Master Thesis

Evolutionary Optimization for Hybrid Quantum Image Generation GAN

Bianca Matilde Massacci
i6261625

Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science of Data Science for Decision Making
at the Department of Advanced Computing Sciences
of the Maastricht University

Thesis Committee:

Dr. Menica Dibenedetto
Dr. Mirela Popa

Maastricht University
Faculty of Science and Engineering
Department of Advanced Computing Sciences

July 3, 2024

Abstract

Quantum Machine Learning (QML) is considered one of the most promising applications of quantum computing, offering significant advantages over classical methods. Among QML applications, quantum generative adversarial networks (QGANs) have shown potential for exponential improvement over classical GANs. Most current research has focused on small, proof-of-concept applications, with one study that demonstrated the ability to generate small gray-scale images with comparable quality but significantly fewer parameters than the state-of-the-art classical nets, utilizing an application-agnostic, hardware efficient parameterized ansatz.

This work investigates the effectiveness of a quantum mutation-based evolutionary search strategy for identifying better-performing ansätze, evaluated based on image quality and the number of required parameters. The ansätze are used as generators in a hybrid quantum-classical GAN for generating 28×28 grayscale MNIST images of digits 0 and 1. The results highlight the feasibility of using architectures that differ from the standard, hardware-efficient ansatz to generate images of comparable quality to the current state-of-the-art, and the evolutionary algorithm's potential in discovering superior configurations. However, further research is necessary to fully understand the extent and scope of the improvements that the evolutionary algorithm can bring.

Keywords: quantum machine learning; generative adversarial networks; evolutionary algorithm; ansatz search; hybrid quantum-classical system; quantum GAN.

Contents

1 Introduction . . . 3
2 Background and Related Work . . . 5
2.1 Generative Adversarial Networks . . . 5
2.2 From Classical to Quantum . . . 7
3 Methods . . . 11
3.1 Evolutionary Algorithm . . . 11
3.2 Quantum GAN for Image Generation . . . 15
4 Experimental Setup . . . 20
4.1 Data and Computational Framework . . . 20
4.2 Evaluation metrics . . . 21
4.3 Quality of the output . . . 21
4.4 Computational Efficiency . . . 23
5 Results . . . 24
5.1 Stage 1: Patches and Layers . . . 24
5.2 Stage 2: Number of Ancillary Qubits . . . 28
5.3 Stage 3: Action Weights . . . 28
5.4 Stage 4: Multi-Action Probability . . . 31
5.5 Stage 5: increased generations and epochs . . . 31
5.6 Stage 6: Best architecture with increased layers . . . 33
5.7 Sample Efficiency . . . 33
6 Discussion . . . 36
6.1 Number of layers . . . 37
6.2 Number of patches and qubits . . . 37
6.3 Action weights . . . 39
6.4 Multi-action Probability . . . 40
6.5 Evaluation Metrics . . . 40
6.6 Ansatz architecture . . . 42
6.7 Final results and benchmarking . . . 43
7 Conclusions and Future Work . . . 49
8 Appendix . . . 51
8.1 Evolutionary Algorithm: Mutation Procedure . . . 51
8.2 Additional Experiments Results . . . 52
8.3 Hyperparameters . . . 53

1 Introduction

Generative Adversarial Networks (GANs) are one of the most popular and successful machine learning architectures for generative applications [1]. Their ability to generate high-quality data across a wide variety of tasks is founded on the adversarial learning strategy on which they are based. Their architecture consists of two separate networks, the generator and the discriminator, that, having opposed objectives, through their competition work in tandem to achieve the ultimate goal of generating realistic, high-quality data. The task of the generator, as its name suggests, is to generate data closely resembling real data. Meanwhile, the task of the discriminator is to accurately distinguish between real data and data generated by its counterpart. This competition drives both networks to continually evolve - with on the one hand the discriminator improving its ability to distinguish real data from generated data, and on the other hand, the generator producing increasingly better data to continue fooling the progressively improving discriminator [2]. Due to their generative capabilities, GANs are one of the most employed generative models across a wide range of fields [3][4], notably in image processing for tasks such as image and video generation [5][6] and super-resolution [7], as well as in other fields, like the medical and chemical sector [1], e.g., for drug discovery [8] and DNA generation and design [9].

The well documented good performance that has made GANs a popular choice for generative applications relies of course on the characteristic adversarial architecture, but it is also the result, as in all deep learning models, of the complexity and high dimensionality of the two networks, a necessary prerequisite for capturing the complex distribution of real datasets. The large number of parameters, however, also carries a significant demand of computing resources. With different and progressively more complex variations to the original architecture [2] proposed in the last decade, the level of accuracy of the models has only been improving, but so have the computational power and hardware efficiency of the networks. This trend, as predicted by Moore's Law [10], has so far been sustained by the parallel exponential improvements in processors production technology. However, as we approach the physical limits of shrinking transistors and other microchips components, this growth is expected to soon come to a halt, a concerning issue, as it suggests that we are poised to enter an era where hardware advancements cannot keep pace with the growth in models' complexity and data volume. A potential solution that has been proposed to tackle this challenge is a complete shift in computation domain, from classical to quantum [11].

Quantum computing, or computation performed on quantum machines [12], and particularly its sub-field quantum machine learning[1] are considered one of the most promising areas for quantum to bring effective improvements to the classical methods [14], including Quantum GANs. The advantage of implementing GANs as quantum architectures lies in the fact that, by exploiting quantum properties such as superposition and entanglement, quantum networks can be more resource-efficient and therefore well suited to handle complex tasks, reaching comparable performance to deep classical architectures with a significant reduction in the number of parameters and required computational resources [15].

Since their introduction in 2018 [16][17], several variations of quantum GANs have been explored. Given the early stage of quantum computing, a field developing within the current constraints imposed by the Noisy Intermediate-Scale Quantum (NISQ) era, most of these works are exploratory efforts with preliminary results, and have focused on simple applications [18]. There are still many aspects to be explored, and issues to be solved, such as how to design efficient and application-tailored ansätze for real data generation (including images), how and if to integrate quantum and classical component in a hybrid net, how to properly scale up the

[1] I refer to Schuld and Petruccione's defintion: '[Quantum Machine Learning is] machine learning with quantum computers, or quantum-assisted machine learning' [13].

nets for dealing with larger and more complex datasets, for what applications are quantum nets truly advantageous, and more. Nonetheless, the theoretically proven advantage that quantum may bring to the classical architecture makes this an area worthy of further exploration.

With regards to the challenge of generating classical data, specifically real images, using quantum nets, Huang et al. [19] introduced in 2021 the first quantum-based GAN capable of learning and generating 8×8 real-world digit images on a superconducting quantum processor. Their work was further expanded by Tsang et al. [20] who successfully trained a simulated hybrid quantum-classical network to generate 28×28 pixels grayscale images. Although these results are impressive, the quality of the images, though comparable, is visually inferior to those generated with fully classical methods, and the application is still restricted to small, simple datasets. This limitation is partly due to the limited resources available, and of course, to these being early exploratory works, but it might also be due to the fact that the proposed generator architecture consists of a general-purpose variational quantum circuit, which, while functional, may not be the most effective or efficient architecture for this specific task.

In an attempt to further explore the potential of variational circuits as generators in hybrid quantum GANs for image generation, in this feasibility study I use an evolutionary algorithm as an optimization technique to identify potentially more effective variational circuits for generating classical data, specifically grayscale digits images. The primary aim of this research is to verify whether alternative architectures to the standard hardware-efficient ansatz can be successfully trained for the task of image generation, and to determine whether the employed evolutionary algorithm can identify an ansatz capable of outperforming such architectures. Developing general methods to generate close-to-optimal ansätze for different applications is important because it takes a step further from standard, generalized hardware-efficient ansätze and instead explores the potential of using parameterized quantum circuits (PQCs) tailored to each specific application. The choice of evolutionary algorithm for ansatz optimization is driven by the complex and difficult nature of ansatz design [11]. An evolutionary algorithm offers a systematic and unbiased approach for exploring the space of possible solutions, bypassing some of the limitations associated with manual architecture design, and thus potentially leading to finding superior configurations. To this end, I experiment with various architectural designs and conduct a hyperparameter search on the best-performing configurations to optimize the evolutionary algorithm.

The study findings demonstrate that the algorithm has significant potential for identifying optimal ansatz architectures, producing circuits with a comparable number of parameters and output quality to current state-of-the-art architectures. However, further research is needed to fully understand the role of various hyperparameters in optimizing the evolutionary search for this specific application and to refine the evolutionary strategy itself, essential to enhance the robustness and reliability of the proposed algorithm.

This thesis is structured as follows: Section 2 contains an overview of the foundational concepts behind this research and related work, including an overview of Generative Adversarial Networks and Quantum Computing. Section 3 provides a detailed description of the employed evolutionary algorithm for ansatz search, and hybrid quantum classical GAN for image generation. Section 4 describes the experimental methods and set-up, including used dataset and metrics. Section 5 contains an overview of the experimental results, divided in 6 separate experimental stages, each focusing on investigating and optimizing a different aspect of the evolutionary GAN pipeline. Section 6 discusses and analyzes key findings, including how and which hyperparameters affect performance, and why, as well as limitations of the study. Finally, section 7 summarizes the work, lists the key takeaways, and suggests potential areas of improvement and directions for future research.

2 Background and Related Work

This section gives an overview of the foundational concepts and notions behind this work, including a brief overview of Generative Adversarial Networks (GANs) and their variants, key principles and notions of quantum computing and quantum machine learning, and prior work in the field of quantum GANs for image generation.

2.1 Generative Adversarial Networks

Generative Adversarial Networks (GANs), introduced by Goodfellow et al. in 2014 [2], have emerged as a powerful deep learning approach capable of learning complex representations without the need for large amounts of annotated training data. As such, they have become a popular tool for semi-supervised and unsupervised learning. Within the field of image and signal processing, some common applications for GANs include image classification and regression, image generation, image-to-image translation (e.g., style transfer, colorization), and super-resolution [3]. Such a variety in applications, one of GANs strengths, stems from their distinctive architecture, characterized by an adversarial learning strategy based on two functions, typically trainable deep neural networks, that are put in competition against each other by having opposite objective functions. On the one hand is the Generator, G, tasked with capturing the distribution of the real data and generating samples from the learned distribution, such that the generated samples are as similar as possible to the real ones. On the other hand is the Discriminator D, which, as the name suggests, is tasked with differentiating whether data it is given is real or has been generated by G. A schematic illustration of a basic GAN is shown in Figure 1.

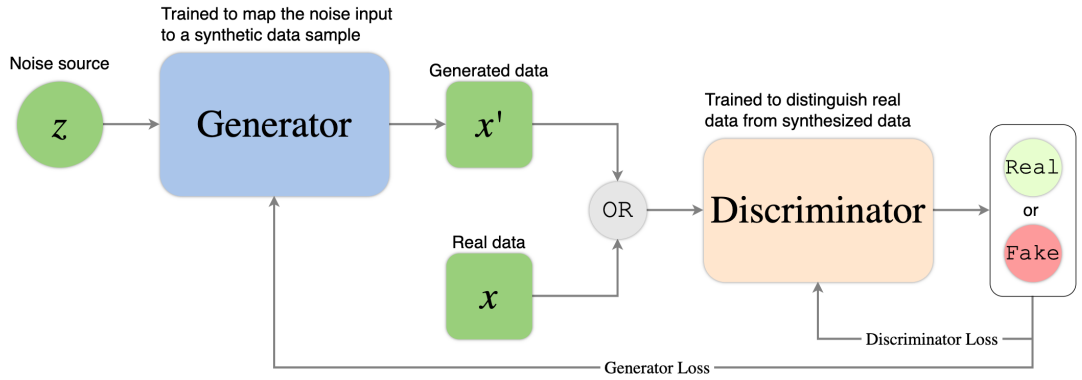


Figure 1: Schematic representation of the GAN framework, depicting the adversarial training process. The generator G maps a noise vector z to synthetic data x' , with the goal of replicating the distribution of the real data x. The discriminator D learns to differentiate between the generator's synthetic output and real data. The discriminator's predictions are used to update the weights of both networks through backpropagation. This competition between the two nets pushes them to constant improvement, ultimately leading to increased quality in the generated data.

The training of GANs is a min-max game where the discriminator and generator have opposite goals. The discriminator tries to maximize the probability of correctly differentiating between real and generated data, while the generator tries to minimize the same probability, by producing

data as realistic as possible, hence 'fooling' the discriminator. In the most classical variation of GANs [2] the objective function is defined as:

$$\min_G \max_D V(G, D) = \mathbb{E}_{x \sim P_r}[\log D(x)] + \mathbb{E}_{\tilde{x} \sim P_g}[\log(1 - D(G(z))] \tag{1}$$

Where $\mathbb{P}_r$ is the real data distribution, $\mathbb{P}_g$ is the model distribution (i.e., generated data distribution), defined by $\tilde{x} = G(z)$, $z \sim p(z)$, and z is a latent vector, sampled from a predefined distribution p, such as the uniform or normal distribution, used as input to the generator, that the generator maps to the output data $\tilde{x}$. $D(x)$ represents the score that D assigns to the input data x, with 1 meaning that D predicts x being real, and 0 meaning that D predicts x being generated. D wants to maximize eq. 1 because it wants to maximize both $\mathbb{E}_{x \sim P_r}[\log D(x)]$, the probability of D correctly labeling real data as real, and $\mathbb{E}_{\tilde{x} \sim P_g}[\log(1 - D(G(z)))]$, the probability of D correctly labeling generated data as non-real. On the other hand, G's goal is to minimize the same equation, by maximizing the probability of D classifying generated data as real. D and G are trained iteratively, meaning that training alternates between updating the discriminator's and the generator's parameters via backpropagation. In practice, when implementing the architecture each net is assigned a separate loss function to minimize:

$$L_D = -\left[y \cdot \log(D(x)) + (1 - y) \cdot \log(1 - D(G(z)))\right] \tag{2}$$

$$L_G = -\left[(1 - y) \cdot \log(D(G(z)))\right] \tag{3}$$

Where y represents the label ($y = 1$) for real data and ($y = 0$) for generated data.

The generator G reaches optimality when $\mathbb{P}_r = \mathbb{P}_g$. In this condition, an optimal discriminator D would be maximally confused and consistently assign an equal probability score of 0.5 to both real and generated data, as the two distributions would be indistinguishable. GANs' ultimate goal is for the generator to produce progressively better data samples. Although in practice equilibrium is rarely reached, GANs still prove successful at generating data of satisfactory quality in a variety of tasks.

A common issue with the original GAN implementation is that the distance function between real and generated data that the network is minimizing is typically non-continuous with respect to the generator's parameters. This can lead to vanishing gradients and difficulties in the training process [21]. As a solution [22] propose the Wasserstein GAN (WGAN).

In the traditional GAN, if the discriminator is trained to optimality before each generation update, minimizing the objective function is equivalent to minimizing the Jensen-Shannon divergence between the generated and real distribution [2]. In the WGAN, the net is instead trained to minimize an approximation of the Wasserstein-1 distance, or Earth Mover Distance (EMD). EMD is informally defined as the minimum cost of transporting mass to transform one distribution into the other. The advantage that Wasserstein-1 distance provides compared to other distance functions such as JS-divergence is that if the generator function is continuous and locally Lipschitz, the Wasserstein distance $W(P_r, P_g)$ will also be continuous and differentiable almost everywhere. This means that $W(P_r, P_g)$ does not require the real data distribution P_r and the generated data distribution P_g to have overlapping support to provide a meaningful distance measure. In other words, even when the generated data is initially very different from the real data, the Wasserstein distance can still effectively guide the training process. Moreover, the distance is continuous and differentiable, therefore providing useful gradients for the optimization of the parameters of the generator. This makes it a more sensible cost function for a Generative Adversarial Net than other distance functions such as Jensen-Shannon divergence,

Kullback-Leibler divergence, and Total Variation distance [23]. Thanks to this property, the Wasserstein GAN is characterized by a more robust training process and requires less meticulous balancing work between the generator and the discriminator, called critic in this variation, than the classical GAN. Additionally, it alleviates the need for highly careful design of the network architecture, and it mitigates mode dropping.

In order to enforce the Lipschitz constraint, WGAN relies on weight-clipping on the critic's weights, forcing them to lie within a predefined interval after each gradient update step. Because this approach has its own issues, as it can lead to undesired behavior and optimization difficulties, including capacity underuse of the critic by clipping its weights, and potentially still exploding and vanishing gradients if the threshold c is not properly tuned, Gulrajani et al. propose a further improvement, the Wasserstein GAN with Gradient Penalty (WGAN-GP), which instead enforces the Lipschitz constraint utilizing gradient penalty [21]. In other words, WGAN-GP introduces a penalization term for the norm of the discriminator's gradients w.r.t. its input. The theoretical motivation for this approach comes from the fact that a differentiable function is 1-Lipschtiz if and only if it has gradients with norm at most 1 everywhere. By forcing the gradient norm of the critic's output with respect to its input, this requirement is manually satisfied. This formulation provides more stable performance and stability compared to the WGAN formulation, when tested on a variety of architectures [21]. For these reasons, the critic from the WGAN-GP architecture is what I will use for the classical component of the presented algorithm.

2.2 From Classical to Quantum

Besides training instability, another challenge presented by GANs is their complexity and high computational demand, particularly for applications such as image generation that require high-dimensional training samples and high number of trainable parameters. Quantum GANs have been proposed as a potential path to address these challenges. These networks, which can be fully quantum or hybrid -incorporating classical and quantum components-, leverage on quantum properties to optimize training costs and time. For some applications, they have been shown to significantly decrease the number of parameters and training epochs needed to achieve comparable results to classical GANs, increasing computational efficiency up to an exponential degree [15].

This section provides a brief historical context of quantum computing and describes some foundational notions of the field. It also introduces a quantum GAN model for image generation, which is the basis of this work.

Quantum Computing

Since the latter half of the 20th century, the development of classical computers hardware has experienced an impressive growth in capabilities and computational power. In 1965, Gordon Moore outlined this trend in what has come to be known as the *Moore's law*, predicting the doubling of transistors on computer chips approximately every two years at a fixed cost [10], with computational capacity and speed growing at a similar rate [24]. This observation has held true to the present day. However, as electronic devices are becoming progressively smaller and reaching atomic scale, quantum effects threaten to interfere with their regular functioning, which would put a halt to the rate of growth described in Moore's law. Such development brings important socio-economic consequences, given that the growth of the information-technology industry has relied on this exponential improvement in performance and functionality of digital electronics [25]. Therefore, the industry and scientific community have been exploring solutions to address this issue. One of the proposed solutions is quantum computing [25], a complete shift

in computing paradigm that allows to exploit the very quantum properties that are currently imposing limitations on classical devices, capitalizing on them to potentially unlock an even more powerful approach to computing [11].

