eman ta zabal zazu

Universidad del País Vasco

Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA

FACULTAD DE INFORMÁTICA

Bachelor Thesis

Degree in Informatics Engineering

Computing

Development and Evaluation of Fully Quantum Generative Adversarial Networks on IBM Quantum Hardware

Beñat Burutaran Maiz

Advisors

Jose A. Pascual Saiz

June 23, 2025

Acknowledgments

I would like to sincerely thank Ikerbasque and the University of the Basque Country (EHU) for providing access to IBM's real quantum hardware, which was essential for the experimental development of this thesis. I am also grateful to my friends Garikoitz Artola and Amaia Ona Villaverde for their help in presenting an early version of this project during the Road to Practitioner (R2P) Program, organized by BasQ in collaboration with IBM.

I would also like to thank my professor, Jose, for allowing me to pursue my thesis on quantum machine learning and for supporting my interest in this field. His openness and guidance played an important role in making this project possible.

Abstract

This thesis presents the design, implementation, and evaluation of a fully Quantum Generative Adversarial Network (QGAN) architecture executed on IBM Quantum hardware. In contrast to hybrid approaches, this model employs quantum circuits for both the generator and discriminator, while classical optimization techniques guide the training process. The QGAN is tested across three environments: ideal noiseless simulators, noisy simulators emulating quantum hardware, and actual IBM quantum devices. The implementation incorporates custom quantum circuit designs and supports multiple data encoding strategies, specifically amplitude and angle encoding. Model performance is assessed using convergence behavior and output quality, measured through Kullback–Leibler divergence and structural similarity metrics, depending on the encoding scheme. Results demonstrate the practical feasibility of running QGANs on Noisy Intermediate-Scale Quantum (NISQ) devices, revealing trade-offs among expressivity, noise resilience, and circuit complexity. The study also identifies critical challenges in scalability and optimization, offering design strategies to address them. Overall, this work advances the development of quantum-native generative models suitable for near-term hardware, contributing to the broader pursuit of quantum-enhanced machine learning and responsible innovation in quantum computing.

Sustainable Development Goals

Agenda 2030, established by the United Nations, provides a global framework for sustainable development through 17 interconnected Sustainable Development Goals (SDGs). While primarily focused on social, economic, and environmental dimensions, it also recognizes the critical role of innovation, science, and technology in achieving these goals. In this context, research on Quantum Generative Adversarial Networks (QGANs) aligns with Agenda 2030 by contributing to SDG 9: "Industry, Innovation and Infrastructure." QGANs represent a frontier in quantum machine learning with potential applications in quantum chemistry, high-energy physics, and secure data generation — fields that directly support sustainable industrial innovation.

Moreover, QGANs could accelerate discoveries in drug development and materials science, supporting SDG 3 (Good Health and Well-being) and SDG 7 (Affordable and Clean Energy) by enabling faster, more accurate simulations at the quantum level. By addressing computational challenges with quantum-enhanced methods, this work contributes to developing infrastructure and knowledge systems that foster sustainable development. In short, advancing QGANs is part of a long-term vision for leveraging quantum technologies in support of a more sustainable future.

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

Quantum Machine Learning (QML) is a research field that combines ideas from quantum computing and machine learning to explore how quantum algorithms can help solve learning tasks more efficiently [5]. Classical machine learning has achieved great success in areas like image recognition, natural language processing and data analysis, but it often requires large amounts of computational resources. Quantum computing, on the other hand, uses the principles of quantum mechanics to perform certain calculations much faster than classical computers. QML aims to take advantage of these quantum properties to speed up learning algorithms or to create new types of models that cannot be easily simulated on classical machines. While still in its early stages, QML shows promise for improving tasks like classification, pattern recognition and generative modeling, especially as quantum hardware continues to develop.

Generative Adversarial Networks (GANs) are a class of machine learning models designed to generate synthetic data that closely resembles real data [6]. GAN consists of two neural networks: a generator and a discriminator. The generator is responsible for creating synthetic data samples, while the discriminator evaluates their authenticity by distinguishing between real and generated data. Through an adversarial training process, both networks continuously improve, leading to the generation of increasingly realistic data. GANs have found applications in various domains, including image synthesis, data augmentation and financial modeling.

