Generative Flow Networks with Parameterized Quantum Circuits

Eli J. Laird
Department of Computer Science
Southern Methodist University
Dallas, Texas
ejlaird@smu.edu

Mitchell A. Thornton
Lyle School of Engineering
Southern Methodist University
Dallas, Texas
mitch@lyle.smu.edu

***Abstract*—The convergence of quantum and classical computing, particularly in the realm of quantum machine learning, has gained substantial attention. Hybrid quantum algorithms, merging the strengths of both computing paradigms, offer a promising avenue, especially in the era of NISQ (noisy intermediate-scale quantum) computing. Leveraging these hybrid approaches allows researchers to employ quantum algorithms on existing quantum processors while harnessing classical methods to handle complex computational challenges. A core component of these hybrid systems is the Parameterized Quantum Circuit Element (PQCE). These circuits, defined by parameterized unitary operations, possess exceptional expressiveness, effectively modeling intricate distributions. This study introduces a novel application of Parameterized Quantum Circuits in Generative Flow Networks for object generation tasks. This model demonstrates efficiency and quality on par with classical deep neural network-based approaches. Despite longer training times, the quantum model achieves comparable results with significantly fewer parameters (151), emphasizing its potential in scenarios prioritizing parameter efficiency.**

***Index Terms*—quantum-classical systems, generative flow networks, parameterized quantum circuits**

I. Introduction

The convergence of quantum and classical systems has emerged as a focal point in the realm of quantum computing, particularly in the context of quantum machine learning. Hybrid quantum algorithms, adeptly combining the strengths of quantum and classical computing, have become instrumental in achieving enhanced computational performance. This synergy proves especially crucial in the current NISQ (noisy intermediate-scale quantum) era, where fully error-corrected large-scale quantum computers remain elusive. Hybrid approaches empower researchers to deploy quantum algorithms on existing quantum processors, strategically utilizing classical methods to address computationally challenging aspects, such as parameter optimization.

A prominent archetype of hybrid quantum-classical systems is the Parameterized Quantum Circuit [1], [2]. Throughout this text we will refer to these circuits as Parameterized Quantum Circuit Elements (PQCEs) to avoid confusion with the popular acronym referring to "Post-Quantum Cryptography (PQC)". These circuits feature unitary operations that incorporate parameters to define rotations, typically constructed from single-qubit rotation gates. The flexibility of PQCEs to adjust rotation degrees through parameter control endows them with remarkable expressiveness [3], enabling effective modeling of intricate distributions. While these circuits transform quantum states using parameters, the optimization of these parameters takes place on classical computers. Analogous to the optimization of parameters in deep neural networks using task-specific loss functions for classification, PQCEs have demonstrated significant success across a spectrum of machine learning applications, encompassing supervised learning [1], [4], [5], generative modeling [1], [2], [6], and reinforcement learning [7]–[9].

One notable domain where PQCEs have outperformed classical counterparts is in reinforcement learning. For instance, [7] introduced a Softmax-PQCE as a policy network, showcasing comparable performance in various and superior results in select reinforcement learning environments compared to classical deep neural network-based models. Additionally, [8] and [9] expanded the applications of PQCEs to include Q-learning, noting substantial improvements in agent efficiency within RL environments.

This paper sets out to adapt the reinforcement learning applications of parameterized quantum circuits to Generative Flow Networks (GFlowNets). GFlowNets, inspired by reinforcement learning methodologies for sequential object generation [10]–[12], iteratively construct objects by sampling actions or building blocks, mirroring the interaction of a reinforcement agent within an environment. However, unlike RL agents that sample actions to maximize rewards, GFlowNets sample actions with probabilities proportional to rewards. This distinction positions GFlowNets to harness the capability of PQCEs in modeling complex action environments, extending the advantages witnessed in the reinforcement learning applications of PQCEs.

Our key contributions in this paper are as follows:

(i) Introduced a Generative Flow Network implemented through a Parameterized Quantum Circuit (GFlowNet-PQCE).

