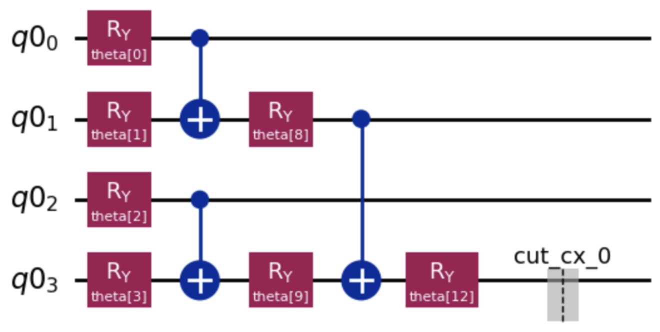


Fig. 7. Sub-circuit A with seven trainiable parameters

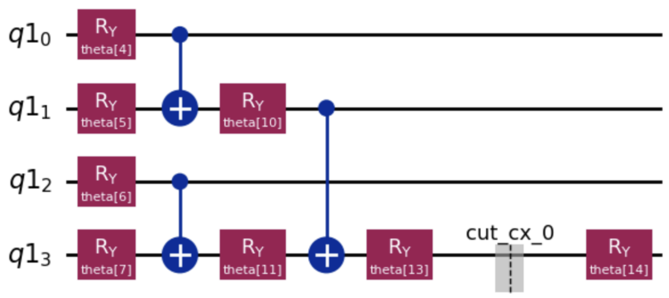


Fig. 8. Sub-circuit B with eight trainiable parameters

E. *Loss and Optimization*

The evaluation and tuning stages in the workflow run to refine the parameter values within the sub-circuits. Here, we define an objective function that governs the behavior of an optimizer, such as SPSA. Since we run the sub-circuits on a Sampler primitive that returns quasi-probability distributions upon the completion of its run, a multi-class objective function is defined. This objective function is invoked by a *minimize* function, to perform the optimization of variational parameters across all sub-circuits and returns updated parameter values.

In our project, we train and test our QML model with circuit cutting with the *L2* and *Cross Entropy* loss, while using *SPSA, COBYLA*, and *ADAM* as our optimizers. However, it is noticed during the training that a combination of *Cross Entropy* loss and *SPSA* performs significantly better than other loss and optimizers.

F. *Reconstruction of Expectation Values*

In the final stage of the workflow, the optimal parameter values received from the optimizer are used to perform a final forward pass of the QML model, and the final quasi-probability distributions from the sub-circuits are received. These probabilities are then used to reconstruct the expectation value of the original circuit. Here, we use the *reconstruct-expectation-value* function from the Circuit Knitting Toolbox. The expectation value is then processed to determine the predicted labels of the test data inputs.

IV. Evaluation and Discussion

The project uses an 8-qubit quantum tensor network that is trained to classify data inputs from the Diabetes dataset. The 8-qubit circuit undergoes circuit cutting and is split into two sub-circuits, A and B. These sub-circuits have 6 different implementations, i.e. combination of basis gates that are placed in the circuit cut locations.

One important thing to note is that the current implementation of the program supports only sequential training of the sub-circuits. This contributes to the increased time in training the circuits on CPU and GPU. However, since the workflow supports parallel support of the sub-circuits, an additional feature can be added as a future work to this project.

We run and train the sub-circuits on three different platforms:

- CPU: We use Qiskit's *AerSimulator* and *ibmq qasm simulators*
- GPU: GPU access provided by NVIDIA was utilized to perform training on the platform. We utilized Qiskit's *AerSimulator* on a GPU backend to simulate the training process.
- AWS AWS simulators, namely, *Local, SV1*, and *TN1* were used to perform training simulation.