Quantum computation, the use of quantum mechanics to perform computations [12], began as a field in the mid 1980s with the first theoretical notion of a computing device based on the principles of quantum mechanics. Since then, the field has evolved significantly, with real quantum computers being available to researchers today. Despite incredible progress, we are still in the Noisy Intermediate-Scale Quantum (NISQ) technology era, characterized by small, noisy, and error-prone quantum systems. Yet, theoretical proofs of quantum advantage [26][27], together with some initial results on noisy processors or simulated quantum systems, highlight the potential of a quantum approach in addressing some of the mentioned challenges we are facing with classical computing [11].

The field of quantum computing is relatively young and there are still a lot of unknowns. Apart from some specific domains, we do not know yet how exactly, and in what applications, can quantum computing, and within that quantum machine learning, effectively provide a practical advantage compared to a classical approach. However, the aforementioned theoretical proofs, together with some promising preliminary results in various QML applications, make quantum computing a valuable and worthwhile challenge for the scientific community to tackle. As a small contribution to this large effort, this thesis works towards the exploration of the limits and potentials of quantum-based generative adversarial networks for the application of real-image generation.

In the following subsections I introduce some foundational notions that are the bases for any quantum computing and quantum machine learning application, and consequently for this research.

Fundamental notions

Quantum bits, or *qubits*, are the basic units of information in quantum computing, analogous to bits in classical computing. Just like a classical bit has 2 possible states it can be in, either 0 or 1, a qubit can also be described as a mathematical object that can be in state $|0\rangle$ and $|1\rangle$, but also in any linear combination of these two states, *superposition*:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{4}$$

where α and β are complex numbers [13]. Even if a qubit is in a state of superposition, when we measure it, we can only get as measurement output either 0, with probability $|\alpha|^2$, or 1, with probability $|\beta|^2$. Because the sum of probabilities must always be equal to one, follows the *normalization constraint* $|\alpha|^2 + |\beta|^2 = 1$. Superposition is a fundamental quantum property that differentiates a quantum system from a classical one. It allows for a qubit to be able to exist in a continuum of states between $|0\rangle$ and $|1\rangle$, at least until measurements, where it collapses to either one. This property is responsible for the ability of quantum systems to store an *exponentially* greater amount of (hidden) information with each added qubit [11]. In fact, for a system of n qubits there are 2^n computational basis states, with the specific quantum state of the system described by 2^n *amplitudes* (coefficients of each computational basis state):

$$|\psi\rangle = \alpha_0|00\ldots0\rangle + \alpha_1|00\ldots1\rangle + \cdots + \alpha_{2^n-2}|11\ldots10\rangle + \alpha_{2^n-1}|11\ldots11\rangle \tag{5}$$

Which can me more succinctly expressed as:

$$|\psi\rangle = \sum_{x=0}^{2^n-1} \alpha_x|x\rangle \tag{6}$$

Note that the normalization condition still holds, i.e. the sum of the squares of the absolute values of the coefficients equals one:

$$\sum_{x=0}^{2^n-1} |\alpha_x|^2 = 1 \tag{7}$$

because each measurement result $x = (00\ldots0, 00\ldots1, 11\ldots10, 11\ldots11)$ can happen with a probability of $|\alpha_x|^2$. This exponential increase in the amount of information that can be stored with a linear growth in the number of qubits represents one of the greatest advantages that quantum computing provides over classical systems [11]. Leveraging this advantage is one way quantum computing may offer a solution to the limitations imposed by the above-mentioned halt of Moore's law in classical technology hardware development.

A common model used for describing quantum computations is that of *quantum circuits*, composed by *wires*, representing qubits, and *gates*, representing operations acting on these qubits to manipulate and transform their states. Gates apply linear transformation to the qubits, which is an important characteristic and stems from a general property of quantum mechanics, as it guarantees preservation of the normalization constraint. Gates are represented as matrices. Single qubits gates are described by 2×2 matrices, while n-qubits gates are described by $2^n \times 2^n$ matrices. Note that there exist an infinite amount of 2×2 valid quantum gates. However, in practice, we typically rely on much smaller sets of well-established gates. Specifically, certain combinations of these gates are classified as *universal sets*. By using only the gates within such a set, any quantum computation can be approximated. This is useful as it allows us to construct any algorithm by only relying on those gates, which can simplify the algorithm design process [11].

Variational Quantum Circuit

Variational or parameterized quantum circuits (VQCs and PQCs, respectively) are a specific type of circuit algorithm that is considered one of the most promising models within the hybrid quantum–classical ML framework for NISQ devices [28]. PQCs are typically composed of fixed gates such as controlled-NOTs or controlled-Zs gates, and trainable parameterized gates, i.e., gates with input parameters, such as rotation gates [19]. PQCs-based ML models are characterized by three main steps: initial state preparation, variational circuit U_θ, measurement and, if necessary, classical post-processing [28]. The variational circuit U_θ is where the core operations are performed and the state is modified. Constructed as a sequence of gates, each of which, by definition, applies a linear operation, U_θ can itself be thought as a large linear transformation operator parameterized by θ (the set of weights of the circuit's parameterized gates). In hybrid systems, the parameters of the circuit are updated using a classical optimization algorithm [28] to minimize a predefined loss function $L(\theta)$. For PQC learning, the output of the circuit is measured against a target result, and the gates' weights are adjusted accordingly, to diminish the distance between real and target output with every iteration of the training process [19]. Theoretically, there always exists a quantum circuit that can be trained to approximate a target function with an arbitrary small error. However, in practice this might require exponentially deep and therefore impractical circuit, especially in the NISQ era where every gate increases the probability of having error in the output. Therefore, variational circuits aim at compromising between being able to well approximate a function, while keeping the number of gates, the depth, and the number of parameters low [28]. Except for the general structure described above, the precise way in which the gates are arranged within the circuit, i.e., the ansatz architecture, is open to variation, and it is not always clear what makes a good ansatz for a specific application. This work aims at verifying whether superior architectures for an ansatz-generator in an image

generating hybrid quantum GAN can be found with the use of an evolutionary algorithm, producing better output results while keeping limited the number of gates and parameters, and the circuit depth.

Quantum GANs for Image Generation

There are very few research efforts to this day that explore the use of a quantum architecture for the generation of image data. Among these, is notable the work of Huang et al. [19], who in 2021 introduce the first architecture capable of learning and generating 8×8 gray scale images on a superconducting quantum processor. In their paper they also introduce the patch vs batch strategy, that allows for the same architectural structure to be used and adjusted to the available quantum resources. Specifically, the quantum batch approach is viable when the quantum resources are abundant, in a case where the number of qubits N is greater than $\lceil \log M \rceil$, where M is the number of pixels of the target images. In such a case, the batch strategy can be applied for parallel training, with a single generator generating multiple images simultaneously. In contrast, the patch strategy applies when there are limited computing resources ($N < \lceil \log M \rceil$). In this case, the images are split in different segments, or patches, which are generated separately, each by a different sub-generator, and stitched together afterwards [19]. Tsang et al. [20] build on this framework and propose a hybrid network, the Patch Quantum Wasserstein GAN (PQWGAN), that is capable of generating 28×28 gray scale images from the MNIST and Fashion MNIST datasets. Their architecture is based on a classical critic, equivalent to that proposed in the WGAN-GP [21], and a PQC generator of single-parameters RY gates for encoding, followed by a layered parameterized segment of rotation and CNOT gates, according to a general, hardware efficient ansatz structure. The PQWGAN demonstrates promising results in generating images comparable to classical GANs with significantly fewer trainable parameters, up to three orders of magnitude less [20]. However, the architecture still exhibits several limitations; while the images are recognizable, they often contain artifacts and noise, and there are several areas with potential for improvement, including explorations of different architectural configurations more tailored at the image generation task.

Having established the foundational concepts and algorithms underlying this work, the following Section presents a detailed explanation of the methods used for this feasibility study.

3 Methods

In this section, I present the feasibility study, which explores the use of an evolutionary algorithm for searching an optimized ansatz, which is then used as the generator of a hybrid classical-quantum generative adversarial network. I first describe the evolutionary algorithm in Section 3.1. Then in Section 3.2 I describe the architecture of the generative net that relies on the found ansatz for image generation.

3.1 Evolutionary Algorithm

The choice of an evolutionary algorithm for the optimization of a variational quantum circuit for the specific application of image generation is motivated by several factors. In general, ansatz design is a non-trivial task [11], complicated by elements such as gate choice, encoding technique, and the circuit structure itself. Additionally, the field is still in an early and exploratory phase, with limited references for designing application-specific ansätze. Consequently, a common choice for exploratory studies is to utilize application-agnostic, hardware efficient ansätze, such as in the case of PQWGAN [20]. Evolutionary algorithms are well suited for circuit search because they are designed to be able to explore a vast space in an unrestricted, unbiased fashion, and can solve such complex optimization problems with lower computing costs compared to traditional optimization techniques [29]. In this research, I verify the feasibility of using a quantum evolutionary search strategy, QES, initially designed for solving continuous optimization problems, for finding an effective quantum ansatz to be used as a generator of a hybrid quantum-classical GAN [29]. The algorithm relies on a mutation strategy, with quantum circuits acting as the individuals in the evolutionary process. The strategy alternates reproduction (copying the best-fit ansatz) and mutation for each of the offspring via a randomly chosen modification among 4 possible actions, guiding the evolution monotonically towards the optimum. The method is adaptable to NISQ devices as the generated circuits typically have a critical path depth of no more than 6, and a maximum circuit depth can also be set as needed. During the research, I explored several variables and configurations for the ansatz structure and initialization parameters. The following subsections describes the default configuration. Variations will be listed under experiments, in Section 4.

Set-up and Initialization

The circuit is initialized as empty, containing only the state embedding layer as shown in Figure 2. State embedding consists into mapping a vector of real numbers to the quantum state of the circuit. The length of the vector is equal to the number of qubits of the circuit, and each element in the vector is used as the parameter of an RY rotation gate applied to the corresponding qubit. This maps the uniform distribution into the quantum states as rotation of each qubit around the Y axis. The choice of a uniform distribution promotes uniform exploration of the latent space. A normal distribution could also be an interesting choice, encouraging the generator to focus more intensively on specific areas, thus enhancing learning, yet still occasionally exploring areas far from the mean. However, I determined that uniform would be a more suitable choice for this research due to resource constraints, as learning a good mapping from a latent sampled from a normal distribution would require more training time [20]. For similar reasons, although an interval for the distribution of $(-\pi, \pi)$ would have covered all possible angles for the RY gates, given the resource constraints, this would have been too large of a search space for the GAN's generator to explore in the limited training time and resources during the second phase of the algorithm. Thus, the interval $U(0, 1]$ was the one used in this work for the latent vector generation.

The gates that the algorithm can choose from during evolution are the 2-qubit Controlled-NOT gate and the parameterized single-qubit generic rotation gate with 3 angles of rotations $U(\theta, \phi, \lambda)$, whose matrix representation is, respectively:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \quad U(\theta, \phi, \lambda) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -e^{i\lambda}\sin\left(\frac{\theta}{2}\right) \\ e^{i\phi}\sin\left(\frac{\theta}{2}\right) & e^{i(\phi+\lambda)}\cos\left(\frac{\theta}{2}\right) \end{pmatrix}$$

This choice of gates as the gate family is due to several reasons. Firstly, $\{CNOT, U(\theta, \phi, \lambda)\}$ forms a universal gate set [11], and can therefore be used to approximate any unitary transformation to an arbitrary degree of accuracy. This is important for the evolutionary process, as it ensures the potential to discover any conceivable quantum state or ansatz, provided sufficient time and generations. Therefore, the algorithm can potentially find the global minimum of the objective function, although it may sometimes get stuck at local minima. If the fitness function is well-designed and accurately reflects the desired properties of the ansatz, this global minimum would correspond to an ideal solution for the given task.

The gate set $\{CNOT, U(\theta, \phi, \lambda)\}$ is also aligned with the typical structure of hardware-efficient parameterized circuits, being made up of an entangling gate, the $CNOT$, and the trainable single-qubit rotation gate, $U(\theta, \phi, \lambda)$ [28]. Tsang et al. [20] use this same gate set for their hardware-efficient PQC, showing that it is possible to achieve reasonably good results for the task of image generation with the set. Finally, preliminary tests were conducted to explore the potential of an alternative gate set family - $\{R_x, R_y, R_z, R_{xx}, R_{yy}, R_{zz}, H\}$ -, but yielded unsatisfactory results. Consequently, $\{CNOT, U(\theta, \phi, \lambda)\}$ was deemed as an appropriate choice for finding better ansatz architectures via evolutionary search, and selected for this task.

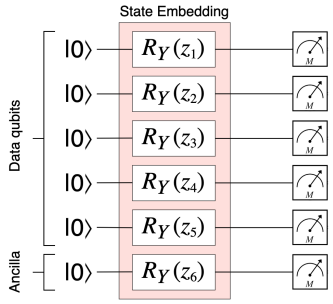


Figure 2: Default starting circuit for the evolutionary algorithm. The circuit is empty except for the Sate Embedding layer of $R_y(\theta)$ rotation gates.

The initialization ansatz shown in Figure 2 is the parent circuit for the first generation of the algorithm. The ansatz then undergoes evolution in a Darwinian selection process based exclusively on mutation. The parent is replicated several times, each replica being an offspring. Each offspring is mutated in some way, for example by adding or removing a gate. The offspring of each generation are evaluated against a fitness metric, described in more details in the following sections. In the default set-up, the metric used is the earth mover distance between the generated and the real data distributions. The offspring with the best fitness is the survivor for that generation, and becomes the parent for the next. The process is repeated until a termination criteria is reached. The general steps of the algorithm are schematized as pseudocode in Algorithm 1.

Algorithm 1: Evolutionary Algorithm for Ansatz Search

Input: Training images, Initial parameters, Fitness function $F(\text{circuit}, \text{training\_data})$, Function to apply an action to a circuit $A(\text{action}, \text{circuit})$.

Output: Optimized ansatz

```
best_ansatz ← Initialize parent ansatz
best_fitness ← F(best_ansatz)
for g = 1 to n_generations do
    population ← [ ]
    fitnesses ← [ ]
    for c = 1 to n_children do
        current_child ← best_ansatz
        n_actions ← 1 or more, depending on multi_action_probability
        for a = 1 to n_actions do
            Select action in {Add, Delete, Swap, Mutate}
            current_child ← A(a, current_child)
        population.append(current_child)
        current_fitness ← F(current_child)
        fitnesses.append(current_fitness)
    current_best_fitness ← min(fitnesses)
    best_i ← find_index(fitnesses, current_best_fitness)
    current_best_ansatz ← population[best_i]
    if current_best_fitness < best_fitness then
        best_fitness ← current_best_fitness
        best_ansatz ← current_best_ansatz
    if termination criteria met then
        return best_ansatz
return best_ansatz
```

Evolution

The evolutionary algorithm begins by making as many copies of the parent ansatz as the specified number of offsprings. Each offspring circuit undergoes through the application of one or more of four possible actions. The number of actions applied to each offspring is determined through a probabilistic approach. Initially, one action is guaranteed. Additional actions may be applied based on a multi-action probability p. Specifically, a random number is drawn between 0 and 1, and if the number is less than p an additional action is added. This process repeats, drawing new numbers and adding one action each time a number is less than p. This is repeated until failure, i.e., when the drawn number exceeds p. In other words, at each iteration there is a p probability of increasing the number of actions by one, which translates in an increase in number of actions by a number k with a probability $P(X = k) = p^k \times (1 - p)$. The possible actions that can be used to mutate a circuit are: Addition (A), Deletion (D), Swapping (S), and Mutation (M). As the names themselves suggest, the actions perform the following modifications to the circuit:

- **Addition**: Select a gate at random from the chosen gate family $\{CNOT, U(\theta, \phi, \lambda)\}$. Add the gate to a qubit of the circuit, also chosen at random. If the selected gate is a 2-qubits gate, choose both qubits at random.

- **Deletion**: Select a random gate from the circuit, and delete it.
- **Swapping**: A gate of the circuit is selected at random, and substituted with another gate picked at random from the given gate family. If the new gate requires the same number of qubits as the original gate, the gates are directly swapped on the same qubit(s). If the new gate requires fewer qubits than the original, the qubits for the new gate are chosen only among the qubits of the old gate. If the new gate requires more qubits than the original, additional qubits are randomly selected from the circuit's remaining qubits. Then, the chosen qubits are shuffled so that the target and control qubits for the new gate are assigned randomly from the selected qubits subset.
- **Mutation**: if there are parameterized gates in the circuit, the algorithm randomly select one among these gates, and changes one of its parameters by an amount up to d_θ in the positive or negative direction.

Refer to the Appendix, Algorithm 2, for schematized pseudocode of how mutation takes place.

Evaluation

Once actions have been performed on all of the children, the current population is evaluated. Each circuit is assigned a fitness score based on the quality of the images that it produces. This is done by generating several batches of images using the given circuit. Then, calculating the Earth Mover Distance between the generated images and real images. If the evolutionary algorithm is tasked with finding an ansatz to be used as a sub-generator within the patch strategy, the ansätze are evaluated on a patch level. For example, if a 28×28 image is divided into 28 patches to be generated separately, and the evolutionary algorithm is tasked with finding an ansatz for the generation of just one patch, the EMD is calculated between batches of generated patches and randomly sampled patches of real images, both of size 1×28.

The circuit that produces images (or patches) with the shortest calculated distance between generated and real data is labeled as the survivor of the generation. If its score is better than previously obtained scores, the corresponding circuit is designated as the new parent circuits for following generations. The process repeats until a given stopping condition is reached.

Termination and Output

Effective termination conditions are important to prevent premature termination while avoiding unnecessary computations. Because the number of required evaluations is application and algorithm dependent, appropriate stop conditions are often found via a trial and error process [30]. This algorithm relies on an ensemble of direct termination conditions, chosen to achieve the aforementioned goals: a limit on the maximum number of generations without improvement and a limit on the total number of fitness evaluations. Both values for these conditions were found via trial and error. If either condition is reached, the algorithm is halted and the best found ansatz is returned.