Code available: https://github.com/elilaird/quantum-gflownets

(ii) Implemented the proposed model in a simple object generation task, with quantum noise included and excluded.
(iii) Analyzed the efficiency and quality of the model's object generation capabilities in comparison to a classical deep neural network-based approach.

II. Related Work

A. Parameterized Quantum Circuit Elements

Parameterized Quantum Circuit Elements (PQCEs) constitute a category of quantum circuits characterized by tunable unitary operations, providing a flexible framework for implementing quantum algorithms. These circuits serve as a foundational element in hybrid quantum-classical systems, where classical systems play a pivotal role in optimizing the parameters defining the unitary rotation operators. In the context of machine learning, PQCEs have demonstrated extensive utility, particularly in applications such as supervised learning [1], [4], [5], generative modeling [1], [2], [6], and reinforcement learning [7]–[9]. While successful in these domains, these circuits have yet to be applied to sequential object creation, a task that requires generating complex combinatorial structures through step-by-step construction processes.

B. Generative Flow Networks

Generative Flow Networks (GFlowNets) constitute a category of generative models that operate sequentially, generating objects by iteratively sampling a policy to append new features to an incomplete object [10] [12]. What sets GFlowNets apart is their distinctive approach of generating objects with a probability distribution proportional to a reward function. This characteristic allows for the network to learn multiple modes of the object distribution and thus promote diversity of object characteristics. In contrast, reinforcement learning policies aim to maximize rewards from a single mode from the distribution, limiting their exploration of the wider object distribution.

A key principle underlying GFlowNets is the maintenance of equilibrium in probabilities (flows) entering and leaving a state. Bengio et al. [12] show that when a policy adheres to this condition, referred to as the Flow Matching Condition, the objects it generates exhibit probabilities proportional to the reward. GFlowNets achieve this condition through training with Trajectory Balance Loss [11], which ensures a balance between the incoming and outgoing flows within specific states. For a deeper discussion of the Flow Matching Condition and Trajectory Balance Loss, we refer the reader to the original works [10]–[12].

III. Proposed Approach

A. Circuit Design

The design of the parameterized circuit in this paper aligns with established practices in PQCE-based reinforcement learning, as seen in prior works such as [7]. We define a parameterized unitary operator $U(s, \theta)$ that takes an n-qubit state and trainable parameters θ. Building on the approach in [5], [7], [13], we partition the circuit into L layers, where each layer consists of a variational layer and an encoding layer, as illustrated in Figure 1.

The variational layer incorporates single-qubit R_z, R_x, and R_y rotations, as well as Controlled-Z gates. The gates within the variational layer are parameterized by a learnable matrix Θ, which governs the rotations. On the other hand, the encoding layer comprises single-qubit R_x rotations and is responsible for encoding the classical reinforcement learning state into a quantum state. While the variational layer is parameterized by the Θ matrix, the encoding layer uses a learnable scaling matrix Λ in accordance with [7]. These alternating layers closely resemble the layers of a deep neural network.

B. Representation of the Policy

1) General Softmax Policy: Given the inherently probabilistic nature of quantum states, projective measurements into these states serve as our GFlowNet policy. To represent the various actions our policy can produce, we partition the Hilbert space into A subspaces, each associated with a corresponding measurement P_a. This measurement P_a projects the quantum state into the subspace representing the action a.

Leveraging the projections from the measurement P_a, we can approximate the expected state $\langle S \rangle$ by measuring the state T times. The expected state $\langle S \rangle$ serves as the action sampled from the policy. In the context of learning a policy in reinforcement learning and GFlowNets, the distribution of the policy is conventionally adjusted during training to transition from an exploratory strategy to an exploitation strategy.