Figures 14-17 show results of the training and training loss on different sub-experiments of the sub-circuits A and B. Figures 14-15 summarize the training loss for sub-circuits A and B respectively, run for 50 iterations on a CPU. Whereas,

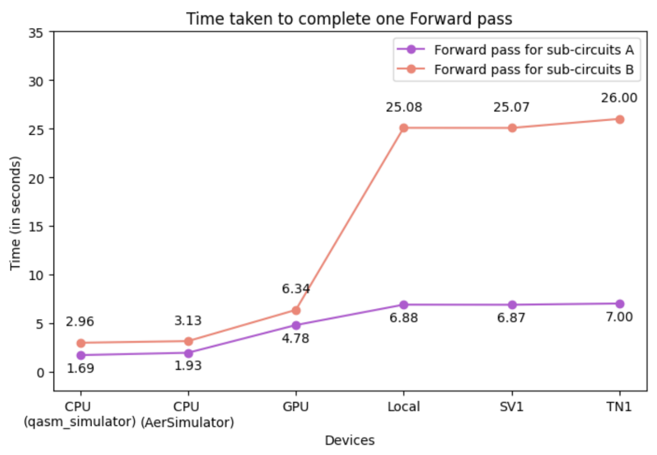


Fig. 9. Comparison of time taken to run one forward pass on different backends.

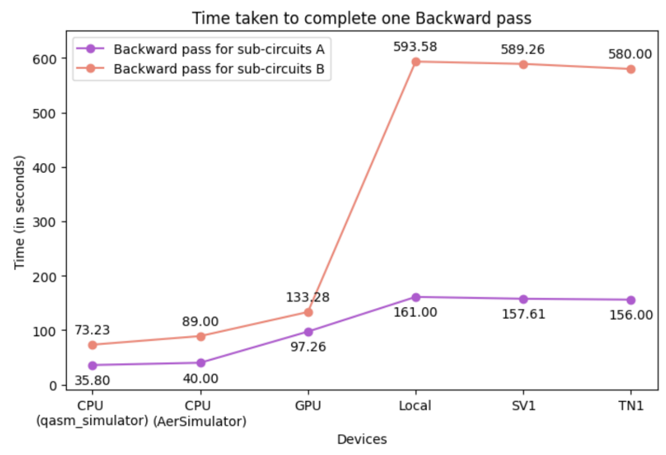


Fig. 10. Comparison of time taken to run one backward pass on different backends.

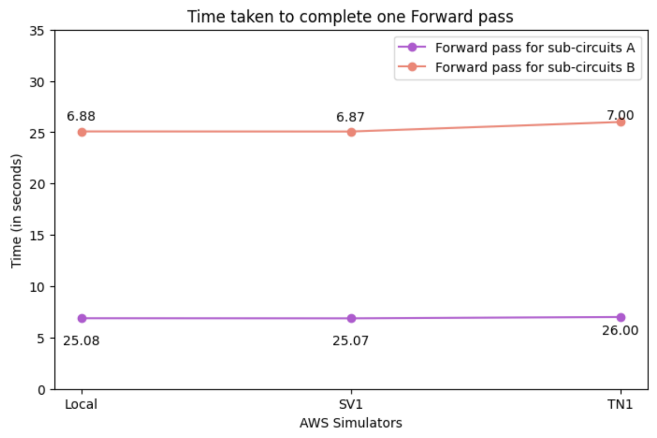


Fig. 11. Comparison of time taken to run one forward pass on different AWS backends.

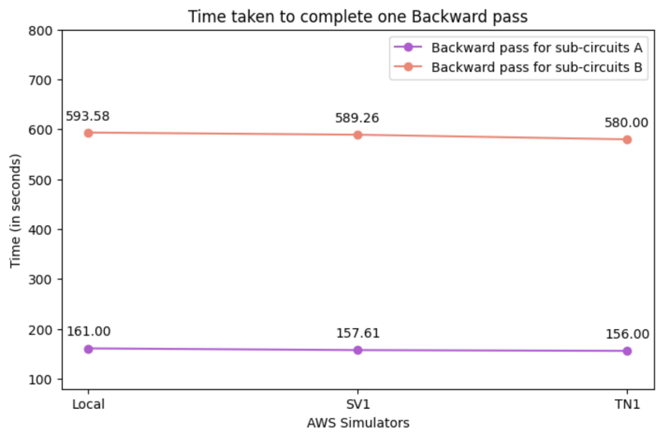
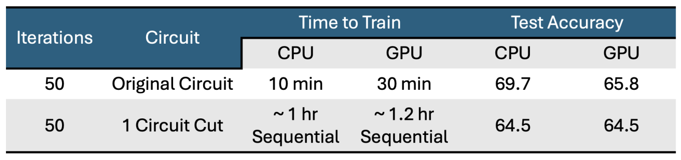


Fig. 12. Comparison of time taken to run one backward pass on different AWS backends.