Additional measures are also implemented to avoid premature termination that are not direct termination conditions. To avoid getting stuck at local minima, if no improvement is found after a set number of generations, the mutation weight for the angle of the parameterized gates is temporarily increased. The likelihood of selecting more than one action per offspring is also incremented to have more significant changes to the current parent, helping to escape stagnation at a local minima. Additionally, an upper bound is set on the maximum length of the critical path, the longest sequence of consecutive gates, to prevent the circuit from becoming too deep and potentially excessively focused on a single qubit.

I also experimented with a hitting-a-bound termination condition, where the algorithm is stopped once the fitness evaluation reaches a certain value, often found heuristically, that is deemed good enough for fitness [30]. I chose as the value the Earth Mover's Distance between real images and images generated by an untrained, hardware-efficient ansatz consisting of RY and CNOT gates. The rationale behind this was that if an untrained ansatz found with the evolutionary approach can produce better images than an untrained baseline ansatz, then a trained version of the algorithm might achieve even better outputs in terms of quality. However, this approach proved unfeasible as the scores of the ansätze generated by the evolutionary algorithm rapidly surpassed those of the untrained baseline ansatz, causing premature termination of the evolutionary search. Therefore, this condition was discarded. In the future, it might be interesting to experiment with other hitting-a-bound termination values, for example based on performance of a trained hardware-efficient ansatz, in case the fitness of the evolutionary algorithm was to be computed on a whole-image level rather than a patch level.

Once the algorithm terminates the evolutionary search, the best-found architecture, along with the necessary hyperparameters, is returned and passed to the subsequent step of the algorithm pipeline -GAN training- where the identified ansatz is utilized as the generator.

3.2 Quantum GAN for Image Generation

The GAN architecture is based on the patch strategy proposed by Huang et al. [19] and later adopted by Tsang et al. [20] with the PQWGAN framework. The network is hybrid, with only the generator being quantum, and the critic a classical neural network.

The choice of utilizing a hybrid GAN is due to two reasons. First, due to computational resource constraints. Since all quantum systems were simulated, there was a hard limit on the size of the system that could be handled. Therefore, to stay within these resource limits, I chose to implement only one between the generator and the critic as quantum. Having both components being quantum would have been more costly and time-consuming, as simulating multiple systems significantly increases the computational overhead and extends the runtime. Second, considering this study's exploratory nature in a largely unexplored domain, I preferred maintaining one of the two model's components as classic. This way, I could rely on a well-tested, widely used and known to be well-functioning critic, limiting the number of variables and unknown elements in the architecture to a more manageable size, allowing me to better focus and experiment with the quantum generator. The choice of having the generator as quantum instead of the critic is purely practical. In fact, in generation tasks aimed at generating classical data, such as in this application, a classical generator and a quantum critic would lead to a counter-productive disadvantage for the generator. Although the generator would be capable of generating the data, it would always be able to be beaten by the critic, which would leverage on quantum supremacy. This would cause convergence difficulties for the generator, undermining the adversarial learning strategy central to GANs. Therefore, in a hybrid set-up, utilizing a quantum generator and a classical critic is the most sensible approach [15].

Classical Critic

The critic of the network, shown in Figure 3, is a classic feed-forward net, composed by a sequence of fully connected layers and LeakyReLU activation functions. The input layer takes the flattened image data and transforms it through a fully connected layer of 512 neurons, followed by a LeakyReLU activation function. Follows a linear layer with 256 neurons, again

followed by a LeakyReLU activation. The slope for the negative values of the LeakyReLU is set to 0.2. Finally, the output is generated by a single neuron that provides a continuous score indicative of the perceived realism of the image. The choice of this architecture is based on it being a standard that has demonstrated good performance in image generation applications with fully classical networks. Moreover, this choice allows for better comparability with classical nets as well as with the PQWGAN framework [20].

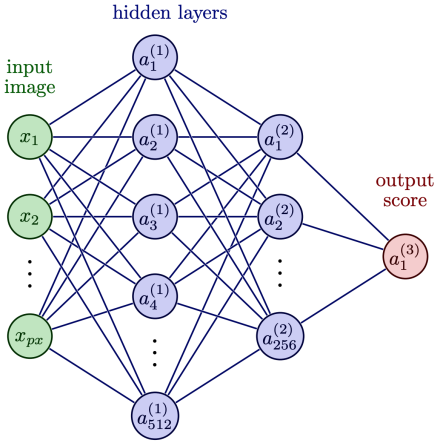


Figure 3: Diagram of the critic architecture (classic). In the input layer, each node is a pixel value from the input image. The two hidden fully connected layers have 512 and 256 nodes respectively. The output layer consists of a single node that outputs a continuous value, the critic score, which assesses the perceived realism of the given image.

Quantum Generator

The generator structure differs for every run, given that it was this thesis objective to experiment with the capability of architectures found with evolutionary search to generate realistic images from the MNIST dataset. To illustrate the general procedure, I select and describe an example architecture found via evolution. Some elements of the architecture are common to all, while others, specifically the number and position of $CNOT$ and parameterized rotation gates, are different for different runs.

The first layer of the circuit is always the state embedding layer. The circuit receives as input a latent vector of length N, where N represents the total number of qubits. Just like for the evolutionary circuit, the latent vector is drawn from a uniform distribution over the interval $[0, 1)$:

$$\mathbf{z} = (z_1, z_2, \ldots, z_N) \quad \text{where} \quad z_i \sim \text{Uniform}(0, 1) \tag{8}$$

The input latent vector is different for each generated image. In configurations involving multiple sub-generators (patch-strategy), the same latent vector is shared among all sub-generators, but it still differs for every generated image. This vector is encoded in the (sub)generator by using each of its elements as the parameter for the RY rotation gates. Starting from the $|0\rangle^{\otimes N}$ state, where N is the number of total qubits of the system, we get to the latent state $|z\rangle$ via the encoding layer:

$$|z\rangle = RY^1(z_1) \otimes RY^2(z_2) \otimes \cdots \otimes RY^N(z_N) \tag{9}$$

Where $RY^i(z_i)$ is the rotation gate RY applied to the i-th qubit of the circuit with parameter the i-th element of the latent vector. State embedding is important because it introduces variability in the input of the circuit, allowing to generate diverse images and to explore different areas of the target distribution. The generator's task, achieved via weight updates, is to learn an appropriate mapping between the latent space and the output, so that the output distribution resembles that of real data.

The state embedding layer is followed by the parameterized layer, where the learning takes place. This is the layer that is found by the evolutionary algorithm, and differs for every experiment. In general, the layer is a combination of $CNOT$s and parameterized $U(\theta, \phi, \lambda)$ gates. An example of an output circuit found via evolution is displayed in Figure 4.

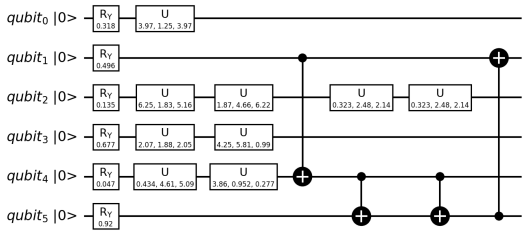


Figure 4: An example of a 6-qubits circuit of parameterized $U(\theta, \phi, \lambda)$ gates and Control NOT gates found by the evolutionary algorithm. The first layer of RY gates is the embedding layer.

The parameterized layer can be thought of as a single large linear operation $U_G(\theta)$ since, by definition, each quantum gate, and all combination of quantum gates, are unitary operations and linearly transform the quantum state:

$$|\psi\rangle = U_G(\theta)\,|z\rangle \tag{10}$$

Where θ is a parameter vector of varying shape, depending on the number of trainable gates of the specific circuit configuration. For example, with a circuit of 6 rotation $U(\theta, \phi, \lambda)$ gates and 28 sub-generators, θ will consist of $3 \times 6 \times 28 = 504$ trainable parameters.

As mentioned, to accommodate for limited computational resources, I generally adopted the patch strategy proposed by [20]. In the patch strategy, each image is divided into segment of equal shape. For example, a valid division would be splitting a 28×28 image into 14 patches of size 2×28. In this case, each sub-generators is tasked with generating only a fragment of the image, a patch. Then, the produced patches are stitched together to obtain the final image. Each sub-generator has an identical structure, the one identified via evolution, and receives the same latent vector when generating patches for the same image. What differentiates the sub-generators and allows them to produce diverse output patches is that each sub-generator has unique weights, which are all trained to optimize the quality of the overall output image, so that each sub-generator specializes in the generation of that specific patch.

Non-linear mapping

Once the parameterized layer operations are executed, it becomes essential to introduce non-linearity into the system. One of the characteristics of deep learning algorithms, in fact, is to learn

non-linear transformations of their input to generate the output. This is what allows models like neural networks to find and replicate complex patterns. However, quantum gates and quantum gate sets apply linear transformations by definition, meaning any transformation applied by the encoding and parameterized layers of the circuits will always be linear. Therefore, introducing non-linearity is necessary for the quantum generator to replicate the non-linear mapping typical of classical generators. This is done with the technique proposed in [19], exploiting ancillary qubits. I illustrate the process of non-linear mapping for one sub-generator, but the process is identical for all sub-generators or also in the case of a single generator.

For a specific sub-generator G_s, whose circuit is represented by the trainable unitary operation $U_s(\theta_s)$ the output state is:

$$|\Psi_s(z)\rangle = U_s(\theta_s)\,|z\rangle \tag{11}$$

$|\Psi_s(z)\rangle$ is the state obtained after applying the unitary operations of the embedding and the parameterized layers, having as input the latent vector $|z\rangle$. Given this state, non-linearity is introduced by taking the partial measurement on the ancillary sub-system, and then tracing out the ancilla qubits, to obtain the resulting state of the data qubits. By so doing, we are left with the post-measurement state $\rho(z)$:

$$\rho(z) = Tr_A \frac{\Pi \otimes \mathbb{I}\,|\Psi(z)\rangle\,\langle\Psi(z)|}{\langle\Psi(z)|\,\Pi \otimes \mathbb{I}\,|\Psi(z)\rangle} \tag{12}$$

Specifically, we are interested in the partial measurement that considers the probabilities where the ancilla qubits are measured in the zero state:

$$\Pi = (|0\rangle\,\langle 0|)^{\otimes N_A} \tag{13}$$

Where N_A is the number of ancillas. Equation 14 then becomes:

$$\rho(z) = Tr_A \frac{(|0\rangle\,\langle 0|)^{\otimes N_A} \otimes \mathbb{I}\,|\Psi(z)\rangle\,\langle\Psi(z)|}{\langle\Psi(z)|\,(|0\rangle\,\langle 0|)^{\otimes N_A} \otimes \mathbb{I}\,|\Psi(z)\rangle} \tag{14}$$

This post-measurement state $\rho(z)$ is dependent on the input state $|z\rangle$ both in the numerator and the denominator, which means it is a non-linear transformation of $|z\rangle$. This allows the circuit to learn a non-linear mapping function from input to output, like in a deep learning system. Finally, we consider the probabilities of each computational basis state of the qubits that are left, the data qubits, to obtain the sub-generator output:

$$G_s(z) = [p(0), p(1), \ldots, p(2^{N-N_A-1})] \tag{15}$$

$G_s(z)$ is a vector of length 2^{N_D} where $N_D = N - N_A$ is the number of data qubits (total qubits minus ancilla). In the implementation, for sake of simplicity, the ancilla qubits are assumed to be at the top part of the qubit register, while the data qubits are assumed to be at the bottom part. This allows to take the partial measurement and traced out ancilla by sub-setting the vector of probabilities of all computational basis states to only include the subset up to the 2^{N_D}-th element.

Training algorithm

The training algorithm follows the same steps of the classical WGAN-GP. The classical critic is initialized with values drawn from a uniform distribution $U(-\sqrt{k}, \sqrt{k})$, where $k = \frac{1}{input features}$ as per default pytorch `nn.Linear` implementation. The number of input features differs according to

the patch size, and is in range between 28, in the case of 28 patches, to 784, in the case of 1 patch, i.e., the whole image. For the majority of the experiments, the number of patches was chosen to be either 28 or 14 because the small-patches setup showed to have better performance, hence 28 to 56 features, with a corresponding weight initialization distribution of $U(-0.19, 0.19)$ and $U(-0.13, 0.13)$. The same distribution is utilized for initializing the bias nodes. The Quantum generator is initialized with weights drawn from a uniform distribution of $U(0, 1]$. I attempted initializing the generator with the weights identified by the evolutionary algorithm, since the latter is itself an optimization algorithm and outputs an already partially optimized circuit. However, by so doing, in the case of multiple sub-generators the weights are all initialized the same, and this hindered the learning process, which heavily relies on being initialized on a diverse array of weights. Hence, I reverted to initialization with small random weights drawn from $U(0, 1]$.

The optimizer used is Adam for both the generator and the discriminator, with a learning rate of 0.0002 for the discriminator and of 0.01 for the generator, and β_1 and β_2 values of 0 and 0.9 respectively. These values are the same used for the PQWGAN [20], because hyperparameter optimization of the hybrid net, in itself a time and resource consuming task, made even more so by the system being partially quantum, went beyond the means and scope of this research.

For every batch of images a new latent vector is generated of size ($n_{batches}$, n_{qubits}), so that each generated image will have its own latent vector, but sub-generators will share the same. Then, the vector is fed to the generator to generate a batch of fake images. At this point, the critic score is calculated for both a batch of real images and the generated batch of fake images. The real and fake validity score is simply the mean of the scores for all images in each batch. The discriminator loss is calculated as:

$$\min_D L(D) = -D(x) + D(G(z)) + \lambda_{GP} * GP \tag{16}$$

Where x is a sample of real data, z is the latent vector input to the generator, $G(z)$ is the generator output, and λ_{GP} is a constant, the coefficient for the gradient penalty term, set to 10 by default. The gradient penalty equation is:

$$GP = (\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2 \tag{17}$$

Where $\hat{x}$ is sampled uniformly in between the real and generated data, as proposed in the original WGAN-GP formulation [21]. The training process is performed alternatively between generator and critic, assuming one of the two is fixed while the other is trained. The generator is only updated once every 5 times that the critic is, because, being penalized, the critic requires more time for training than in traditional GANs. The critic weights are updated in order to minimize equation 16, while for the generator, the weights are updated according to:

$$\min_G L(G) = -D(G(z)) \tag{18}$$

Because the system is entirely simulated on a device that is classically end-to-end differentiable, the weight update for the generator also take place via pytorch classical backpropagation algorithm.

The following Section contains an overview of the experimental set-up, including the dataset used for running the experiments, and the evaluation metrics employed to assess the performance of both the evolutionary algorithm and the entire QES-GAN pipeline.

4 Experimental Setup

4.1 Data and Computational Framework

I picked the publicly available MNIST dataset [31] for all experiments in this study. MNIST (Modified National Institute of Standards and Technology) dataset is a collection of hand-written digit images. It includes 60,000 training samples and 10,000 test samples. All images in the datasets are already preprocessed with size normalization, and the written digits are centered in the 28×28 pixels grid. For resource efficiency, I only utilized a total of 1700 images, split into training and validation among the different algorithms as shown in Table 1.

Table 1: Dataset Sizes for GAN and Evolutionary Algorithm

Model	**Training Size**	**Validation Size**	**Total**
Evolutionary	400	100	500
GAN	1000	200	1200
Total:			**1700**

Figure 5 shows some samples of the data. I chose MNIST for several reasons. Firstly, it is a classic benchmark dataset in machine learning applications, and therefore it can provide a standardized framework to compare this work and its results with other studies and research. Moreover, the dataset has suitable characteristics for the application. Due to the exploratory nature of the work, to Quantum GANs being a field in its early stages of research, and to the limited available resources to simulate quantum systems, it is sensible to use small images for the experiments. MNIST images are gray scale, which reduces the image tensor size by 1/3 compared to a 3-channel RGB image. Their default size, 784 pixels in a 28x28 grid, is small enough to be compatible with limited computing resources, but large enough to be an interesting challenge for a (simulated) quantum generator. Although small, this resolution is still sufficient to capture the essential features and variations of hand-written digits, making it a relevant training set for a pattern recognition and generation task. Moreover, the primary objective of the thesis is to test the evolutionary algorithm ability to find ansätze suitable for image generation within a hybrid GAN framework, and using a smaller dataset still allows the algorithm to learn the key features it needs to look for without consuming too much memory and unnecessarily slowing down computations. Because of the same reasons, I further subset the dataset to only the first two digits, '0' and '1'. This allowed to reduce the search space for our quantum generator to a manageable size. A broader search scope would have been too large and expensive for the architecture to handle at this stage of development and with the available resources.

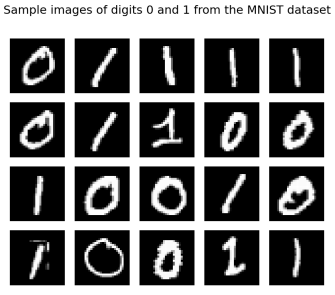


Figure 5: Samples images of digits 0 and 1 from the MNIST dataset.

During loading, images are normalized to a mean of 0.5 and a standard deviation of 0.5. This is done to have consistent data distribution, stabilize the learning process and facilitate convergence. Initialization parameters for all experiments are listed in the Appendix, in Section 8.3.

The project was developed with Python 3.10 [32] using Qiskit [33] for quantum computing tasks, PennyLane [34] for both quantum computing and handling the classical-quantum interface, and Pytorch [35] for the classical ML tasks. The quantum systems were simulated without noise using both PennyLane's DefaultQubit and Qiskit's Statevector simulators. This research was made possible, in part, using the Data Science Research Infrastructure (DSRI) hosted at Maastricht University[2], which provided access to remote CPU and GPU nodes.

All code developed for this study is publicly available and can be accessed on GitHub[3]. Setup instructions and documentation are provided in the README file found in the code repository.

4.2 Evaluation metrics

Evaluating GANs for image generation is a challenging task because the generated images do not have a clear right or wrong answer. There can be multiple valid solutions, and the evaluation cannot rely on a simple 1-on-1 correspondence with real images. Several measures have been proposed and used for GAN evaluation, but there is no consensus on the measure that best captures output quality, as this is application and model dependent. In general, a good measure should establish the quality of the generated images in terms of both realism and intra- and inter-class diversity in the samples [36]. For this research, I used a combination of metrics, both quantitative and qualitative, to assess the performance of each tested architecture and gauge the evolutionary algorithm's ability to generate interesting ansatz configurations for image generation. The results were evaluated for output image quality as well as architectural complexity. The former was assessed using an ensemble of quantitative and qualitative metrics, as described below. Of these, not all performed equally well, with FID, EMD, and visual inspection being the ones that I ultimately relied on the most, as they proved to be the most informative. Generated architectures were also compared in terms of their complexity and required number of parameters, with simpler architectures being preferred given comparable quality outputs. The following paragraphs contain a description of the metrics tried and selected for the evaluation of the experiments illustrated in Section 5.