To facilitate this transition, [7] introduced Softmax-PQCE, which involves passing the expected state through a softmax function parameterized by β, as depicted in Equation 1. By updating the β parameter, the model can shift from an exploration strategy (large β) to an exploitation strategy (small β).

$$\pi_{\theta,\lambda}(a|s) = \frac{e^{\beta \langle S_a \rangle_{s,\theta}}}{\sum_{a'} e^{\beta \langle S_{a'} \rangle_{s,\theta}}} \tag{1}$$

2) Forward and Backward Policies: GFlowNets employ two distinct policies for action sampling: a forward policy P_f and a backward policy P_b. The forward policy reflects the probability of generating the state S_{t+1} given state S_t, denoted as $P_B(S_{t+1}|S_t)$. Conversely, the backward policy represents the probability of sampling a parent state S_t given the child state S_{t+1}, expressed as $P_B(S_t|S_{t+1})$. In simpler terms, the forward policy is the constructive policy responsible for building the object at S_{t+1} by taking action a, while the backward policy deconstructs the object by removing action a from the object at S_{t+1}.

To model the forward and backward policies, we employ unnormalized action logits, departing from the normalized softmax probabilities utilized in [7]. Subsequently, these logits undergo processing through a linear layer with $A \times 2$ output neurons, where the first A neurons represent the forward policy logits and the last A neurons represent the backward policy logits, as illustrated in Equation 2.

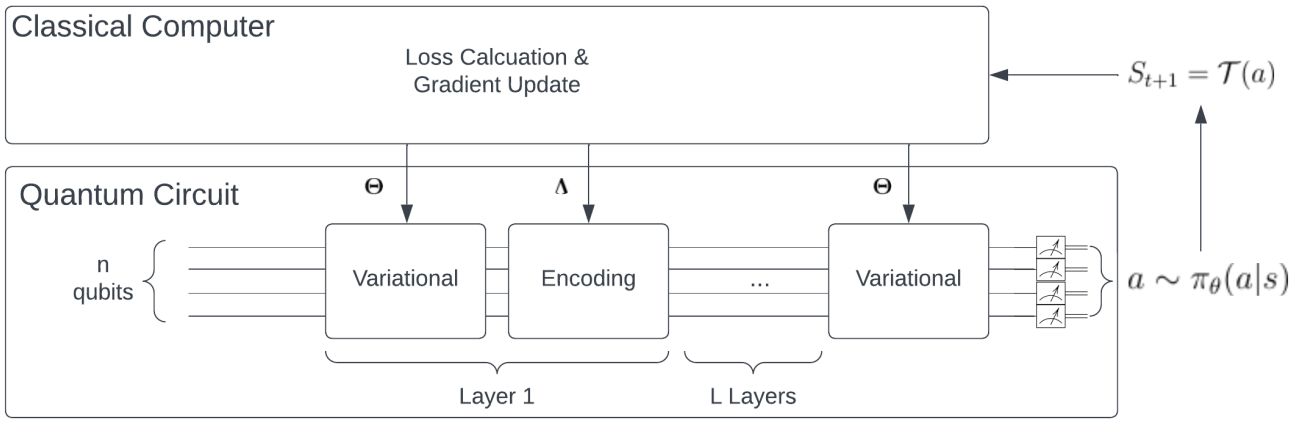

Fig. 1: Parameterized Quantum Circuit with alternating variational and encoding layers and classical computer (top) for loss and gradient calculations. Variational layers are parameterized by the learnable $\mathbf{\Theta}$ matrix and encoding layers are parameterized by the scaling matrix $\mathbf{\Lambda}$.

$$\begin{aligned} Logits &= Linear(e^{\beta \langle S_a \rangle_{s,\theta}}) \\ P_F(S_{t+1}|S_t) &= Logits[0 : A-1] \\ P_B(S_t|S_{t+1}) &= Logits[A : A \times 2] \end{aligned} \tag{2}$$

C. Reward

GFlowNets adopt a strategy of sampling actions with probabilities proportional to the associated reward. In the context of a GFlowNet, the reward serves to shape the modes of a probability distribution over actions. Unlike traditional reinforcement learning settings where the objective is to maximize the overall reward, GFlowNets focus on sampling from the modes of the distribution.