Iterations	Circuit	Time to Train		Test Accuracy	
		CPU	GPU	CPU	GPU
50	Original Circuit	10 min	30 min	69.7	65.8
50	1 Circuit Cut	~ 1 hr Sequential	~ 1.2 hr Sequential	64.5	64.5

Fig. 13. Final results of training 8-qubit QML model with 1 circuit cut on CPU and GPU

Figures 16-17 show training loss for the same sub-circuits performed on a GPU.

V. Future Work

Improvements in the project can be made in the following parts:

- Use of better quantum library and source code that supports circuit cutting and QML training process. The current implementation makes edits in the Qiskit source

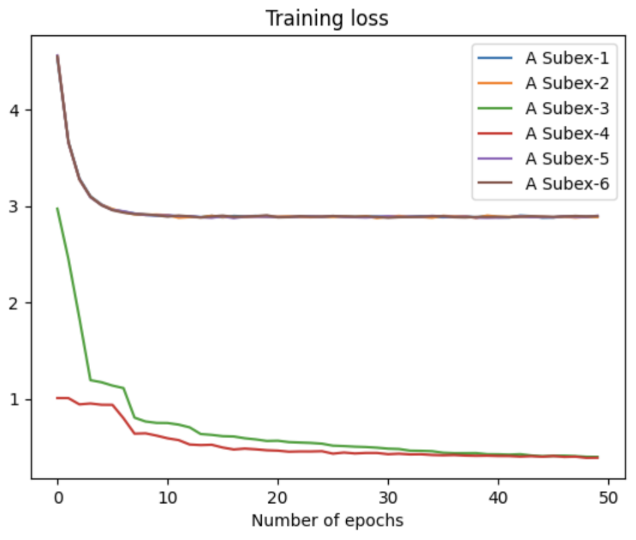


Fig. 14. Loss in training a 4-qubit sub-circuit (A) on a CPU for 50 iterations

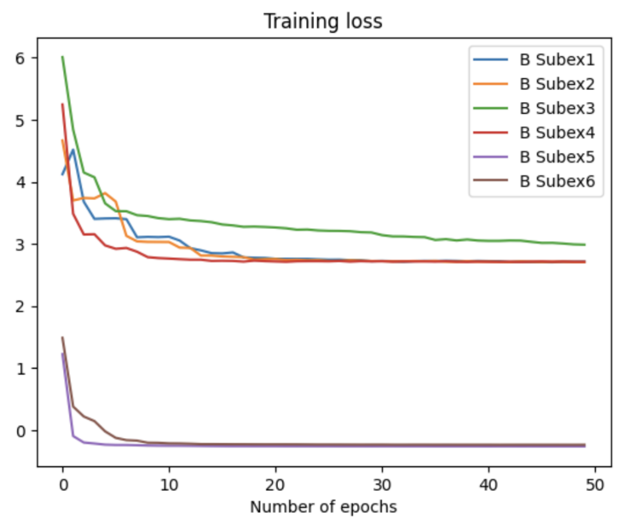


Fig. 15. Loss in training a 4-qubit sub-circuit (B) on a CPU for 50 iterations

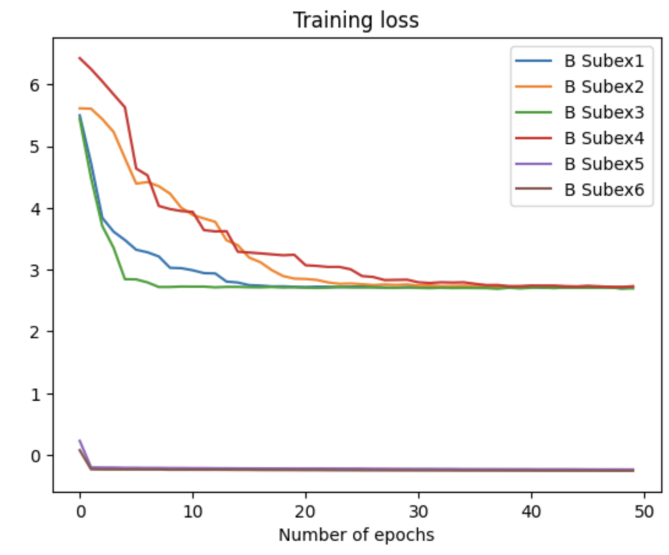


Fig. 17. Loss in training a 4-qubit sub-circuit (B) on a GPU for 50 iterations

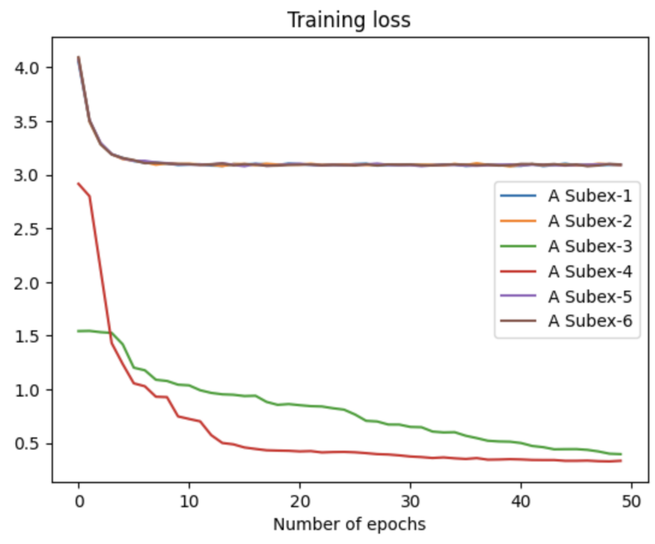


Fig. 16. Loss in training a 4-qubit sub-circuit (A) on a GPU for 50 iterations

code and runs only for a single cut in the circuit. Moreover, the sub-circuits in the project need to be of the same size to be executed with the code base.
- Introduction of parallelism using GPU can significantly improve the training time for sub-circuits. The current sub-circuits are running in sequential order.
- Better callback functions that can be applied to each sub-circuit. Since not every sub-circuit will have the same training loss graphs, we may require different callbacks for different sub-circuits.
- Support for multiple cuts and unequal sub-circuits is needed.
- Real backend support for mid-circuit measurements that occur in some of the sub-circuits is required.