4.3 Quality of the output

Earth Mover Distance

As the fitness function for the evolutionary algorithm, I used the Wasserstein-1 distance, or Earth Mover Distance (EMD), between generated images and real ones, both to guide the search and as a measure of quality of the final output on the validation set. The EMD is a similarity metric between distributions. It represents the minimal cost required to transform one distribution into another, where cost is defined as the product between the amount of probability mass that needs to be moved and the distance it needs to travel [37]. In the patch configuration, where the algorithm is tasked with finding an ansatz for a single patch, the EMD is calculated between patches randomly sampled from real images and those generated by the evolved ansatz. Although a more robust approach would involve calculating the distance over multiple patches or the entire generated image, this would be excessively expensive to do at each generation of

[2]Refer to their website for further information: https://dsri.maastrichtuniversity.nl/

[3]https://github.com/BiancaMass/EVOL-WGAN-COMPLETE.git

the evolutionary algorithm as it would require generating batches of images with multiple sub-generators for every offspring. Therefore, I opted to calculate the EMD based only on a single patch. The EMD was also employed for the GAN network itself. In fact, although the WGAN is designed in a way such that the critic, when properly trained, approximates the Wasserstein distance, I also independently calculated the distance every few epochs as an additional quality metric. In this case, the calculation was performed on the entire generated images.

Fréchet Inception Distance

Fréchet Inception Distance (FID) is a standard measure for evaluating the performance of GANs [36], originally introduced in 2017 [38]. FID is a network-based, distribution to distribution distance metric. It works by combining the Fréchet distance with Google's Inception V3 model [39]. Specifically, it passes the generated and real images through the Inception net, and subsequently measures the distance between the distributions obtained from a pooling layer, usually the last one, based on their means and covariance matrices. Being a distance estimation, lower FID values correspond to better generated image quality. FID was chosen as a metric because it provides a quantitative assessment of both the realism (visual quality) and diversity of the generated images, and it has been shown to align with human perceptual evaluation. It has also been shown to be robust to noise and to detect artifacts and intra-class mode dropping, i.e., generating only one image per class [36]. Because I was working with a limited number of small images, I opted to utilize the global average pooled features extracted from the second max pooling layer instead of the default third max pooling layer, as it relies on only 192 features instead of the original 2048, and has a less sparse space. These scores are layer dependent and hence a score obtained from this layer cannot be compared with a score obtained from the third pooling layer [40]. Although using the second max-pooling layer does not guarantee correlation between the FID score and perceptual quality, through my experiment I find that there is still correlation between the two, and the FID calculated this way is still a valid and reliable metric for image visual quality. Due to computational constraints, FID was calculated only at the last iteration of the GAN training process on 200 generated images and 200 test images. It was therefore an informative metric on the quality of the generated images at the final stage of training.

Visual Inspection

In addition to quantitative metrics, I also used visual inspection, as checking each architecture's output manually was still a feasible task given the simplicity of the dataset, with only 2 classes, and it remains one of the most common ways to evaluate GANs [36]. This was done by observing randomly sampled images during GAN training every 20 batches to monitor progress, as well as examining randomly sampled images from the final output. These images were then qualitatively scored on a 5-points scale as very poor, poor, fair, good, or excellent. Although this method may be infeasible for larger and more complex applications and is limited by subjectivity and individual bias, the dataset's simplicity required no domain knowledge, unlike highly sensitive applications such as medical imaging, making visual inspection a viable and effective method for this particular study.

Pre-trained Wasserstein Critic

Network-based distance estimators are commonly used in GANs as measures of generated image quality, because they provide a quantitative metric for the quality of generated images by estimating a distance between distributions, bypassing the challenge of not having a one-to-one correspondence with a real dataset [36]. FID is an example of this approach, being based on a

pretrained Inception network. Another example is the Wasserstein critic itself, a neural network trained to approximate the Wasserstein-1 distance between real data and generated data.

As part of this work, I tested using a pretrained critic, i.e., an estimate of the Wasserstein distance between real and generated images, to guide the evolutionary process. This approach provides the advantage of having a cost function that can be trained for a specific application, to the exact dataset up to a patch level, potentially alleviating one of the challenges encountered during evolution, of evaluating the quality of images at a small-patch level, where minimal spatial and contextual image information is available.

I experimented with training a critic on both hybrid and fully classical architectures. However, despite various attempts, the function failed to steer the evolution toward interesting configurations. Additionally, the approach was cumbersome and non-adaptable, requiring a separately pretrained critic for each dataset and each patch configuration to achieve the desired level of specialization. It also had high time complexity, both due to the need for separate training of a critic for each dataset and patch configuration, and because calculating the estimated distance at each iteration during evolution was resource-intensive, slowing down the evolutionary process. Reaching a point where it might have worked would have required more fine-tuning and improvement, demanding a time commitment beyond the scope of this research. Due to these reasons and the unsatisfactory results, I decided to abandon the estimated Wasserstein distance and opt instead for the EMD calculation for the final formulation of the algorithm, a standardized metric that proved to be more reliable, consistent, and efficient.

Sample efficiency

Sample efficiency was used to estimate the reliability of FID and EMD as metrics, and for benchmarking. The procedure consists in finding the number of samples required for any metric to be able to correctly distinguish between real and generated images. This is done by calculating the distance, in this case the FID or the EMD, between two disjoints sets of real images S_{r0} and S_{r1}, and a set of real images S_{r0} and generated images S_g, all of equal size n. A distance measure ρ is considered reliable and efficient if it scores the distance between the two real sets $\rho(S_{r0}, Sr1)$ as lower than the distance between the real and generated sets $\rho(S_{r0}, Sg)$ with a small n. In other words, the number of samples needed for a measure to distinguish real from generated data reflects its sample complexity [36]. I tested FID and EMD's sample efficiency using generated images from different models: a fully classic WGAN-GP net, PQWGAN [20] with default parameters, and my best generated models. As shown in Section 5, Table 9 and 10, both metrics passed the test even on very small sample size, and even against images generated by fully classical, well-performing, benchmark architecture. These test results, together with a general concordance with visual inspection, confirmed that EMD and FID calculated on the second pooling layer are indicative quantitative measures for the assessment of the net's output quality.

4.4 Computational Efficiency

Number of generator parameters

Computational efficiency could not be directly compared in terms of training time due to testing on different hardware. As an indicative measure, I compared the number of trainable parameters of the quantum generator across models. I did not consider the critic, as that segment was fixed and not the focus of this research. Although the main goal was to test the feasibility of the approach rather than to optimize efficiency, analyzing the number of parameters still provides interesting insights into the computational demands of the found networks.

5 Results

The experiments were designed to evaluate whether the evolutionary algorithm is capable of identifying high-performing ansatz architectures that could produce better visual-quality outputs than a standard layered hardware-efficient ansatz of parameterized gates followed by a cascade of fixed two-qubit gates.

I initially conducted exploratory experiments to assess the algorithm's performance across various architectural configurations, namely the number of patches, layers, and ancilla qubits. For this preliminary exploration I trained the GAN network for only 12 epochs and a limited number of generations. This was done to identify the better configurations while balancing the need for reasonable training times and computational costs. After identifying the better configurations for patches and ancilla qubits, I expanded the scope of the exploration by increasing the number of evolutionary generations and testing different evolutionary hyperparameters, namely action weights and multi-action probability. Finally, using the best found parameters and hyperparameters, I performed 10 full searches for final testing. Although the GAN's hyperparameters themselves can be fine-tuned, extensive optimization was beyond the scope of this research, and it would not have been a feasible task due to the time and resources required for fine-grained tuning of simulated quantum circuits. Therefore, I kept the GAN's parameters consistent with those used in Tsang et al. [20], which also allows for a more direct comparison of results. For all experiments, performance is evaluated based on EMD and FID score calculated between generated images and the test set, and visual inspection, as well as informed by the GAN training curves and pattern of improvement of the EMD score during evolutionary search. Architectures are also compared in terms of the number of parameters to evaluate computational complexity. Configurations that achieve comparable results but have fewer parameters and less depth are considered as better-performing, as they require less computational resources and are more hardware-efficient and less error-prone on real devices. The algorithm's non-deterministic nature means that results can vary each time it is executed, even with identical initialization parameters. I chose not to use the same random seed for all runs to understand the behavior and variability of the algorithm under different conditions. The seed was randomly generated before each run, and saved together with the rest of the output, for sake of replicability. I conducted multiple iterations for most configurations but due to time limits and resource constraints, a fully exhaustive search was not feasible. The findings should therefore be interpreted as a preliminary assessment of the algorithm's performance, providing initial insights and guidelines for its further improvement and potentiality for the application.

In this section, I present a summary of the experimental results for the various configurations and parameters. An overview of the experimental structure is displayed in Figure 6.

5.1 Stage 1: Patches and Layers

The first stage of experiments was performed with the goal of testing how well the algorithm works with different setups and to find out which architecture configuration gives better results. This then allowed to narrow down the number of experiments for subsequent fine-tuning. Given that this was an exploratory stage, the number of epochs and evolutionary generations was kept low to maximize exploration, prioritizing the number of configurations tested over the depth of any single configuration. I started with testing the performance of the evolutionary algorithm on different patch sizes, from 28 to 784 pixels, with a corresponding increase in the number of data qubits from 5 to 10, while keeping other parameters and hyperparameters constant. The hyperparameters for the tested architectures are listed in the Appendix, Table 13, while results are shown below, grouped by number of layers in Table 2 and by patch number in Table 3.

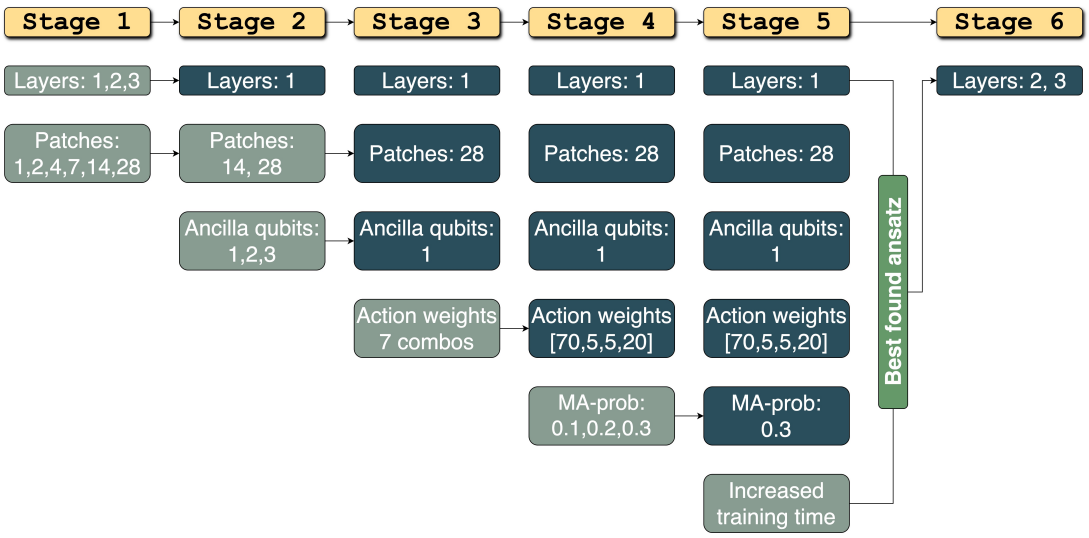


Figure 6: Schematic representation of experiments structure divided in Stages 1-6. Tested parameters are shown in light gray, while found best parameters selected for successive stages are shown in stone blue. The last stage was not a test on the evolutionary algorithm itself, but an expansion of the previous stage with its best found architecture. The tested parameters are: **1**) Different number of layers and patches. **2**) Different number of patches and ancillary qubits. **3**) Different combinations of action weights. **4**) Different values of multi-action probability. **5**) Best found parameters from Stages 1-4 used for 10 evolutionary searches, to test general performance of the algorithm. **6**) The best architecture from Stage 5 used as a layer of a multi-layer generator.

Summary - Patches and Layers

The first stage tested configurations with different number of layers (1, 2, 3) and patches (1, 2, 7, 14, 28). The best results were obtained with the largest number of patches (28), with an average FID of 9.85 ± 9.13 and an average EMD of 18.37 ± 1.73. The difference between groups of different layers was minimal and non-significant.

Table 2: *Stage 1*. Evaluation metrics for different configurations of layers and number of patches, organized by number of layers. Qubits refer to the data qubits plus ancilla, fixed as 1 in this stage. FID and EMD are the calculated distances between generated images and test set, with corresponding group mean and standard deviation. Refer to Table 13 in the Appendix for complete overview of hyperparameters.

Layers	Patches	Qubits	FID	FID $\bar{x} \pm \sigma$	EMD	EMD $\bar{x} \pm \sigma$	Visual
1	1	11	25.01		19.80		very poor
1	2	10	23.07		19.77		very poor
1	7	8	19.52	17.95 ± 7.98	19.99	**19.35** $\pm$ 1.30	very poor
1	14	7	17.48		20.13		very poor
1	28	6	4.69		17.04		fair
2	1	11	25.01		19.80		very poor
2	2	10	23.69		19.84		very poor
2	7	8	19.06	18.41 ± 8.46	20.00	19.52 ± 1.08	poor
2	14	7	3.91		17.62		fair
2	28	6	20.39		20.32		very poor
3	1	11	22.99		19.78		very poor
3	2	10	24.11		19.85		very poor
3	7	8	16.76	**17.19** $\pm$ 7.81	19.69	19.44 ± 0.96	poor
3	14	7	17.64		20.13		very poor
3	28	6	4.46		17.74		fair

Table 3: *Stage 1*. Evaluation metrics for different configurations of layers and number of patches, organized by number of patches. Qubits refer to the data qubits plus ancilla, fixed as 1 in this stage. FID and EMD are the calculated distances between generated images and test set, with corresponding group mean and standard deviation. Refer to Table 13 in the Appendix for complete overview of hyperparameters.

Patches	**Layers**	**Qubits**	**FID**	**FID** $\bar{x} \pm \sigma$	**EMD**	**EMD** $\bar{x} \pm \sigma$	**Visual**
1	1	11	25.01		19.80		very poor
1	2	11	25.01	24.34 ± 1.17	19.80	19.80 ± 0.01	very poor
1	3	11	22.99		19.78		very poor
2	1	10	23.07		19.77		very poor
2	2	10	23.69	23.62 ± 0.52	19.84	19.82 ± 0.04	very poor
2	3	10	24.11		19.85		very poor
7	1	8	19.52		19.99		very poor
7	2	8	19.06	18.44 ± 1.48	20.00	19.89 ± 0.18	poor
7	3	8	16.76		19.69		poor
14	1	7	17.48		20.13		very poor
14	2	7	3.91	13.01 ± 7.88	17.62	19.29 ± 1.45	fair
14	3	7	17.64		20.13		very poor
28	1	6	4.69		17.04		fair
28	2	6	20.39	**9.85** ± 9.13	20.32	**18.37** ± 1.73	very poor
28	3	6	4.46		17.74		fair

5.2 Stage 2: Number of Ancillary Qubits

Building on the insights gained from the initial experiments, which indicated that the algorithm generally achieves better performance with smaller patches, for the second stage of experiments I focused exclusively on configurations with 14 and 28 patches, with corresponding patch sizes of 56 and 28 pixels respectively, as shown in Table 4, together with quantitative results. This stage was aimed at testing whether increasing the number of ancillary qubits improves performance in order to select the best configuration for subsequent experiments. The hyperparameters for stage 2 experiments are shown in the Appendix, Table 14.

Summary - Ancillary qubits

Stage 2 tested the effect of varying the number of ancillary qubits, from 1 to 3. A greater number of ancilla led to worse performance scores. The best scores were obtained with only 1 ancillary qubit, with an average FID of 6.95 ± 2.97 and an average EMD of 18.22 ± 0.87.

Table 4: *Stage 2*: evaluation metrics. Qubits refer to the data qubits plus ancilla. FID and EMD are the calculated distances between generated images and test set, with corresponding group mean and standard deviation. Refer to Table 14 in the Appendix for complete overview of hyperparameters.

Ancilla	Qubits	Patches	FID	FID $\bar{x} \pm \sigma$	EMD	EMD $\bar{x} \pm \sigma$	Visual
1	7	14	5.15		18.39		poor
1	7	14	12.34		19.62		very poor
1	7	14	4.60	**6.95** $\pm$ 2.97	18.14	**18.22** $\pm$ 0.87	fair
1	6	28	5.50		17.61		poor
1	6	28	5.61		18.48		poor
1	6	28	8.50		17.08		fair
2	8	14	10.14		19.59		poor
2	8	14	4.01		18.28		fair
2	8	14	6.37	7.76 $\pm$ 3.82	18.82	18.69 $\pm$ 0.98	fair
2	7	28	10.44		18.74		poor
2	7	28	12.48		19.70		fair
2	7	28	3.10		17.03		fair
3	9	14	17.63		20.11		very poor
3	9	14	6.22		18.50		fair
3	9	14	17.62	13.79 $\pm$ 7.00	20.13	19.56 $\pm$ 0.97	very poor
3	8	28	3.66		18.16		fair
3	8	28	20.47		20.32		very poor
3	8	28	17.13		20.16		very poor

5.3 Stage 3: Action Weights

Based on the results from previous iterations, for the third stage of experiments I kept the generator hyperparameters fixed at 1 layer, 28 patches, 5 data qubits, and 1 ancillary qubit. This ultimate configuration choice was motivated by two reasons. Firstly, it is the configuration with

the most confined search space for the evolutionary algorithm, given that it has the least number of total qubits. Secondly, and perhaps for this very reason, it was also the configuration that obtained better FID, EMD, and visual inspection scores overall. Having established the general generator architecture, subsequent stages focused on improving performance by tuning evolutionary parameters. Specifically, in stage 3 I looked at different combinations of action weights. Considering that there are four weights (Add, Delete, Swap, Mutate) whose sum must be 100, the number of possible combinations, even partitioning the space in increments of 5 or 10, is quite high, making it infeasible to test them all with the allocated time and resources. Given such constraints, it was necessary to take a decision about which weight combinations to test, restricting the number of total combinations. Therefore, I initially partitioned the hyperparameter space with 4 different weight combinations using Latin hypercube sampling (LHS) to cover as much explored space as possible, but limiting the number of run experiments as much as possible. The results from this preliminary exploration are included in the Appendix, Table 12. The main outcome of the tests indicated that combinations with greater addition produced the best EMD and FID scores. Therefore, in this stage of experiments I decided to mainly prioritize the addition weight. I also tested some combinations with slightly greater mutation values, given its role as an optimizing action during evolution. Overall, although this approach allowed to achieve a more rapid evolution towards a more densely populated circuit, it is important to keep in mind that it limited a full exploration of the search space. Therefore, future work is needed to better understand the role of each weight in the evolutionary process. Overview of results for the different weight configurations are shown in Table 5. The metrics suggest 70, 5, 5, 20 for the Add, Delete, Swap, and Mutate actions respectively as the best performing configuration, although the difference between groups is not statistically significant. Nonetheless, this setup was chosen for subsequent experiments due to its lower mean and standard deviation compared to the others. The approach favors exploration through addition, includes some exploitation through mutation, and minimizes deletion and swap actions.