One distinctive feature of GFlowNets is their ability to sample non-maximum but high-reward actions, fostering a diversification in actions. The reward in a GFlowNet is task-specific, tailored to the objectives of the particular application. In this paper, our exploration involves employing a GFlowNet to iteratively construct objects belonging to a specific class.

D. Trajectory Balance Loss

In training the policy as delineated in Section III-B, we employ the Trajectory Balance Loss function introduced in [11] for optimizing GFlowNets. This loss function is designed to incentivize the policy to sample actions with a probability proportional to the reward. The computation of the Trajectory Balance Loss is carried out for a completed object at state S after its construction through the sampling of the policy $\pi_{\theta,\lambda}(a|s)$.

We define the sequence of intermediate objects at S_t as a trajectory τ. The loss for a state S constructed over a trajectory of τ samples can be computed using the following equation:

$$L_{TB}(S) = \left(\log \frac{Z \prod_t^\tau P_F(S_{t+1}|S_t)}{R(S_t) \prod_t^\tau P_B(S_t|S_{t+1})} \right)^2 \tag{3}$$

, where P_F and P_B are the forward and backward policies, Z is a trainable scaling parameter, and $R(\cdot)$ is the reward function.

E. Training Procedure

We train the forward and backward policies by minimizing the trajectory balance loss using Algorithm 1. This algorithm involves a transition function $\mathcal{T}(a)$ that produces a new state S_{t+1} given a sampled action a, the reward R, and the policy $\pi_{\theta,\lambda}$. The training process begins by initializing the state to a predefined value S_0, tailored to the specific application. For instance, in generating a smiley face, the initial state would be an empty circle. The initial state is then added to the trajectory τ, which serves to track the intermediate states S_t throughout the process.

Subsequently, the generation of a complete object S takes place by sampling actions from the policy $\pi_{\theta,\lambda}$, executing these actions using the transition function $\mathcal{T}$, and recording intermediate states in the trajectory τ until a predetermined maximum number of actions is reached. The total reward for each state in the trajectory is then summed and used to calculate the loss.

IV. Evaluation

To assess the performance of the GFlowNet-PQCE policy in object generation, we apply the policy to a simple object generation task involving the creation of smiley faces, introduced in the popular GFlowNet tutorial [14]. In our experiments, we compare the quantum GFlowNet with a classical deep neural network-based approach for generating these objects. While the primary focus of this paper is to demonstrate the feasibility of generating compositional objects with a quantum GFlowNet, we conduct a comparative analysis with the classical GFlowNet, evaluating efficiency and the ability to generate the specified class.

Algorithm 1 GFlowNet-PQCE Policy Training Loop

Input: policy $\pi_{\theta,\lambda}$, transition function $\mathcal{T}$, $EPOCHS$, $R(\cdot)$, $MAX\_ACTIONS$

for epoch in $EPOCHS$ **do**
  $actions\_taken \leftarrow 0$
  S_0 ▷ Initialize state
  $\tau \leftarrow \emptyset + \{S_0\}$ ▷ Initialize trajectory
  repeat
    $a \leftarrow \pi_{\theta,\lambda}$ ▷ Sample action
    $S_{t+1} \leftarrow \mathcal{T}(a)$ ▷ Take action
    $\tau \leftarrow \tau + \{S_{t+1}\}$ ▷ Append S_{t+1} to trajectory
    $actions\_taken++$
  until $actions > MAX\_ACTIONS$
  $\theta \leftarrow \theta - \eta \nabla Loss_{TB}(R(S), log_Z, \tau)$ ▷ Update θ
end for

Efficiency is assessed by measuring the total time required for training over *50,000* epochs and the time required to generate a single object. The choice of *50,000* epochs aligns with the classical GFlowNet's convergence benchmark as outlined in [14].