References

[1] D. Guala, S. Zhang, E. Cruz, C. A. Riofrío, J. Klepsch, and J. M. Arrazola, “Practical overview of image classification with tensor-network quantum circuits,” Scientific Reports, vol. 13, no. 1, p. 4427, Mar. 2023, doi: https://doi.org/10.1038/s41598-023-30258-y.

[2] Gadi Aleksandrowicz, ‘Qiskit: An Open-source Framework for Quantum Computing’. Zenodo, Jan. 23, 2019. doi: 10.5281/zenodo.2562111.

[3] Jim Garrison, ‘Qiskit-Extensions/circuit-knitting-toolbox: Circuit Knitting Toolbox 0.6.0’. Zenodo, Feb. 12, 2024. doi: 10.5281/zenodo.10651875.

[4] Jin-Guo Liu and Lei Wang. Differen- tiable learning of quantum circuit born ma- chines. Physical Review A, 98(6):062324, 2018. DOI: 10.1103/PhysRevA.98.062324.

[5] L. Brenner, C. Piveteau, and D. Sutter, “Optimal wire cutting with classical communication,” arXiv.org, Feb. 07, 2023. https://arxiv.org/abs/2302.03366

[6] Mehmet Akturk, “Diabetes Dataset,” Kaggle.com, 2020. https://www.kaggle.com/datasets/mathchi/diabetes-data-set

[7] M. Beisel, J. Barzen, M. Bechtold, F. Leymann, F. Truger, and B. Weder, “QuantME4VQA: Modeling and Executing Variational Quantum Algorithms Using Workflows,” Proceedings of the 13th International Conference on Cloud Computing and Services Science, 2023, doi: https://doi.org/10.5220/0011997500003488.

[8] Vojtech Havlíček, Antonio D Córcoles, Kris- tan Temme, Aram W Harrow, Abhinav Kandala, Jerry M Chow, and Jay M Gam- betta. Supervised learning with quantum- enhanced feature spaces. Nature, 567(7747): 209–212, 2019. DOI: 10.1038/s41586-019- 0980-2.

[9] Qiskit, author = Qiskit contributors, title = Qiskit: An Open-source Framework for Quantum Computing, year = 2023, doi = 10.5281/zenodo.2573505