Summary - Action weights

Different combinations of the 4 action coefficients (Add, Delete, Swap, Mutate) were tested. An initial exploration of the space based on LHS partitioning indicated that the Addition coefficient should be prioritized. Seven combinations prioritizing addition were then tested. The combination that led to the lowest distance metrics was 70, 5, 5, 20 for the coefficient of the Add, Delete, Swap and Mutate actions respectively, with an average FID of 2.56 ± 0.86 and an average EMD of 16.48 ± 0.57.

Table 5: *Stage 3*: evaluation metrics. The first column shows the action weights (Add, Delete, Swap, Mutate) used for each experiment. FID and EMD are the calculated distances between generated images and test set, with corresponding group mean and standard deviation. Refer to Table 15 in the Appendix for an overview of the used hyperparameters.

A, D, S, M	**FID**	**FID** $\bar{x} \pm \sigma$	**EMD**	**EMD** $\bar{x} \pm \sigma$	**Visual**
60, 10, 10, 20	8.50		17.08		fair
60, 10, 10, 20	5.50	6.54 ± 1.70	17.61	17.72 0.71	poor
60, 10, 10, 20	5.61		18.48		poor
60, 5, 5, 30	2.79		17.38		fair
60, 5, 5, 30	6.51	4.91 ± 1.91	18.37	17.55 ± 0.75	poor
60, 5, 5, 30	5.41		16.89		fair
65, 5, 5, 25	12.04		18.54		poor
65, 5, 5, 25	3.28	5.95 ± 1.70	17.36	17.56 ± 0.90	fair
65, 5, 5, 25	2.54		16.77		fair
70, 5, 5, 20	1.68		15.88		fair
70, 5, 5, 20	2.60	**2.56** ± 0.86	17.01	**16.48** ± 0.57	fair
70, 5, 5, 20	3.40		16.55		fair
75, 5, 5, 15	9.78		17.70		fair
75, 5, 5, 15	6.10	6.30 ± 3.39	18.49	17.90 ± 0.52	poor
75, 5, 5, 15	3.02		17.51		fair
80, 5, 5, 10	3.49		17.25		poor
80, 5, 5, 10	3.62	3.34 ± 0.38	17.29	17.22 ± 0.09	fair
80, 5, 5, 10	2.90		17.12		fair
85, 5, 5, 5	10.92		18.72		poor
85, 5, 5, 5	3.49	5.58 ± 4.66	17.72	17.79 ± 0.89	fair
85, 5, 5, 5	2.34		16.94		fair

5.4 Stage 4: Multi-Action Probability

Once established the weight combination, stage 4 was aimed at exploring how the evolutionary algorithm performs with different multi-action probability values. Also in this case, because of time constraints I could not do a full parameter search, so I limited testing for relatively low values: 0.1, 0.2, and 0.3. Results for the fourth stage are displayed in Table 6, while hyperparameters are listed in the Appendix, Table 16.

Summary - Multi-Action probability

Three values of multi-action probability were tested: 0.1, 0.2, and 0.3. Larger MA-probability resulted in lower average distance scores, with a value of 0.3 leading to the lowest average FID and EMD values, 3.81 ± 0.81 and 17.39 ± 0.24 respectively.

Table 6: *Stage 4*: evaluation metrics. The first column shows the tested values of the multi-action probability. FID and EMD are the calculated distances between generated images and test set, with corresponding group mean and standard deviation. Refer to Table 16 in the Appendix for complete overview of hyperparameters.

MA prob.	FID	FID $\bar{x} \pm \sigma$	EMD	EMD $\bar{x} \pm \sigma$	Visual
0.10	9.78		17.70		fair
0.10	6.10	6.30 ± 3.39	18.49	17.90 ± 0.52	poor
0.10	3.02		17.51		fair
0.20	3.68		18.03		fair
0.20	3.78	4.62 ± 1.54	18.00	17.79 ± 0.38	fair
0.20	6.39		17.35		fair
0.30	4.72		17.14		fair
0.30	3.19	**3.81** ± 0.81	17.41	**17.39** ± 0.24	poor
0.30	3.53		17.62		fair

5.5 Stage 5: increased generations and epochs

After determining the optimal architecture (stages 1 and 2) and evolutionary hyperparameters (stages 3, 4), in Stage 5 I conducted a more in-depth analysis of the best found configuration by extending the algorithm's runtime for both the evolutionary component and GAN training. To this end, I increased the maximum number of evaluations for the evolutionary algorithm to $60,000$, with 15 offspring per generation, resulting in a potential maximum of $4,000$ generations. This is a very high number, and it was never actually reached during implementation, with an average number of generations of 922.5 for the 10 runs. However, I chose to keep it so high as I wanted to ensure the evolutionary algorithm had abundant time for exploring of the solution space. At the same time, to avoid unnecessarily long runtimes and a waste of resources, while still ensuring sufficient search time, other termination criteria were also adjusted. Specifically, the maximum number of generations without improvement was set to 500, after which the evolution would terminate if no improvement was found. Additionally, the total number of GAN epochs was set to 40, which was found to be excessive as the training plateaued much earlier perhaps due to the small size of the generator ansätze. The reason for increasing these parameters was to

ensure the evolutionary process had sufficient time to discover interesting configurations and to verify that previous results were not hindered by premature termination, as well as confirming the approximate number of generations and epochs required for the configuration to achieve optimal performance. The experiment was repeated 10 times to enhance statistical significance; results are shown in Table 7.

Summary - Increased run time

Stage 5 increased the number of generations during evolution and the number of GAN epochs, to identify architectures with lower FID and EMD scores. The best-found scores were a FID of 2.07 for architecture F_04 and an EMD of 15.68 for architecture F_08. However, this improvement is not guaranteed, as there were some instances where the algorithm did not perform well despite the increased run-time.

Table 7: *Stage 5*: evaluation metrics. The table shows FID and EMD between generated images and test set, number of found parameterized gates and CNOT gates, and total number of generator parameters. Refer to Table 17 in the Appendix for complete overview of the hyperparameters.

ID	FID	EMD	EMD (evol)	$U(\theta, \phi, \lambda)$	CNOT	Params	Visual
F_01	3.12	17.38	2.76	10	6	840	fair
F_02	2.63	17.29	2.61	6	5	504	poor
F_03	6.85	18.10	2.78	3	4	252	poor
F_04	**2.07**	15.79	2.53	9	8	756	good
F_05	2.52	17.09	**2.37**	8	9	672	poor
F_06	6.93	17.80	2.69	4	8	336	fair
F_07	5.90	16.66	2.80	7	7	588	fair
F_08	6.24	**15.68**	2.52	5	7	420	fair
F_09	22.46	19.82	2.92	5	8	420	poor
F_10	3.40	17.68	2.88	8	1	672	poor

5.6 Stage 6: Best architecture with increased layers

Finally, the last stage of experiments utilized the best-performing architecture identified in Stage 5 as a layer in a multi-layer generator. Specifically, the ansatz was repeated two or three times to test how performance changes with the repetition of the same ansatz structure. The repetition causes an increase in gates and number of trainable parameters, therefore enhancing expressivity.

Summary - Increased layers

Stage 6 increased the number of layers of architecture F_04, an already well-performing single-layer architecture identified in stage 5. This resulted in similar performance scores and improved visual output quality.

Table 8: *Stage 6*: evaluation metrics. FID and EMD scores, visual label, and total number of trainable parameters for the ansätze obtained by layering 2 or 3 times architecture F_04 from Stage 5.

ID	Layers	FID	EMD	Params - 1L	Tot. params	Visual
F_04_L2	2	2.970	15.241	756	1512	good
F_04_L3	3	3.448	15.516	756	2268	good

5.7 Sample Efficiency

To test the efficiency of the various metrics and benchmark the models, I evaluated the performance of both FID and EMD on different sample sizes and models. Specifically, I computed the distance between disjoint sets of real images sampled without replacement, and compared it with the distance between a set of real images and images generated by a fully classical GAN, the hybrid classical quantum PQWGAN model with default parameters [20], and my two best performing models from stage 6. Table 9 and 10 contain test results for FID and EMD respectively. A visual display of results is provided in Figure 7. The results show that the metrics are robust on all sample sizes, with the distance between the two sets of real images being lower for all sample sizes for FID, and for all but $N = 20$ for EMD, where the images generated classically performed slightly better.

Summary - Sample efficiency

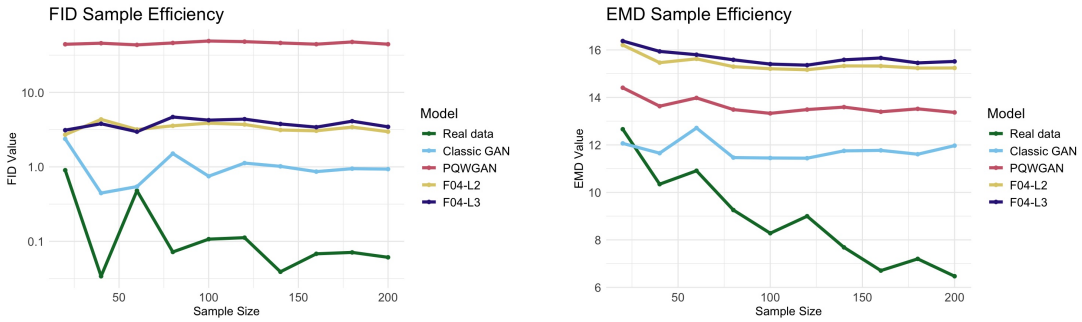
Both FID and EMD proved to be effective metrics with low sample efficiency, capable of distinguishing between real images and images generated by various models (including a fully classical GAN) across different sample sizes.

Table 9: Sample efficiency for *FID score* calculated on various sample sizes. The distance is calculated between a set sampled without replacement uniformly from the real distribution and **a**) Another, disjoint sample of real data, **b**) A sample of images generated by a fully-classic WGAN-GP network **c**) A sample of images generated by the PQWGAN model with default parameters and **d**) and **f**) A sample of generated images from my two best found models.

N	Real	WGAN-GP	PQWGAN	F_04_L2	F_04_L3
20	**0.902**	2.375	44.295	2.705	3.110
40	**0.034**	0.445	45.758	4.340	3.796
60	**0.479**	0.543	43.462	3.176	2.968
80	**0.072**	1.508	46.133	3.561	4.674
100	**0.107**	0.749	49.002	3.865	4.240
120	**0.112**	1.124	48.044	3.710	4.374
140	**0.039**	1.017	46.145	3.125	3.771
160	**0.068**	0.863	44.391	3.051	3.415
180	**0.071**	0.946	47.597	3.424	4.107
200	**0.061**	0.933	44.378	2.970	3.448

Table 10: Sample efficiency for *EMD score* calculated on various sample sizes. The distance is calculated between a set sampled without replacement uniformly from the real distribution and **a**) Another, disjoint sample of real data, **b**) A sample of images generated by a fully-classic WGAN-GP network **c**) A sample of images generated by the PQWGAN model with default parameters and **d**) and **f**) A sample of generated images from my two best found models.

N	REAL	WGAN-GP	PQWGAN	F_04_L2	F_04_L3
20	12.660	**12.064**	14.407	16.207	16.376
40	**10.346**	11.652	13.631	15.464	15.938
60	**10.911**	12.713	13.979	15.624	15.799
80	**9.257**	11.466	13.488	15.296	15.584
100	**8.280**	11.448	13.329	15.208	15.404
120	**8.997**	11.440	13.488	15.165	15.359
140	**7.685**	11.749	13.592	15.326	15.584
160	**6.706**	11.770	13.395	15.322	15.661
180	**7.200**	11.608	13.518	15.235	15.454
200	**6.469**	11.968	13.368	15.241	15.516



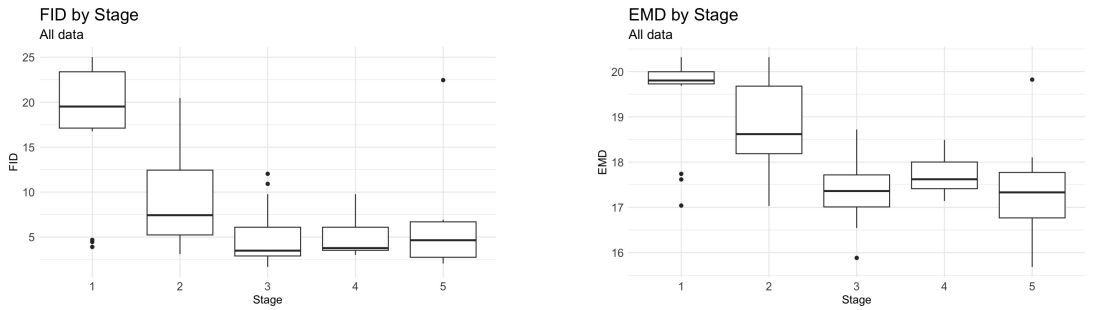
(a) Sample efficiency of FID for different sample sizes and models.

(b) Sample efficiency of EMD for different sample sizes and models.

Figure 7: **a**) FID and **b**) EMD sample efficiencies for different sample sizes. We see that the metric are mostly efficient for all sample sizes, with real-real distances being lower than real-generated. The model that performs best across the board is the fully classic GAN, followed by the two best models identified in this research for FID, or by PQWGAN [20] for EMD. Note that the FID y-scale is logarithmic.

6 Discussion

The aim of this study was to evaluate the effectiveness of the QES strategy in identifying optimal ansätze to be used as generators in a hybrid quantum GAN. Through a series of experiments, different architectural configurations and hyperparameters were tested, to determine their impact on the performance and quality of the generated outputs. Overall, the results of the pipeline improved progressively through each experimental stage, both in terms of quantitative (EMD and FID) and qualitative (visual inspection) evaluation metrics, as shown in Figure 8, indicating that tuning the hyperparameters and the architectural structure was both necessary and beneficial for improving performance. However, the algorithm generally struggled to find ansätze capable of generating high-quality images efficiently, and not all tested hyperparameters showed a significant impact on model performance. The high stochasticity of the results makes this method unpredictable and results at times difficult to interpret, at least at the current level of development and tuning. Some of the found solutions were able to produce images of comparable quality to those generated by the PQWGAN [20], but only after layering the architecture 2-3 times, as done in the sixth experimental stage. This contrasts with the tested hypothesis that the algorithm could discover an ansatz with inherent depth and superior performance to an hardware-efficient ansatz in terms of visual quality of the output or network efficiency.



(a) FID values by experimental stage.

(b) EMD values by experimental stage.

Figure 8: The general decreasing trend of FID and EMD values over the various experimental stages, showing the improvement in algorithm performance consequent of architecture structure and hyperparameter tuning.

A significant limitation faced during the development, testing, and experimental phases was the restricted and intermittent access to computational resources, which allowed for only simulating small systems, posing a limit on the number of runs and iterations that could be performed for each tested configuration. As a result, it was challenging to achieve sufficient repetitions for robustness. For analogous reasons, I could not fine-tune all GAN and evolutionary hyperparameters, and I focused only on a few while keeping others fixed. The time constraint was another significant challenge, as each experiment took between 5 and 100 hours, with a mean of 18.5 hours per run, depending on the number of evolutionary generations, GAN epochs, complexity of the identified ansatz, and computing hardware, amounting to more than 1000 hours of combined CPU and GPU time. For these reasons, the results should be considered preliminary explorations, as some runs could only be repeated 1-3 times, and not all hyperparameters could be properly tuned.

With these limitations in mind, in the following subsections I discuss the findings reported in Section 5. These include the effects of hyperparameter tuning and why certain configurations

proved superior to others, the limitations of the evolutionary algorithm and potential areas for improvement, as well as insights into what constitutes an effective ansatz structure.

6.1 Number of layers

Within this research's methodology, testing with multiple layers consisted into repeating the ansatz architecture found by the evolutionary algorithm several times, with each repetition being identical in type, number, and position of gates, but with the parameters of the $U(\theta, \phi, \lambda)$ being trained individually for each layer. Layering was performed both in stage 1, as a preliminary exploration of how it could affect the algorithm, and in stage 6, once better configurations were found after hyperparameter tuning and more experimental runs, in order to test if and how layering could benefit an already promising architecture. As shown in Table 2, in stage 1 layering did not have a significant effect on the FID and EMD values. Both the single factor ANOVA results and the two-way ANOVA with replication indicate that there were no statistically significant differences between the FID scores of different layer configurations, suggesting that layering did not contribute to improving the quality of the generated images during this stage of the experiments. After verifying this, subsequent experiments did not use any layering up until stage 6.

Although ideally the evolutionary algorithm should find deep enough architectures on its own, such that layering would not be required, this was not found to be the case. In all tested configurations, the maximum depth ever reached was 10. In stage 5, when the evolutionary was run for longer, the average circuit depth was 6.4, which however only refers to the maximum depth for a qubit, and does not imply all qubits had those many gates. In fact, with an average of 6.5 $U(\theta, \phi, \lambda)$ gates and 6.3 CNOT gate spread over 6 qubits, each qubit had on average a little over 2 gates. This is not enough depth and gates for the network to be able to learn truly meaningful representations, and explains why layering promising architectures improved performance in stage 6. Therefore, layering the found ansatz can be advantageous and bring improvements in output quality, but only in the case where the found architecture already shows some learning ability and capacity of generating meaningful data.