The generation quality of a GFlowNet is defined by its capacity to generate objects of a specific class with a probability proportional to the target reward distribution. Given the limited number of possible actions for the smiley face generation task, we can precisely define the reward distribution T by setting rewards for smiley faces and frowny faces accordingly. Because there are only two classes, we can measure the generation quality by the absolute difference in the target number of objects of type 'smiley' T_s, and the generated objects of class 'smiley' G_s, as illustrated in Equation 4. Furthermore, we gauge the percentage of valid faces generated, where a valid face exhibits no conflicting features, such as simultaneous smiling and frowning.

$$Quality\ Disparity = \left| T_s - \frac{\sum_i^N f(O_i) == s}{N} \right| \quad (4)$$

where T_s is the target number for the class 'smiley', $f(\cdot)$ defines the class of the object O_i generated by the GFlowNet.

A. Smiley-Face Generation

This experiment aims to assess the capability of the proposed GFlowNet-PQCE in generating simple objects characterized by a limited number of possible actions. In this context, a complete object refers to a graphical face, which can be categorized as either 'smiley' or 'frowny'. Each face is generated with the following action space: {smile, frown, left eyebrow down, right eyebrow down, left eyebrow up, right eyebrow up}. Figure 2 provides a visual representation of a smiley and frowny face as illustrative examples.

B. Experiment Design

The experimental setup outlined in the previous section will be executed using the TensorFlow (TF) [15] and TensorFlow Quantum (TFQ) [16] Python packages. Leveraging TensorFlow enables the encapsulation of Parametrized Quantum Circuits (PQCE) and GFlowNets within custom Keras models, facilitating straightforward gradient tracking and updates. For the classical GFlowNets used as a comparison, PyTorch [17] will be employed. All experiments are run on an AMD EPYC 7763 processor with 128 CPU cores.

Fig. 2: Left: example of a 'smiley' face. Right: example of an 'frowny' face from [14].

The parameterized quantum circuits, implemented in TFQ, will undergo simulation using both a noiseless simulator and a noisy simulator. The first simulator, an ideal simulator embedded within the *ControlledPQCE* TFQ class, abstains from modeling noise within the quantum circuits. In contrast, the second simulator, integrated into the *NoisyControlledPQCE* TFQ class, introduces noise to the simulator measurements via Monte Carlo Sampling. Both classes leverage the Cirq simulator, integrated into TFQ, which automatically executes simulations when invoked through the Keras wrapper.

We define the target reward distribution to follow a 2:1 smiley-to-frowny ratio between classes, adhering to the approach presented in [14]. To strengthen the reward signal, the rewards were subsequently scaled by a factor of 2 to amplify the reward signal while preserving the established ratio, resulting in a reward of 4, 2, and 0 for the smiley, frowny, and invalid classes respectively.

In tackling the smiley face generation task, the quantum GFlowNets used 6 qubits, allocating one for each possible action. The GFlowNet-PQCE architecture used 6 layers, with a variational layer parameterized by Θ as the initial layer. Subsequent layers consist of another variational layer and an encoding layer parameterized by Λ. Each variational layer used 3 θ parameters per qubit, responsible for the R_x, R_y, and R_z rotations. Meanwhile, each encoding layer required a singular scaling parameter λ per qubit.

Conversely, the classical GFlowNet adopted a simple two-layer multilayer perceptron design with 512 hidden neurons. As outlined in [11], [14], both GFlowNet variants learned a Z parameter to facilitate trajectory balance loss optimization. In total, the GFlowNet-PQCE required only 151 parameters, compared to the 9741 for the classical GFlowNet. A comprehensive summary of all parameters is provided in the Table I.