Quantum computing introduces the potential for significant advancements in the field of GANs. Quantum Generative Adversarial Networks (QGANs) leverage the principles of quantum mechanics, such as superposition and entanglement, to enhance generative modeling [7]. The ability of quantum computers to represent and process complex probability distributions more efficiently than classical systems suggests that QGANs could offer advantages in terms of computational efficiency and expressivity. This opens up possibilities for generating high-dimensional data in fields where classical models struggle, such as quantum chemistry and material sciences.

QGANs have evolved to include both hybrid and fully quantum designs, demonstrating high parameter efficiency and stable performance metrics while supporting applications from image synthesis to molecular generation. Hybrid QGANs, which combine quantum generators with classical discriminators, are the most common architecture, offering the best balance between computational efficiency and trainability. Fully quantum models are less common but provide unique insights into the power of quantum state learning. Notable advancements include parameter-efficient architectures and entanglement-based models that leverage quantum state correlations to improve convergence and mitigate error.

Despite their promise, QGANs remain an emerging and largely experimental field. Research in this area faces several challenges, including hardware constraints, optimization difficulties and theoretical uncertainties. Current quantum processors, known as Noisy Intermediate-Scale Quantum (NISQ) devices, have limited qubit capacity and are susceptible to errors, which restricts the scalability of QGANs. Additionally, quantum circuit optimization is an ongoing area of research, as deep quantum circuits suffer from problems like barren plateaus, which make training unstable. Furthermore, QGANs, like their classical counterparts, suffer from difficulties such as mode collapse, where the generator fails to produce diverse samples, reducing the model's effectiveness.

The main goal of this thesis is to develop an updated QGAN that uses both a quantum generator and a quantum discriminator, with classical parameter optimization methods. The number of qubits in the circuits will change depending on the size of the training data. This data will consist of probability distributions, and the design will focus on making the model executable on IBM's quantum hardware.

The implemented model will be evaluated across multiple environments, including ideal simulators, noisy simulators, and real quantum hardware. To ensure practical relevance, the architecture is specifically tailored to current NISQ devices. Additionally, the project will explore different methods for encoding classical data (i.e., images) into quantum states.

CHAPTER 2

The aims of the project

This thesis focuses on the implementation and evaluation of a modern QGAN. The main objective is to build a QGAN model in which both the generator and the discriminator are quantum circuits. These circuits will be trained using classical optimization techniques, which is the most practical approach given the limitations of current quantum devices.The implementation targets IBM Quantum hardware, taking into account the restrictions of the current Noisy Intermediate-Scale Quantum (NISQ) era.

To ensure the circuits can run efficiently on real hardware, different quantum circuit designs and transpilation methods will be explored. These techniques help optimize the quantum circuits for specific backends by reducing gate count and circuit depth, which is essential for improving fidelity and minimizing noise.

The QGAN will be trained on probability distributions, which are commonly used benchmarks in quantum machine learning. The number of qubits in the circuits will be adjusted depending on the complexity and size of the input data, allowing for flexibility and adaptation to different tasks. In addition, this project will implement multiple methods for encoding classical data into quantum circuits. Two such methods to be explored are amplitude encoding and angle encoding.

Another important goal is to compare the performance of the fully quantum QGAN in a noiseless simulator, which idealizes quantum behavior, to a noisy or real quantum hardware, which reflects the imperfections and decoherence present in current NISQ era devices. The model will be developed with the intention of being executed on real IBM quantum devices, with particular attention to limiting circuit depth, managing noise and ensuring efficient runtime.

By addressing these goals, this work aims to offer practical insights into how QGANs can be designed and executed in current state-of-the-art hardware. It also seeks to support further development in the field of quantum machine learning, particularly in generative modeling.

The tasks to follow can be summarized as:

- Research about QGAN state of the art, designs, and implementations.
- Design and implement a QGAN model executable on real IBM Quantum devices.
- Implement encoding methods for input data.
- Explore different QGAN architectures, configurations, and data sizes.
- Evaluate the performance in terms of accuracy, efficiency, and compatibility with current quantum devices.
- Discuss the results of the model and the different approaches.