Summary - Number of Layers

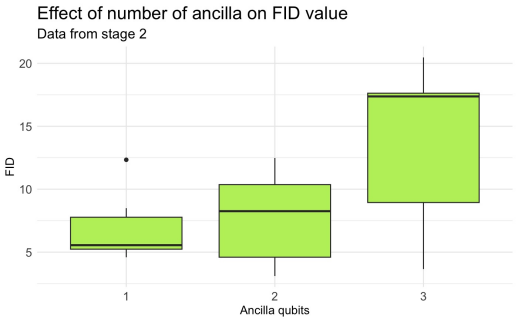
Layering the found ansatz can be advantageous and bring improvements in output quality, but only in the case where the found architecture already shows some learning ability and capacity of generating meaningful data.

6.2 Number of patches and qubits

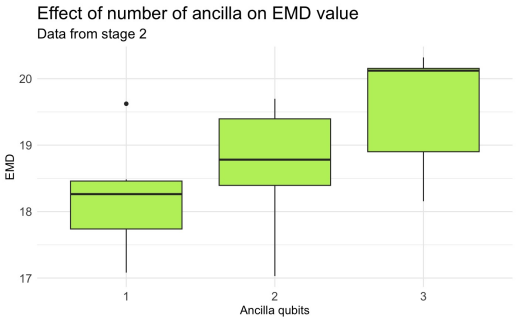
Results from the first stage show that the number of patches has an impact on algorithm performance, with a one-way ANOVA showing significant difference in mean FID ($F(4,1) = [3.4]$, $p = 0.03$) for different patch numbers, and 28 patches configurations resulting in the best FID and EMD scores on average, as shown in Table 3. As patches get smaller, the algorithm proposes architectures with more parameterized gates, and therefore more trainable parameters for the generator. Moreover, the number of parameters not only increases due to the positive correlation between number of patches and number of gates ($r = 0.63$, $p = 0.01$), but also because the more patches, the more sub-generators there will be for image generation, and hence, given equal number of gates per circuit, the more total parameters. This results in a positive correlation between

total number of parameters and number of patches ($r = 0.71$, $p = 0.003$), and, because there is a negative correlation between the number of total parameters and performance as measured in FID ($r = -0.64$, $p = 0.01$) and EMD (-0.55, $p = 0.03$) scores, a greater number of patches result in lower FID ($r = -0.75$, $p = 0.001$) and EMD ($r = -0.55$, $p = 0.03$) values, i.e., better performance, from the data of stage 1 experiments.

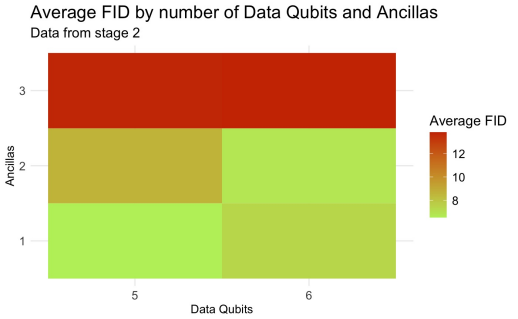
The reason why the algorithm identifies more gates in configuration with smaller patches could be due to the algorithm's superior capability to explore smaller search spaces associated with fewer qubits ansätze, leading to finding more effective configurations.



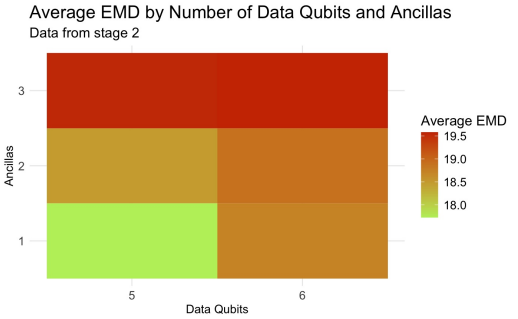
(a) The effect of the number of ancilla qubits on FID value.



(b) The effect of the number of ancilla qubits on EMD value.



(c) FID values by number of ancilla qubits (y axis) and data qubits (x axis).



(d) EMD values by number of ancilla qubits (y axis) and data qubits (x axis).

Figure 9: FID and EMD values in relation to number of ancilla and number of both ancilla and data qubits from stage 2. Larger FID and EMD values correspond to worse output quality.

In terms of ancillary qubits, stage 2 results show a similar trend (Table 4) with a two-way ANOVA showing no statistically relevant interaction between the effects of number of patches (only 14 and 28 tested in this stage) and ancilla on FID and EMD, but simple main effects analysis show a statistically significant effect of ancilla on FID values, and both of ancilla and number of patches on EMD values, with a greater number of ancillary qubits related to a worsening in performance. Configurations with 3 ancillary qubits exhibited greater FID and EMD scores compared to configurations with fewer ancillary qubits, given the same number of data qubits. Figure 9 shows the inverse relationship between quantitative metric scores and number of ancillary and data qubits.

Overall, a greater number of qubits, and particularly of ancilla qubits, leads to a worsening in performance. This might be due to an unnecessary complexity introduced in the circuit, inefficient utilization of limited resources, and interference of the ancilla with the GAN training

process.

Summary - Number of Qubits

A greater number of patches, corresponding to fewer data qubits per sub-generator, was found to correlate with better performance scores. On the other hand, increasing the number of ancillary qubits led to a worsening in performance, particularly for FID scores.

6.3 Action weights

The combination of action weights tested prioritized addition and mutation over swap and delete actions. This choice was motivated by an initial exploration of each weight's role during evolution, which showed that addition led to the best performance, as well as the theoretical motivation that addition and mutation can most effectively build up the circuit from an empty state and optimize it over the generations. In fact, the two actions favor exploration via introducing new gates (addition) and exploitation via the monotonic optimization of existing gates by changing their parameters (mutation). The single factor ANOVA on stage 3 results indicate a lack of significant difference between weight groups, indicating that the tested weight ratios do not lead to a difference in performance of the algorithm, as measured in FID and EMD scores. However, given that a weight had to be chosen for subsequent experimentation, the configuration [70, 5, 5, 20] was the one picked being the group that displayed the best balance between good performance, with a low average FID (2.56 ± 0.86) and EMD (16.48 ± 0.57) with relatively low standard deviation, and medium visual scores, indicating more consistency and predictability of results. These observations, however, should all be interpreted with caution due to the very small sample size of $n = 3$.

Future work should investigate more on the appropriate weights for the algorithm, increasing sample size and exploring how the algorithm behaves with larger swap and deletion weights as well. Another interesting approach to test, once the algorithm's dynamic in relation to the action weights is better understood, is a dynamic adjustment of weights as the circuit evolves, for example, in case of an empty-circuit initialization, starting with a larger addition weight and gradually decreasing it while increasing the probabilities of mutation, swapping, and deletion as the circuit becomes more complex. Finally, it would also be interesting to see how a different initialization set-up, for example with a partially populated circuit, affects the evolutionary process and the optimal weight balance.

Summary - Action weights

The results showed no significant differences between the tested combinations of action coefficients, suggesting that the weights might not strongly impact performance. Further experiments are needed to better understand the effects of different action weights on the algorithm's behavior. The coefficients 70, 5, 5, 20 for the actions Add, Delete, Swap, Mutate respectively led to the lowest EMD and FID on average, and were therefore selected as the best combination for subsequent experiments.

6.4 Multi-action Probability

Single factor ANOVA test on stage 4 experiments (Table 6) does not show statistical significance in algorithm performance between multi-action probability values of 0.1, 0.2, and 0.3. However, the value of 0.3 produced the best FID 3.81 ± 0.806 and EMD 17.39 ± 0.24 scores with lowest variance, indicating more consistency in results. This might indicate that the evolution generally benefits for more dynamism; the higher multi-action probability may accelerate the search process by introducing more changes to the circuit, which is advantageous when the circuit is initially empty and non-functional. Therefore, despite the lack of statistical significance, the value of 0.3 was chosen for subsequent experiments.

More extensive testing is needed to verify the effect of the hyperparameter across the complete range of its values, as well as exploring dynamic adjustment of the value as the ansatz becomes more and more populated.

Summary - Multi-Action probability

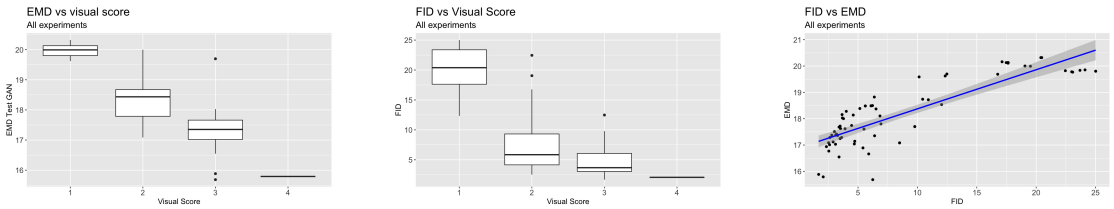
Multi-action probability increases the likelihood of multiple actions being taken for each offspring. Three values of the hyperparameter were tested: 0.1, 0.2, 0.3. Although the differences between groups are not statistically significant, higher MA-probability values resulted in lower average distance scores.

6.5 Evaluation Metrics

The metric used for performance evaluation are the Earth Mover Distance (EMD), both for the evolutionary output and the final GAN output, and the Fréchet inception distance (FID) only for the final GAN output, all calculated on test sets. For this work, FID was calculated on the second max pooling layer features of the Inception net, to adapt the metric to the much smaller images and simpler dataset. Note that the Earth Mover distance was also used as the fitness function for the evolutionary search, and an estimate of the distance (the critic net score) is what guided the GAN in the training. For this particular task, the estimate, i.e., the critic score, performed well in guiding the GAN towards improvement in the quality of the output over time, in the cases when an expressive and high-quality enough generator was provided.

A general agreement is observed among all metrics across the entire dataset ($n = 73$). Visual inspection scores, assigned on a 5-steps scale from '*very poor*' to '*excellent*', highly correlate both with FID scores ($r = -0.77$, $p < 0.01$) and EMD scores ($r = -0.84$, $p < 0.01$) on the Pearson correlation test. This indicates that both metrics well agree with human perception. There is also a strong positive correlation between the FID and EMD themselves ($r = 0.85$, $p < 0.01$), which further supports their robustness at evaluating the quality of generated images against real ones, confirming their suitability for the application. The relationship among FID, EMD, and visual scores is displayed in Figure 10.

For the EMD calculated on the training set after evolution, there were positive but not as strong correlations between it and the visual score, FID, and EMD score after GAN training, $r = -0.60$, $r = 0.77$, $r = 0.54$, respectively, all $p < 0.01$, $n = 63$. These results indicate that although the function performs reasonably well at guiding the algorithm during evolution, there is room for improvement in its effectiveness, and its values cannot always be trusted to predict a good outcome during GAN training. Moreover, the EMD calculated on the test set after evolution is only correlated ($r = 0.66$, $p = 0.04$) with the EMD at the end of GAN training,



(a) EMD vs. Visual Score. **(b)** FID vs. Visual Score. **(c)** EMD vs. FID

Figure 10: Relationship on the whole dataset between: **a**) EMD values and assigned visual scores, **b**) FID values and assigned visual scores, and **c**) EMD and FID values, all calculated between real images and the test set. Being distance metrics, both EMD and FID have better scores when lower, while visual scores are in the scale 1 = 'very poor', 2 = 'poor', 3 = 'fair', 4 = 'good'. No output was assigned the score 5 = 'excellent'.

but the correlation is non-significant with FID and visual inspection scores, suggesting that the relationship between EMD (evolutionary) and FID and visual scores is less robust.

I attribute the reason behind the discrepancy in EMD performance between the two algorithm's segments, evolution and GAN training, mainly to a structural issue of how the value is calculated. In fact, while the EM distance for the GAN is computed between entire images, i.e., a test set of real images and an equally numerous set of generated images, during and after evolution the EMD for training and testing is calculated on a patch level. That means, if the configuration tested had only one patch, i.e., a single ansatz generating the whole image, then there would be no difference in the way EMD is calculated during evolution and during GAN. However, if, like for most tested configurations, the images are divided in patches, then the EMD during evolution is only calculated on a patch level. The evolutionary generates a set of patches for each offspring, for example 100 images of size 1×28, and these are compared to the same number of real images: 100 patches of size 1×28, each sampled at random from the images in the test set. Now, for large enough patches this should not be an issue, but for small patches of 1 or 2 rows, the EMD is limited because it lacks contextual information from neighboring pixels.

Looking at EMD's behavior over generations, shown in Figure 11 for the 5th experimental stage, we see that the score rapidly improves during the first few generations, until hitting a plateau where the algorithm stagnates, after which evolution is terminated due to lack of found improvement over hundreds of generations. What is happening is that in the initial stages of evolution the algorithm is quick to populate the empty circuit, encouraged by the large addition weight, but then fails to optimize it beyond a certain point. If we observe the generated patches, they all start from being black (empty circuit) to having more shades of gray pixels once the circuit is populated. Then, the cost function plateaus, suggesting that EMD can distinguish black patches (generated) from grayscale ones (real) in the early stages of the evolution, but, once the generated patches reach a certain degree of diversity in grayscale values, the function becomes meaningless. The supposition, not tested, is that EMD would perform better on larger patches.

However, despite these observations, in these specific experiments the smaller patches were the ones that obtained the better final results, perhaps because the algorithm still managed to better handle the smaller search space of the fewer qubits configurations. The smaller patches also have another advantage: not only, on average, they were the configurations with more trainable gates and hence more expressive power. But also, in case of smaller patches, each generator is repeated more times, so the total number of trainable parameters available to the GAN increases between 2-fold and 28-fold.

A future improvement, that was not tested in this research because it would have been computationally prohibitive with the given resources, would be to combine the superiority of the small patch division in generating ansätze, with the supposed better robustness of EMD scores on larger patches, by keeping the generator structure as it is, divided in small patches, but calculating the EMD during evolution on entire images. This would require generating sets of whole images to be compared to the training data either for each off spring, or every X generations.

It is also worth noting that an attempt was done at using alternative metrics for the circuit evaluation during evolution, namely Inception Score and FID, but they proved to be too expensive to calculate at each iteration, and so I abandoned this approach. Moreover, it is not guaranteed that such metrics would have performed better on small patches, as the issue with having too little information to compare persists independently on the metric used.

In conclusion, although with limitations, the EMD calculated on small patches was the best found compromise between computational affordability and performance. While, with the given resources, the choice of calculating EMD on small patches was a necessary one, future implementations should look into ways to improve the accuracy and robustness of the calculation, either by modifying the patch configuration, or modifying the way the cost is computed.

Summary - Evaluation metrics

A general agreement is observed between visual inspection, EMD, and FID scores calculated on the final images generated by the algorithm, indicating the suitability of these metrics for the task. The EMD is also used as the fitness function to guide evolutionary search. While the function is capable of guiding the search towards better configurations, it shows some limitations in performance, likely due to it being calculated on single patches only, to save computing resources.

6.6 Ansatz architecture

There is a moderate negative correlation between the number of total generator parameters - which are proportional to the number of $U(\theta, \phi, \lambda)$ gates in the ansatz, the number of layers, and the number of patches- and the FID ($r = -0.55$, $p < 0.001$) and EMD scores ($r = -0.54$, $p < 0.001$) of the circuits. This indicates that having more $U(\theta, \phi, \lambda)$ gates generally enhances circuit performance, which aligns with the intuition that $U(\theta, \phi, \lambda)$ gates, being the trainable gates, allow for greater optimization and network expressivity during GAN training. However, number of gates is not the only factor affecting performance, as there are also cases of circuits with similar or equal number of total trainable parameters but difference in performance, such as F_10 (672 parameters, FID=3.402, EMD=17.682, visual = good) and F_05 (672 parameters, FID=2.518, EMD=17.087, visual = poor) from stage 5, suggesting that while a greater number of trainable parameters is useful for enhancing performance, the position of the gates in relation to other gates and in the ansatz also plays an important role.

While running a qualitative comparison of different architectures from 28 patches configuration, of which I display an example in Figure 13, having selected the best and worst performing circuits from each round, we can note some common characteristics and general observations. The best circuits have a generally balanced distribution of U and CNOT gates, leaning towards more alternation of these gates. There is also a more even distribution and spread of gates between the qubits. The worst circuit instead are more likely to have less parameterized gates and

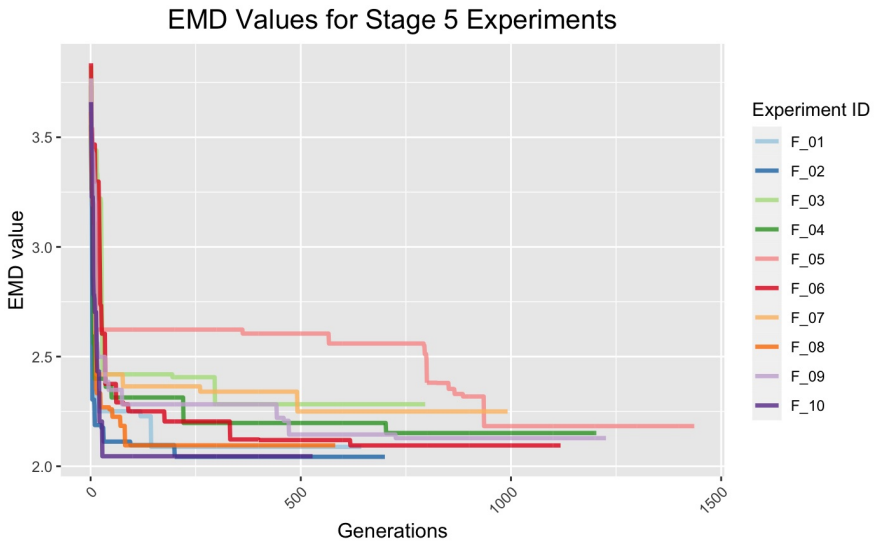


Figure 11: EMD values during evolution for stage 5 of experiments.