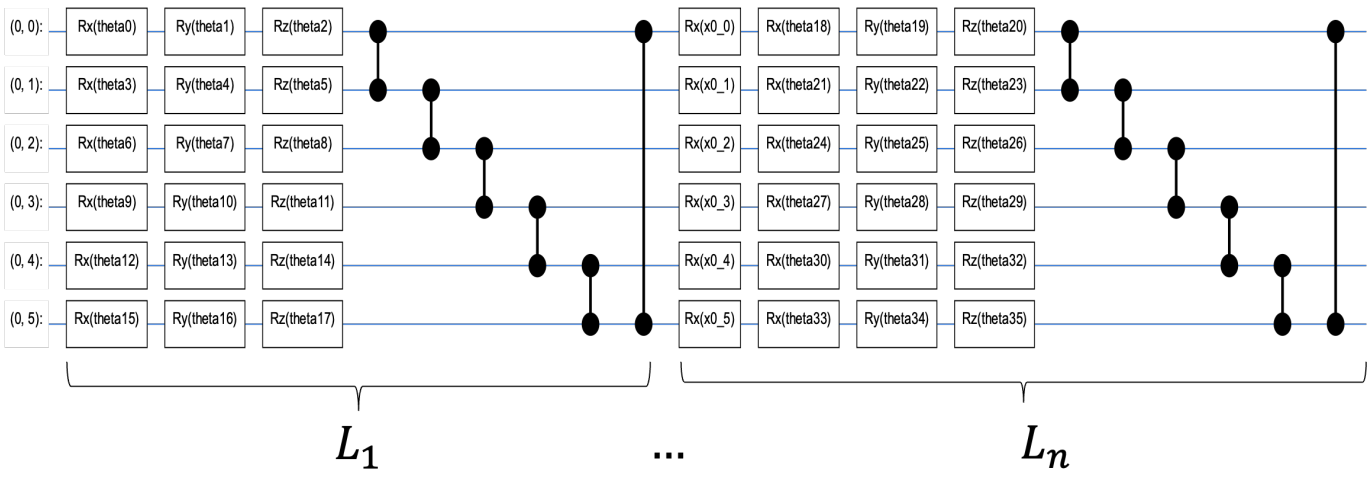

Fig. 3: In depth circuit diagram for the Parameterized Quantum Circuit used in the smiley face generation task.

TABLE I: Parameter Counts for Quantum and Classical GFlowNets.

Parameter	Quantum	Classical
Lambda (Λ)	30	–
Theta (Θ)	108	–
Linear (Input)	–	3072
Linear (Hidden 1)	–	512
Linear (Hidden 2)	–	6144
Linear (Output)	12	12
Log Z	1	1
Total Parameters	**151**	**9741**

V. Results

A. Smiley Face Generation Quantitative Results

1) Efficiency: The quantitative outcomes from the smiley face experiment are shown in Tables II and III. Examining the efficiency results in Table II, we observe a notable difference in training durations between the GFlowNet-PQCE and the classical GFlowNet over *50,000* epochs.

The classical GFlowNet completed training in a mere 4 minutes compared to the significantly longer 6 hours and 52 minutes for the noisy GFlowNet-PQCE and 51 minutes for the noiseless GFlowNet-PQCE. In summary, the classical GFlowNet was approximately 12.75 times faster than the noiseless GFlowNet-PQCE and 103 times faster than the noisy GFlowNet-PQCE. When observing the average time required to generate a single object we find that the classical approach again was the fastest with an average generation time of *0.00268* seconds. In comparison, the noiseless GFlowNet-PQCE and the noisy GFlowNet-PQCE exhibited longer average generation times of *0.05748* seconds and *0.08018* seconds, or 21.5 and 30 times slower than their classical counterpart.

Several optimization strategies could improve the training efficiency of GFlowNet-PQCEs. First, joint optimization of circuit architecture and parameters could deliver improvements with minimal computational overhead. For noisy quantum circuits, applying error mitigation techniques such as zero-noise extrapolation would reduce the impact of hardware noise. Additionally, implementing hardware-aware training that accounts for specific device characteristics could substantially reduce training times while preserving the quantum models' advantage in parameter efficiency. We encourage exploration of these optimization strategies for future work.