CHAPTER 3

Generative Adversarial Networks (GANs)

Generative Adversarial Networks, or GANs, are a type of machine learning model introduced by Ian Goodfellow in 2014 [6]. The main idea behind GANs is to train two models at the same time: a generator and a discriminator. The generator creates new data that looks similar to real data, while the discriminator tries to tell the difference between real data and the data produced by the generator.

During training, the generator improves by learning how to produce data that the discriminator cannot easily distinguish from real data. At the same time, the discriminator improves by learning to detect whether the input data is real or fake. This creates a game between the two models, where each tries to outperform the other. When training is successful, the generator becomes good at producing data that looks very realistic.

3.1 Components

A classical GAN is composed of the following two main components:

The generator is a neural network denoted by $G(z;\theta_G)$, where z is a noise vector sampled from a simple prior distribution (e.g., Gaussian or uniform), and θ_G represents the generator's parameters. The role of the generator is to transform this noise vector into a data sample that mimics the real data distribution as closely as possible.

The discriminator is another neural network denoted by $D(x;\theta_D)$, where x is a data sample and θ_D represents the discriminator's parameters. The discriminator receives either a real data sample from the dataset or a fake sample from the generator. It outputs a probability indicating whether the input is real (i.e., from the training set) or fake (i.e., generated).

3.2 Architecture

The GAN framework follows an adversarial training scheme, where the generator and discriminator are optimized with conflicting objectives. This setup can be described as a

```
1  input: Number of training iterations $N$, batch size $m$
2  input: Learning rates $\alpha_D$ for discriminator and $\alpha_G$ for generator
3  input: Generator $G(z;\theta_G)$, Discriminator $D(x;\theta_D)$
4  for number of training steps
5      for discriminator training steps
6          Sample a minibatch of $m$ real data samples $\{x^{(1)},\dots,x^{(m)}\} \sim p_{\text{data}}(x)$
7          Sample a minibatch of $m$ noise samples $\{z^{(1)},\dots,z^{(m)}\} \sim p_z(z)$
8          Generate fake data samples $\tilde{x}^{(i)} = G(z^{(i)})$ for $i = 1,\dots,m$
9          Compute discriminator loss:
               $$L_D = -\frac{1}{m}\sum_{i=1}^{m}\left[\log D(x^{(i)}) + \log(1 - D(\tilde{x}^{(i)}))\right]$$
10         Update discriminator parameters:
               $$\theta_D \leftarrow \theta_D - \alpha_D \nabla_{\theta_D} L_D$$
11     rof
12     for generator training steps
13         Sample a minibatch of $m$ noise samples $\{z^{(1)},\dots,z^{(m)}\} \sim p_z(z)$
14         Generate fake data samples $\tilde{x}^{(i)} = G(z^{(i)})$ for $i = 1,\dots,m$
15         Compute generator loss:
               $$L_G = -\frac{1}{m}\sum_{i=1}^{m}\log D(\tilde{x}^{(i)})$$
16         Update generator parameters:
               $$\theta_G \leftarrow \theta_G - \alpha_G \nabla_{\theta_G} L_G$$
17     rof
18 rof
```

Algorithm 3.1: Training algorithm for Generative Adversarial Networks.

two-player minimax game.

$$\min_{\theta_G}\max_{\theta_D} \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x;\theta_D)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(\tilde{x};\theta_D))]$$

Here, $p_{\text{data}}(x)$ is the distribution of real data and $p_z(z)$ is the prior distribution of the noise samples z. $D(x;\theta_D)$ is the probability that the discriminator classifies input x as real, and $\tilde{x} = G(z;\theta_G)$ is the data generated from noise input z. The generator tries to minimize the loss so that $D(\tilde{x};\theta_D)$ becomes close to 1, meaning the discriminator believes the generated data is real. The procedure is presented in Algorithm 3.1.