Summary - Ansatz architecture

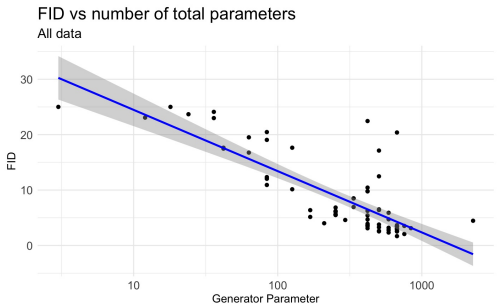
A greater number of trainable gates $U(\theta, \phi, \lambda)$ is associated with improved performance. The positioning of these gates within the circuit and in relation to other gates also impacts performance. The better-performing circuits tend to have a balanced mix of CNOT and trainable gates, an alternating pattern of these gates, and a more even distribution across qubits.

a greater number of CNOTs. There is also a less uniform spread of the gates between qubits, with higher intra-circuit depth imbalance. Therefore, a general conclusion is that ansätze with a simple and clean structure, featuring clear layering and a balanced combination of $U(\theta, \phi, \lambda)$ and CNOT gates, where $U(\theta, \phi, \lambda)$ gates are slightly more prevalent, are more likely to perform better as generators of a hybrid GAN.

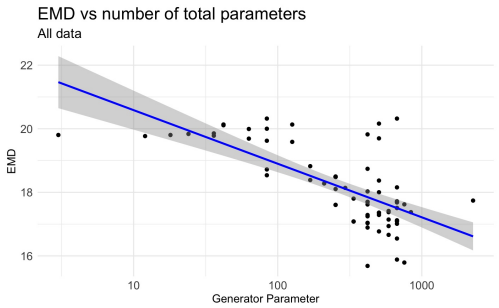
6.7 Final results and benchmarking

Despite the mentioned limitations, some circuits identified by the evolutionary algorithm showed potential for generating high-quality images. In stage 6, the best performing ansatz from stage 5, shown in Figure 14, was first layered with two or three layers, and then, each architecture was trained for a greater number of epochs. The experiments achieved relatively positive results compared to previous iterations, as they successfully generated discernible images of 0s and 1s for the first time. The layered architectures also obtained the best human perception scores, the best EMD scores, and competitive FID scores as shown in Section 5, Table 8.

Figure 15 presents examples of the output from the proposed models: 'F_04' and its layered versions 'F_04_L2' and 'F_04_L3', as well as example outputs from PQWGAN, fully classic WGAN-GP, and real sample images, for comparison. Observing the images, we see that 'F_04', the ansatz found via evolution, successfully learned a representation but lacked the expressive



(a) FID scores vs total number of generator parameters.



(b) EMD scores vs total number of generator parameters.

Figure 12: The negative correlation between FID and EMD vs number of total generator parameters, indicating that more parameters lead to better fitness measures. The x-axis scale is logarithmic.

Model	FID	EMD	Params
F_04	2.068	15.791	**756**
F_04_L2	2.970	15.516	1512
F_04_L3	3.448	15.241	2268
PQWGAN	44.378	13.368	1512
WGAN-GP	**0.933**	**11.968**	1,481,232

Table 11: Metric scores and number of parameters for the evolutionary generated model 'F_04', its 2 and 3 layer versions 'F_04_L2' and 'F_04_L3', PQWGAN, and WGAN-GP.

power to fully capture the target dataset, mainly generating low-quality digits resembling a 1. Its 2-layers and 3-layers versions 'F_04_L2' and 'F_04_L3' showed significant improvement, successfully generating clear images of 0 and 1 digits with moderate intra-class variance. In comparison, the PQWGAN [20] model, that has a comparable number of parameters, generated images where 0s and 1s were still partially distinguishable but more blurry and noisy, with significant inter-class overlap. The fully classical net outperformed both, generating images closely resembling the real ones, but with a parameter count three orders of magnitude greater than the quantum models.

The FID and EMD values mostly aligned with this visual analysis. The classical model scored best in both metrics. For EMD, it was followed by PQWGAN and then the three proposed models. However, for the FID metric, the proposed models performed significantly better than PQWGAN, perhaps due to FID being more sensitive to the noisiness of the images. It is possible that a simple denoising operation would bring PQWGAN's FID value back to competitive levels. This noisiness and difference in distribution reflected in the FID values can be seen looking at the histograms of the pixel values distribution, presented in Figure 16. We see that PQWGAN images have much higher noise levels and a more dispersed distribution across pixel values, while the other quantum models and the classic WGAN-GP better captured the strong bimodal distribution of real images, with most pixel values concentrated at 0 and 1, predominantly featuring black and white images with sharp edge transitions. Therefore, while the FID score may suggest that the evolutionary algorithm identified an ansatz structure that, when layered, outperforms the current state of the art, this conclusion should be interpreted within the specific

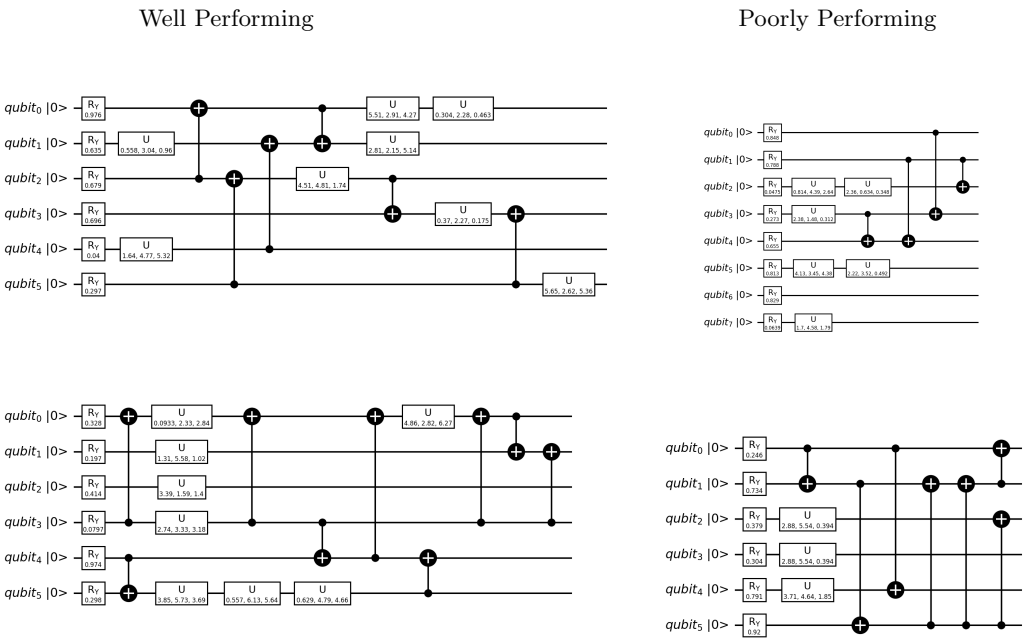


Figure 13: Some examples of well performing and poorly performing ansätze.

context of the metric and its limitations.

Another characteristic that is common among all quantum models, and differentiates them from the classic WGAN-GP, is the presence of artifacts in the same locations of all output images, potentially due, as Tsang et al. suggest [20], to a too limited amount of data qubits, and a lack of sufficient qubits for the generators to clean-up the imperfections of the entangled output state. Another possible explanation might be the use of a uniform prior for the input latent space, that, while easier to learn a mapping from than a Gaussian, spreads the latent vectors uniformly across the space, making it more challenging to achieve high-quality images due to the lack of focus around the mean of the latent space, and therefore leading to more noise and artifacts in the generated outputs.

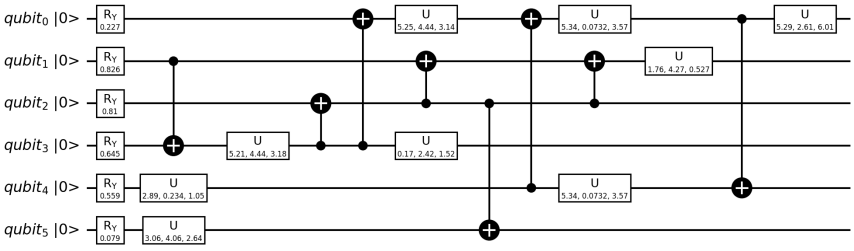
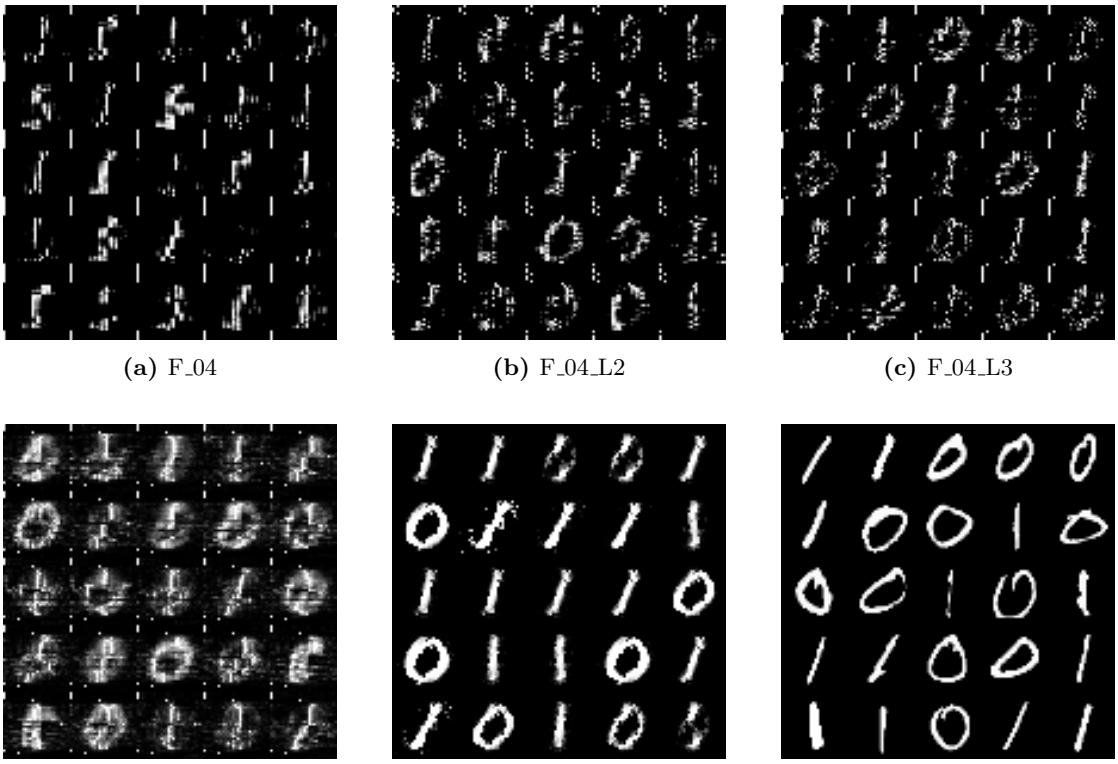


Figure 14: Best ansatz identified in the 5th stage of experiments, used as a base layer for the architectures of the final stage.



(a) F_04 **(b)** F_04_L2 **(c)** F_04_L3

(d) PQWGAN **(e)** Classic WGAN-GP **(f)** Real

Figure 15: Sample output from **a**) the best model from stage 5, the same model layered **b**) twice and **c**) three times, **d**) PQWGAN, **e**) a fully classical WGAN-GP, and **f**) real samples.

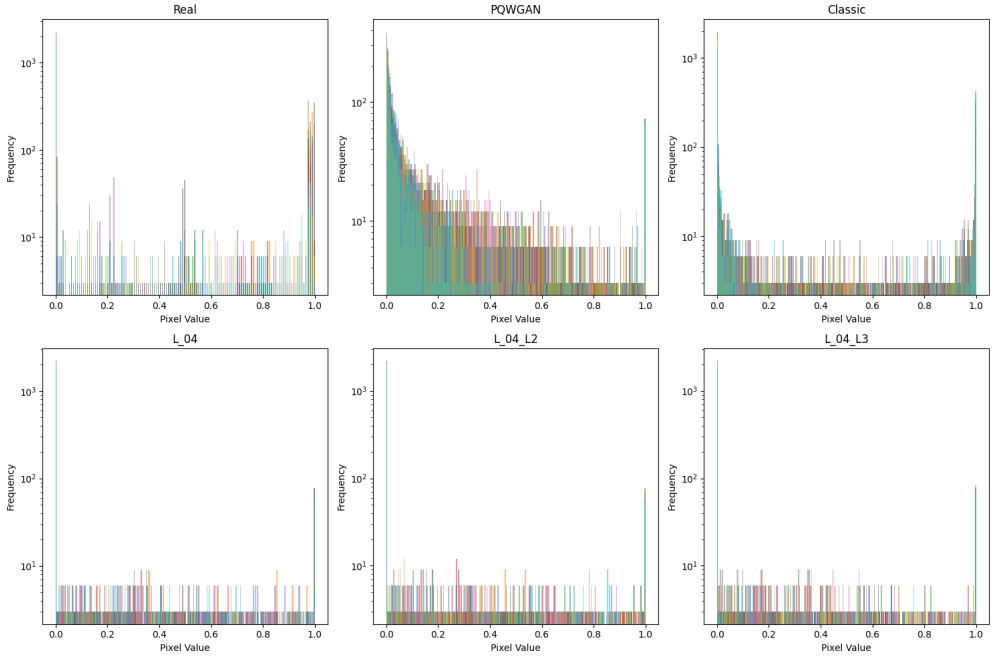


Figure 16: Histogram of pixel values for 20 generated images from various models. From left to right, top to bottom: real images, images generated with the PQWGAN net (default parameters), a fully classic WGAN-GP, and my three models 'F_04', 'F_04_L2', 'F_04_L3'. Note that the y-scale is logarithmic.

Summary - Benchmarking

Layering a well-performing single-layer architecture improved the visual quality of the output, enabling the networks to learn representations of the digits 0 and 1. The resulting images are sharper compared to those produced by the current state-of-the-art hybrid quantum GAN, PQWGAN, used as a benchmark. Although the image quality is still significantly lower than those generated by a fully classical WGAN-GP, the quantum GANs achieve this with three orders of magnitude fewer parameters. These results demonstrate the promising potential of quantum GANs to achieve comparable visual quality of the output to classical networks with a substantially reduced number of parameters.

In conclusion, guided by experimental results, the selection of better architectural configurations and fine-tuning of hyperparameters, despite the given resource constraints, led to an overall improvement in the performance of the evolutionary GAN pipeline. The evolutionary algorithm successfully identified progressively better-performing ansätze, capable of generating increasingly higher-quality images. The final results, considering both quantitative and qualitative metrics, showed that the best-found ansatz, when layered 2 or 3 times, achieved competitive results with state-of-the-art quantum architectures. Although still not matching the quality of images generated by fully classical networks, the architecture generated distinguishable 0 and 1 digits with moderate intra-class variance and significantly reduced noise compared to the state of the art, using three orders of magnitude fewer parameters than the classical architecture.

These preliminary results demonstrate the algorithm's potential in finding better architectures. Further improvements could be achieved through better parameter tuning and exploration of aspects such as changing the latent space distribution, increasing the number of data qubits, reducing the number of patches -given sufficient resources-, and refining evolutionary aspects like action weights, multi-action probability, fitness function calculation methods, and dynamic hyperparameter adjustment. These modifications could lead to significant improvements in pipeline performance and generated image quality while maintaining the current low parameter count.

7 Conclusions and Future Work

This work explored the feasibility of using a mutation-based evolutionary algorithm, based on the work of Lipardi et al. [29], for identifying and optimizing ansätze to be used as generators in a hybrid quantum-classical GAN. The GAN was designed to generate 28×28 MNIST images of digits 0 and 1, within the patch-GAN framework proposed by Huang et al. [19] and expanded by Tsang et al. [20]. The experiments aimed to identify optimal structural elements -number of patches, layers, and ancilla qubits- as well as optimal evolutionary parameters -action weights and multi-action probability- for generating the ansätze. While some hyperparameters did not significantly affect the performance of the evolutionary GAN pipeline, others, including the number of patches, qubits, and ancilla, were found to have a significant impact on at least one of the evaluation metrics. The final identified architectures, composed of the best performing ansatz layered multiple times, showed competitive performance compared to the current state of the art hybrid quantum-classic GANs both in terms of quantitative metrics and parameter complexity. While the output images are not yet comparable with those generated by traditional fully classical networks, the significant reduction in number of parameters, together with the promising performance of the networks -which demonstrated the ability to learn the dataset distribution well enough to distinguish the given digits, and with moderate intra-class variance- indicates that the algorithm has the potential of finding more efficient and better performing circuits for image generation in a hybrid quantum GAN. However, further hyperparameter optimization and algorithmic improvements are needed.

While these results are promising, they should be contextualized in light of the limitations of the study. Intermittently available and limited computational resources did not allow for a full and throughout exploration and tuning of the hyperparameters. Because the system was entirely simulated, the same constraints posed harsh limits on the size of the architecture, i.e., number of qubits, that could be handled. Moreover, simulating the systems, while useful for the short term in exploratory studies as this one, it is fundamentally non scalable, and therefore not a sustainable approach in the long run.

There are several improvements to the algorithm, not all due and related to computational limits, that are worthy of exploration and could benefit overall performance. The fitness function calculation within the evolutionary strategy was one of the main limitations in the study methodology. It would be interesting to verify how the algorithm performs with a change in approach, by changing the fitness function itself, for example to the Fréchet Inception Distance (FID), and/or calculating the fitness on multiple patches or entire images, rather than single small patches, at least once every some generation of evolution, to limit computational overload. A further improvement could involve pairing the approach with the integration of gradient descent during evolution, to push the evolution towards faster convergence to optimal solutions. If these edits led to a more robust and reliable fitness function, this would allow for a further optimization for the entire pipeline, of launching GAN training only for the solutions that reach a certain fitness threshold during evolution, which was not possible to implement at the current stage of development, given the too weak correlation between evolutionary fitness and final results. More extensive hyperparameter tuning and a more in depth exploration of the effect of each hyperparameter on performance is required in order to adjust the algorithm to the specific application of image-generation ansatz search. Elements that would be interesting to explore include the number of data qubits, as well as hyperparameters such as d_θ, action weights, multi-action probability, and gate family. Finally, a different initialization set-up, including non-empty circuits, could lead to faster convergence and better discovered solutions.

As stated, the algorithm was entirely simulated without noise. Future work should investigate the effect of noise and performance on real devices. It would also be interesting to explore whether

different encoding methods can improve the algorithm's effectiveness, and test its performance with different applications and potentially more complex datasets to assess its scalability and robustness. Finally, an interesting future direction, which was the original idea that lead to this work, would be to investigate whether an adapted version of this model could be used to achieve the task of image super-resolution, given the necessary modifications.