TABLE II: Training and Generation Times for Classical and Quantum GFlowNet-PQCEs for 50k epochs. Times Exclude Gradient Calculations and Logistical Logging Operations.

Model	Training Time (hh:mm:ss)	Avg. Generation Time (s)
Quantum Noisy	06:52:59	0.08018
Quantum Noiseless	00:51:58	0.05748
Classical	00:04:09	0.00268

2) Generation Quality: The results measuring object generation quality are detailed in Table III. Our target reward distribution dictates a 2:1 smiley-to-frowny face ratio, translating to a 66 smiley faces within a set of 100 samples.

In terms of quality, both noiseless GFlowNet-PQCE and the classical GFlowNet show comparable ability in matching the reward distribution. The classical GFlowNet yielded 72 smiley faces out of 100, deviating by +6 faces from the desired 66, while the noiseless GFlowNet-PQCE generated 61 smiley faces out of 100, showing a deviation of -5 faces. Notably, the noisy GFlowNet-PQCE generated only smiley faces, indicating that it collapsed to maximizing the reward instead of modeling the desired target distribution. A deeper investigation into the causes of this distribution collapse is reserved for future works.

It is important to highlight that all three GFlowNets, both classical and quantum, successfully generated 100% valid faces out of the 100 sampled, underscoring the efficacy of the models in maintaining validity throughout the generation process. Avenues for future exploration include investigating

the impact of a larger action space and non-binary reward distributions.

TABLE III: Quality Metrics for Quantum and Classical GFlowNets in Smiley Face Generation.

Model	Quality Disparity ↓	Valid Faces (%) ↑
Quantum Noisy	44	100
Quantum Noiseless	5	100
Classical	6	100

B. Parameter Efficiency of Quantum vs Classical GFlowNets

In preceding sections, we demonstrated the ability of quantum GFlowNet-PQCEs to minimize a trajectory loss function, enabling the generation of objects with quality comparable to classical GFlowNets. A notable distinction between the quantum and classical approaches lies in the number of trainable parameters required to learn the reward distribution. Specifically, both the noisy and noiseless versions of quantum GFlowNet-PQCEs require a mere *151* trainable parameters, as opposed to the *9741* parameters demanded by the classical GFlowNet, shown in Table I. This contrast in parameter count suggests that PQCEs might be preferable in operational scenarios where a minimal parameter footprint is crucial, despite the considerably longer training time required.

VI. Conclusion

This work demonstrates a proof of concept application of Parameterized Quantum Circuits (PQCEs) within the domain of Generative Flow Networks (GFlowNets). The proposed GFlowNet-PQCE model exhibits promising results in the generation of simple objects, such as smiley faces, showcasing its parameter efficiency and quality in comparison to classical deep neural network-based approaches. We demonstrate that GFlowNet-PQCEs exhibit comparable performance in matching a reward distribution in the simple object generation task while also learning to generate only valid faces. Despite the longer training times of GFlowNet-PQCEs, they require fewer trainable parameters than their classical counterparts, highlighting their advantage in compute-constrained environments. Overall, this study provides a first step in employing parameterized quantum circuits to implement GFlowNets for object generation tasks, paving the way for further research into quantum-classical hybrid models and their applications in generative machine learning.

Acknowledgments

This work made use of computing resources provided by the O'Donnell Data Science and Research Computing Institute at Southern Methodist University.