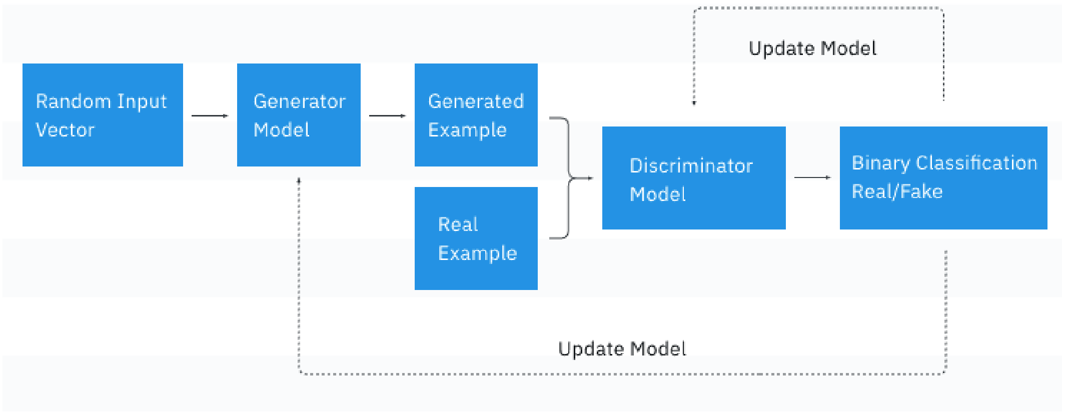


Figure 3.1: Generative Adversarial Network training process graph from [1].

During training, the two networks are updated in alternating steps. Firstly, the discriminator is trained to improve its ability to classify real versus generated data. Secondly, the generator is trained to improve its ability to fool the discriminator.

Each of these steps is driven by the minimization of a loss function, one for the generator and one for the discriminator. The discriminator loss function is defined as

$$L_D = -\left[\log D(x) + \log(1 - D(\tilde{x}))\right] \tag{3.1}$$

where $D(x)$ is the probability of labeling x real data correctly. The discriminator is trained to maximize this loss, which helps it better distinguish between real and fake data. The generator loss function is defined as

$$L_G = -\log D(\tilde{x}) \tag{3.2}$$

where $D(\tilde{x})$ is the probability that the discriminator labels $\tilde{x}$ fake data as real. Minimizing this loss means the generator is improving its ability to produce convincing samples.

This dynamic creates a feedback loop, as shown in Figure 3.1. As the discriminator improves at identifying fake samples, the generator must also improve to maintain training progress.

The theoretical equilibrium of this game is reached when the generated data distribution matches the real data distribution. At this point, the discriminator outputs 0.5 for all inputs, indicating total uncertainty, and no longer provides useful feedback to the generator.

3.3 Gradient Computation

Parameters θ_G and θ_D are optimized using gradient-based optimization techniques. Gradients are calculated using backpropagation, a standard technique in neural networks that applies the chain rule to compute the derivative of the loss with respect to each model parameter. For a neural network with layers, the output of one layer becomes the input to the next.

Suppose the loss function is $\mathcal{L}$, a parameter in the n-th layer is θ and z_k denotes the output of the k-th layer for $k = 1, \ldots, n$. The derivative of the loss with respect to θ is

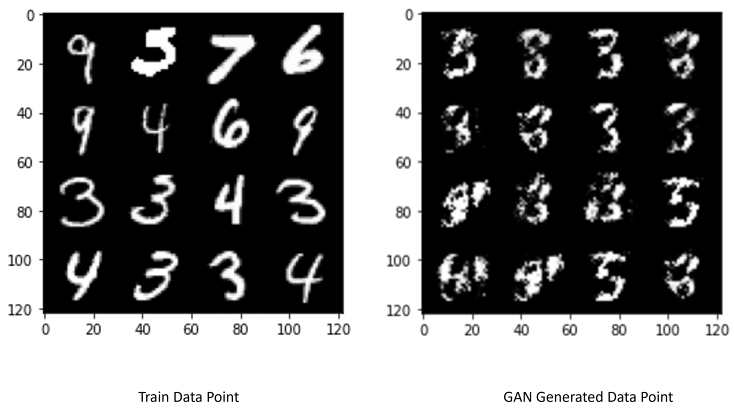


Figure 3.2: GAN mode collapse effect.