In conclusion, this work demonstrated the potential of using a mutation-based evolutionary algorithm for optimizing ansätze in hybrid quantum-classical GANs, showing promise despite current limitations. Future research, supported by better hardware and more powerful resources, should address the algorithm's limitations as well as verify its performance on real devices. With further improvements and optimization, the proposed approach could lead to discovering more efficient and better performing ansätze for image generation, narrowing the existing gap in output quality between quantum and fully classical systems, while still exploiting the significant reduction in parameters and network complexity offered by the quantum systems.

8 Appendix

8.1 Evolutionary Algorithm: Mutation Procedure

Algorithm 2: Mutation Procedure of Quantum Ansatz Through Evolutionary Actions

Input: Parent ansatz, Number of children, Dictionary of possible gates, Action probabilities `[Add, Delete, Swap, Mutate]`, $d\theta$

Output: Best child ansatz after one generation

$P \leftarrow$ Empty population to store children of parent ansatz
for *k = 1 to n_children* **do**
    $P_k \leftarrow$ Copy of parent ansatz
    Choose `action` based on given probabilities
    if *action == Add* **then**
        Select a gate from the given gate dictionary
        Determine the number of required qubits for the gate
        Select destination qubits randomly from P_k and add gate to P_k
    else if *action == Delete* **then**
        Select a random gate from P_k
        Remove selected gate from P_k
    else if *action == Swap* **then**
        Select a random gate from P_k
        Randomly choose a new gate from the given gate dictionary
        if *n_qubits_new_gate* $==$ *n_qubits_original_gate* **then**
            Remove old gate from P_k
            Add new gate to P_k on the same qubits as removed gate
        else if *n_qubits_new_gate* $<$ *n_qubits_original_gate* **then**
            Select destination qubits as a subset of original qubits
            Remove old gate from P_k
            Add new gate to P_k on chosen destination qubits
        else if *n_qubits_new_gate* $>$ *n_qubits_original_gate* **then**
            Randomly choose missing qubits from remaining qubits in P_k
            Remove old gate from P_k
            Add new gate to P_k on chosen qubits
    else if *action == Mutate* **then**
        Select a parameterized gate in P_k
        Randomly choose one of its parameters to mutate
        Mutate the parameter by $\pm d\theta$
    Evaluate fitness of P_k
    Add P_k to P
best_ansatz $\leftarrow$ select ansatz with highest fitness from P
return *best_ansatz*

8.2 Additional Experiments Results

Table 12: Preliminary experiments on action weights

Weights	EMD	FID	U gates	CX gates	Params	Visual
7, 10, 46, 37	20.320	5.761	1	0	84	very poor
7, 10, 46, 37	20.320	20.054	1	0	84	very poor
7, 10, 46, 37	20.320	20.049	1	0	84	very poor
7, 10, 46, 37	20.320	20.065	1	0	84	very poor
18, 34, 9, 39	18.423	5.761	2	1	168	poor
18, 34, 9, 39	18.472	3.261	3	1	252	poor
18, 34, 9, 39	20.199	14.305	2	2	168	very poor
18, 34, 9, 39	18.431	4.449	5	2	420	poor
38, 21, 27, 14	18.306	5.496	4	2	336	medium
38, 21, 27, 14	18.697	8.858	2	5	168	medium
38, 21, 27, 14	19.098	6.675	2	5	168	poor
38, 21, 27, 14	18.110	3.095	3	4	252	poor
21, 14, 38, 27	17.884	4.626	3	2	252	poor
21, 14, 38, 27	18.555	3.428	3	0	252	poor
21, 14, 38, 27	17.945	5.544	4	2	336	poor
21, 14, 38, 27	18.707	4.039	4	1	336	poor

8.3 Hyperparameters

Table 13: *Stage 1.* Fixed hyperparameters and corresponding descriptions for the experiments shown in Table 2 and 3, in Section 5.

Parameter	Value	Description
`n_ancilla`	1	Number of ancilla qubits for the ansatz.
`n_children`	10	Number of copies of the parent ansatz to which one or more actions is performed per generation.
`d`θ	0.1	Maximum amount of change of the angle of a gate if Mutate action is selected for that gate.
`evaluation_patch`	random	Which patch to use for evaluating the generated images against the real ones, whether chosen randomly or specified explicitly (e.g., the top patch).
`action_weights`	60, 10, 10, 20	Probability to choose one of the 4 actions for each mutated child: *Add*, *Delete*, *Swap*, *Mutate*.
`multiaction_probs`	0.1	Probability to get multiple actions in the same generation.
`max_gen_until_change`	8	If no improvement is found for these many generations, the value of `d`θ is increased to facilitate escaping local minima.
`max_gen_no_improvement`	20	*End condition.* Terminate the algorithm if no improvement is found after these number of generations.
`max_depth`	10	Upper bound for the quantum circuits depth, i.e., the length of the critical path (longest sequence of gates).
`max_evaluations`	100	*End condition.* Tot number of generations is `max_evaluations` / `n_children`.
`batch_size`	25	Number of images in one batch, for both evolutionary and GAN.
`n_batches`	8	Number of batches of images preloaded into memory and used to draw a subset for the evaluation of the algorithm.
`batch_subset`	3	The number of images used to calculate the cost will be == `batch_subset`*`batch_size`.
`gan_n_epochs`	12	The number of epochs the GAN is trained for.
`gan_training_size`	1000	Size of the training set for the GAN network.
`gan_validation_size`	200	Size of the validation set for the GAN network, used to calculate FID score.

Table 14: *Stage 2.* Fixed hyperparameters and corresponding descriptions for the experiments shown in Table 4, in Section 5.

Parameter	Value	Description
`n_children`	15	Number of copies of the parent ansatz to which one or more actions is performed per generation
$d\theta$	0.1	Maximum amount of change of the angle of a gate if Mutate action is selected for that gate.
`evaluation_patch`	random	Which patch to use for evaluating the generated images against the real ones, whether chosen randomly or specified explicitly (e.g., the top patch).
`action_weights`	60, 10, 10, 20	Probability to choose one of the 4 actions for each mutated child: *Add, Delete, Swap, Mutate.*
`multiaction_probs`	0.1	Probability to get multiple actions in the same generation.
`max_gen_until_change`	20	If no improvement is found for these many generations, the value of $d\theta$ is increased to facilitate escaping local minima.
`max_gen_no_improvement`	150	*End condition.* Terminate the algorithm if no improvement is found after these number of generations.
`max_depth`	16	Upper bound for the quantum circuits depth, i.e., the length of the critical path (longest sequence of gates).
`max_evaluations`	15000	*End condition.* Total number of generations == `max_evaluations` / `n_children`.
`batch_size`	25	Number of images in one batch, for both evolutionary and GAN.
`n_batches`	8	Number of batches of images preloaded into memory and used to draw a subset for the evaluation of the algorithm
`batch_subset`	3	The number of images used to calculate the cost will be == `batch_subset*batch_size`.
`randn_latent`	FALSE	`False`: latent sampled from uniform distrubution. `True`: latent sampled from normal distribution.
`gan_n_epochs`	12	The number of epochs the GAN is trained for.
`n_layers`	1	The number of times the circuit found via evolution is repeated for (layered) in the generator architecture.
`gan_taining_size`	1000	Size of the training set for the GAN network.
`gan_validation_size`	200	Size of the validation set for the GAN network, used to calculate FID score.

Table 15: *Stage 3.* Fixed hyperparameters and corresponding descriptions for the experiments shown in Table 5, in Section 5.

Parameter	Value	Description
`n_data_qubits`	5	Number of data qubits of the circuit (for each patch).
`n_ancilla`	1	Number of ancillary qubits of the circuit (for each patch).
`image_side`	28	Number of pixels per image side (images are square).
`patch_shape`	(1, 28)	(rows, columns) for every patch, in pixels.
`pixels_per_patch`	28	Number of pixels per patch
`n_patches`	28	Number of total patches per generated image.
`n_children`	15	Number of copies of the parent ansatz to which one or more actions is performed per generation
$d\theta$	0.1	Maximum amount of change of the angle of a gate if Mutate action is selected for that gate.
`evaluation_patch`	random	Which patch to use for evaluating the generated images against the real ones, whether chosen randomly or specified explicitly (e.g., the top patch).
`multiaction_probs`	0.1	Probability to get multiple actions in the same generation.
`max_gen_until_change`	20	If no improvement is found for these many generations, the value of $d\theta$ is increased to facilitate escaping local minima.
`max_gen_no_improvement`	150	End condition. Terminate the algorithm if no improvement is found after these number of generations.
`max_depth`	16	Upper bound for the quantum circuits depth, i.e., the length of the critical path (longest sequence of gates).
`max_evaluations`	15000	End condition. Total number of generations == `max_evaluations / n_children`.
batch_size	25	Number of images in one batch, for both evolutionary and GAN.
`n_batches`	8	Number of batches of images preloaded into memory and used to draw a subset for the evaluation of the algorithm
`batch_subset`	3	The number of images used to calculate the cost will be == `batch_subset*batch_size`.
`randn_latent`	FALSE	False: latent sampled from uniform distribution. True: latent sampled from normal distribution.
`gan_n_epochs`	12	The number of epochs the GAN is trained for.
`n_layers`	1	The number of times the circuit found via evolution is repeated for (layered) in the generator architecture.
`gan_training_size`	1000	Size of the training set for the GAN network.
`gan_validation_size`	200	Size of the validation set for the GAN network, used to calculate FID score.

Table 16: *Stage 4*. Fixed hyperparameters and corresponding descriptions for the experiments shown in Table 6, in Section 5.

Parameter	Value	Description
`data qubits`	5	Number of data qubits of the circuit (for each patch).
`n_ancilla`	1	Number of ancillary qubits of the circuit (for each patch).
`image_side`	28	Number of pixels per image side (images are square).
`patch_shape`	1, 28	(rows, columns) for every patch, in pixels.
`pixels_per_patch`	28	Number of pixels per patch.
`n_patches`	28	Number of total patches per generated image.
`n_children`	15	Number of copies of the parent ansatz to which one or more actions is performed per generation.
`d_`θ	0.1	Maximum amount of change of the angle of a gate if Mutate action is selected for that gate.
`evaluation_patch`	random	Which patch to use for evaluating the generated images against the real ones, whether chosen randomly or specified explicitly (e.g., the top patch).
`Action Weights`	70, 5, 5, 20	Probability to choose one of the 4 actions for each mutated child: Add, Delete, Swap, Mutate.
`max_gen_until_change`	20	If no improvement is found for these many generations, the value of d is increased to facilitate escaping local minima.
`max_gen_no_improvement`	150	End condition. Terminate the algorithm if no improvement is found after these number of generations.
`max_depth`	16	Upper bound for the quantum circuits depth, i.e., the length of the critical path (longest sequence of gates).
`max_evaluations`	15000	End condition. Total number of generations == max_evaluations / n_children.
`batch_size`	25	Number of images in one batch, for both evolutionary and GAN.
`n_batches`	8	Number of batches of images preloaded into memory and used to draw a subset for the evaluation of the algorithm.
`batch_subset`	3	The number of images used to calculate the cost will be == batch_subset*batch_size.
`randn_latent`	FALSE	False: latent sampled from uniform distribution. True: latent sampled from normal distribution.
`gan_n_epochs`	12	The number of epochs the GAN is trained for.
`n_layers`	1	The number of times the circuit found via evolution is repeated for (layered) in the generator architecture.
`gan_training_size`	1000	Size of the training set for the GAN network.
`gan_validation_size`	200	Size of the validation set for the GAN network, used to calculate FID score.

Table 17: *Round 5*. Fixed hyperparameters and corresponding descriptions for the experiments shown in Table 7, in Section 5.

Parameter	Value	Description
`data qubits`	5	Number of data qubits of the circuit (for each patch).
`n_ancilla`	1	Number of ancillary qubits of the circuit (for each patch).
`image_side`	28	Number of pixels per image side (images are square).
`patch_shape`	1, 28	(rows, columns) for every patch, in pixels.
`pixels_per_patch`	28	Number of pixels per patch.
`n_patches`	28	Number of total patches per generated image.
`n_children`	15	Number of copies of the parent ansatz to which one or more actions is performed per generation.
`d_`θ	0.1	Maximum amount of change of the angle of a gate if Mutate action is selected for that gate.
`evaluation_patch`	random	Which patch to use for evaluating the generated images against the real ones, whether chosen randomly or specified explicitly (e.g., the top patch).
`Action Weights`	70, 5, 5, 20	Probability to choose one of the 4 actions for each mutated child: Add, Delete, Swap, Mutate.
`max_gen_until_change`	30	If no improvement is found for these many generations, the value of d is increased to facilitate escaping local minima.
`max_gen_no_improvement`	500	End condition. Terminate the algorithm if no improvement is found after these number of generations.
`max_depth`	16	Upper bound for the quantum circuits depth, i.e., the length of the critical path (longest sequence of gates).
`max_evaluations`	60000	End condition. Total number of generations == max_evaluations / n_children.
`batch_size`	25	Number of images in one batch, for both evolutionary and GAN.
`n_batches`	8	Number of batches of images preloaded into memory and used to draw a subset for the evaluation of the algorithm.
`batch_subset`	3	The number of images used to calculate the cost will be == batch_subset*batch_size.
`randn_latent`	FALSE	False: latent sampled from uniform distribution. True: latent sampled from normal distribution.
`gan_n_epochs`	40	The number of epochs the GAN is trained for.
`n_layers`	1	The number of times the circuit found via evolution is repeated for (layered) in the generator architecture.
`gan_training_size`	1000	Size of the training set for the GAN network.
`gan_validation_size`	200	Size of the validation set for the GAN network, used to calculate FID score.

Bibliography

[1] J. Gui, Z. Sun, Y. Wen, D. Tao, and J. Ye, "A review on generative adversarial networks: Algorithms, theory, and applications," *IEEE transactions on knowledge and data engineering*, vol. 35, no. 4, pp. 3313–3332, 2021.

[2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[3] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 53–65, 2018.

[4] A. Dash, J. Ye, and G. Wang, "A review of generative adversarial networks (gans) and its applications in a wide variety of disciplines: From medical to remote sensing," *IEEE Access*, 2023.

[5] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum, "Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling," *Advances in neural information processing systems*, vol. 29, 2016.

[6] C. Vondrick, H. Pirsiavash, and A. Torralba, "Generating videos with scene dynamics," *Advances in neural information processing systems*, vol. 29, 2016.

[7] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4681–4690, 2017.

[8] M. Benhenda, "Chemgan challenge for drug discovery: can ai reproduce natural chemical diversity?," *arXiv preprint arXiv:1708.08227*, 2017.

[9] N. Killoran, L. Lee, A. Delong, D. Duvenaud, and B. Frey, "Generating and designing dna with deep generative models," 12 2017.

[10] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, 1965.

[11] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2011.

[12] J. D. Hidary, *Quantum Computing: An Applied Approach*. New York: Springer, 2021.

[13] M. Schuld and F. Petruccione, *Machine learning with quantum computers*. Springer, 2021.

[14] D. Pastorello, *Concise guide to quantum machine learning*. Springer Nature, 2022.

[15] T. A. Ngo, T. Nguyen, and T. C. Thang, "A survey of recent advances in quantum generative adversarial networks," *Electronics*, vol. 12, no. 4, p. 856, 2023.

[16] P.-L. Dallaire-Demers and N. Killoran, "Quantum generative adversarial networks," *Physical Review A*, vol. 98, 04 2018.

[17] S. Lloyd and C. Weedbrook, "Quantum generative adversarial learning," *Physical review letters*, vol. 121, no. 4, p. 040502, 2018.

[18] C. Tian, X. Zhang, J. C.-W. Lin, W. Zuo, Y. Zhang, and C.-W. Lin, "Generative adversarial networks for image super-resolution: A survey," *arXiv preprint arXiv:2204.13620*, 2022.

[19] H.-L. Huang, Y. Du, M. Gong, Y. Zhao, Y. Wu, C. Wang, S. Li, F. Liang, J. Lin, Y. Xu, *et al.*, "Experimental quantum generative adversarial networks for image generation," *Physical Review Applied*, vol. 16, no. 2, p. 024051, 2021.

[20] S. L. Tsang, M. T. West, S. M. Erfani, and M. Usman, "Hybrid quantum-classical generative adversarial network for high resolution image generation," *arXiv preprint arXiv:2212.11614*, 2022.

[21] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," *Advances in neural information processing systems*, vol. 30, 2017.

[22] M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks," International Conference on Learning Representations (ICLR) 2017, April 2017.

[23] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*, pp. 214–223, PMLR, 2017.

[24] M. Roser, H. Ritchie, and E. Mathieu, "What is moore's law?," *Our World in Data*, 2023. https://ourworldindata.org/moores-law.

[25] J. Shalf, "The future of computing beyond moore's law," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2166, p. 20190061, 2020.

[26] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pp. 124–134, 1994.

[27] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pp. 212–219, 1996.

[28] M. Benedetti, E. Lloyd, S. Sack, and M. Fiorentini, "Parameterized quantum circuits as machine learning models," *Quantum Science and Technology*, vol. 4, no. 4, p. 043001, 2019.

[29] V. Lipardi, R. Schiattarella, and G. Acampora, "A quantum evolutionary strategy for optimization problems," in *Workshop on Quantum Artificial Intelligence 2023, Program and Short Papers*, Università degli Studi di Napoli Federico II, 2023.

[30] S. N. Ghoreishi, A. Clausen, and B. N. Jørgensen, "Termination criteria in evolutionary algorithms: A survey.," in *IJCCI*, pp. 373–384, 2017.

[31] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[32] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[33] Qiskit contributors, "Qiskit: An open-source framework for quantum computing," 2023.

[34] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, S. Ahmed, V. Ajith, M. S. Alam, G. Alonso-Linaje, B. AkashNarayanan, A. Asadi, *et al.*, "Pennylane: Automatic differentiation of hybrid quantum-classical computations," *arXiv preprint arXiv:1811.04968*, 2018.

[35] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[36] A. Borji, "Pros and cons of gan evaluation measures," *Computer vision and image understanding*, vol. 179, pp. 41–65, 2019.

[37] L. V. Kantorovich, "Mathematical methods of organizing and planning production," *Management science*, vol. 6, no. 4, pp. 366–422, 1960.

[38] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," *Advances in neural information processing systems*, vol. 30, 2017.

[39] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[40] M. Seitzer, "pytorch-fid: FID Score for PyTorch." `https://github.com/mseitzer/pytorch-fid`, August 2020. Version 0.3.0.