References

[1] M. Benedetti, E. Lloyd, S. H. Sack, and M. Fiorentini, "Parameterized quantum circuits as machine learning models," *Quantum Science and Technology*, vol. 4, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:189999815

[2] D. Zhu, N. M. Linke, M. Benedetti, K. A. Landsman, N. H. Nguyen, C. H. Alderete, A. Perdomo-Ortiz, N. Korda, A. Garfoot, C. Brecque, L. Egan, O. Perdomo, and C. Monroe, "Training of quantum circuits on a hybrid quantum computer," *Science Advances*, vol. 5, no. 10, p. eaaw9918, 2019. [Online]. Available: https://www.science.org/doi/abs/10.1126/sciadv.aaw9918

[3] Y. Du, M.-H. Hsieh, T. Liu, and D. Tao, "Expressive power of parametrized quantum circuits," *Physical Review Research*, vol. 2, no. 3, jul 2020. [Online]. Available: https://doi.org/10.1103/2Fphysrevresearch.2.033125

[4] V. Havlíček, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta, "Supervised learning with quantum-enhanced feature spaces," *Nature*, vol. 567, no. 7747, pp. 209–212, mar 2019. [Online]. Available: https://doi.org/10.1038/s41586-019-0980-2

[5] M. Schuld, R. Sweke, and J. J. Meyer, "Effect of data encoding on the expressive power of variational quantum-machine-learning models," *Physical Review A*, vol. 103, no. 3, mar 2021. [Online]. Available: https://doi.org/10.1103/2Fphysreva.103.032430

[6] J.-G. Liu and L. Wang, "Differentiable learning of quantum circuit born machines," *Phys. Rev. A*, vol. 98, p. 062324, Dec 2018. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.98.062324

[7] S. Jerbi, C. Gyurik, S. Marshall, H. J. Briegel, and V. Dunjko, "Parametrized quantum policies for reinforcement learning," in *Neural Information Processing Systems*, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:244843259

[8] A. Skolik, S. Jerbi, and V. Dunjko, "Quantum agents in the gym: a variational quantum algorithm for deep q-learning," *Quantum*, vol. 6, p. 720, may 2022. [Online]. Available: https://doi.org/10.22331/q-2022-05-24-720

[9] S. Y.-C. Chen, C.-H. H. Yang, J. Qi, P.-Y. Chen, X. Ma, and H.-S. Goan, "Variational quantum circuits for deep reinforcement learning," *IEEE Access*, vol. 8, pp. 141 007–141 024, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:195767325

[10] E. Bengio, M. Jain, M. Korablyov, D. Precup, and Y. Bengio, "Flow Network based Generative Models for Non-Iterative Diverse Candidate Generation," Nov. 2021. [Online]. Available: http://arxiv.org/abs/2106.04399

[11] N. Malkin, M. Jain, E. Bengio, C. Sun, and Y. Bengio, "Trajectory balance: Improved credit assignment in GFlowNets," Oct. 2022, arXiv:2201.13259 [cs, stat]. [Online]. Available: http://arxiv.org/abs/2201.13259

[12] Y. Bengio, S. Lahlou, T. Deleu, E. J. Hu, M. Tiwari, and E. Bengio, "GFlowNet Foundations," Aug. 2022, arXiv:2111.09266 [cs, stat].

[13] A. Pérez-Salinas, A. Cervera-Lierta, E. Gil-Fuster, and J. I. Latorre, "Data re-uploading for a universal quantum classifier," *Quantum*, vol. 4, p. 226, feb 2020. [Online]. Available: https://doi.org/10.22331/2Fq-2020-02-06-226

[14] B. Emmanuel, "GFlowNet Tutorial." [Online]. Available: https://colab.research.google.com/drive/1fUMwgu2OhYpQagpzU5mhe9_Esib3Q2VR

[15] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[16] M. Broughton, G. Verdon, T. McCourt, A. J. Martinez, J. H. Yoo, S. V. Isakov, P. Massey, R. Halavati, M. Y. Niu, A. Zlokapa, E. Peters, O. Lockwood, A. Skolik, S. Jerbi, V. Dunjko, M. Leib, M. Streif, D. V. Dollen, H. Chen, S. Cao, R. Wiersema, H.-Y. Huang, J. R. McClean, R. Babbush, S. Boixo, D. Bacon, A. K. Ho, H. Neven, and M. Mohseni, "Tensorflow quantum: A software framework for quantum machine learning," 2021.

[17] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019.