computed using the chain rule as

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial z_n} \cdot \frac{\partial z_n}{\partial z_{n-1}} \cdot \frac{\partial z_{n-1}}{\partial z_{n-2}} \cdots \frac{\partial z_1}{\partial \theta}$$

These gradients are then used in optimization algorithms such as stochastic gradient descent (SGD) or Adam to update the parameters of both the generator and the discriminator. In practice, the optimization is often done by alternating between one or more steps of updating D, and then updating G, since the two objectives are adversarial and coupled.

3.4 Challenges in GAN Training

In practice, training a GAN presents several difficulties [8]. Unlike traditional supervised learning models, GANs involve a dynamic interaction between two networks with opposing objectives, which often leads to unstable behavior. These are the main challenges in GAN training:

- **Mode Collapse:** One of the most common problems in GANs is *mode collapse*, where the generator learns to produce only a limited variety of outputs (Figure 3.2). Instead of capturing the full diversity of the real data distribution, the generator may repeatedly generate the same or similar samples, even for different inputs z. This occurs because the generator finds a local optimum that consistently fools the discriminator, but does not generalize well across the data space.
- **Vanishing Gradients:** Another major issue is *vanishing gradients*, which arises when the discriminator becomes too strong. If the discriminator can easily distinguish fake samples from real ones, the loss for the generator becomes very small, and the gradient signal may vanish. As a result, the generator stops learning, and training fails to progress.
- **Training Instability:** GAN training is highly sensitive to hyperparameters, network architectures and optimizer settings. The adversarial nature of the setup can lead to oscillatory behavior, where the generator and discriminator continuously outsmart each other without converging. Sometimes, the model may diverge entirely, producing meaningless outputs.

To overcome these challenges, numerous improvements and variants of GANs have been proposed to tackle these problems [8]. These enhancements generally aim to stabilize training, ensure better convergence and promote diversity in the generated outputs. By modifying the loss functions, optimization strategies and network architectures, researchers have been able to significantly improve the performance and reliability of GAN models in a wide range of applications [9].

CHAPTER 4

Quantum Generative Adversarial Networks (QGANs)

Quantum GANs are inspired by classical GANs but use quantum circuits instead of classical neural networks. In a typical QGAN, either the generator, the discriminator, or both are implemented as quantum circuits. These circuits take advantage of quantum phenomena such as superposition and entanglement to represent data distributions in ways that may be more efficient than classical models. In QGANs, some parts of the training are still done using classical optimization techniques, while quantum devices are used to generate or classify quantum states.

The main differences lie in how data is represented and processed. Classical GANs operate with real-valued vectors and matrices, while QGANs use quantum states and unitary operations. Gradients in classical GANs are computed using backpropagation, whereas in QGANs, gradient calculation relies on more efficient but limited methods. Therefore, QGANs require careful circuit design, encoding strategies and efficient transpilation.

4.1 Quantum Hardware Limitations

Unlike classical computers, quantum computers face a unique set of limitations that impact their reliability, scalability and usability. These limitations arise from both physical hardware constraints and the nature of quantum operations. For quantum machine learning models such as QGANs, which rely on repeated circuit executions, gradient evaluations and precise parameter updates, these limitations pose significant challenges. Understanding the impact of quantum hardware noise, decoherence and measurement constraints is crucial for constructing a model based on the feasibility and performance of current quantum devices.

4.1.1 Fundamental Limitations

Quantum computing is fundamentally limited by the nature of quantum mechanics. These limitations affect how quantum circuits are designed, executed and interpreted. The most important of these are:

- **Measure Collapse:** Quantum systems exist in superpositions of multiple states. However, once a measurement is made, the system collapses to a single classical outcome, destroying the superposition. This irreversible process limits the amount of information that can be extracted from a quantum system.

- **Limited Observability:** Quantum states cannot be directly accessed or fully observed. Instead, the outcome of a quantum computation must be inferred by repeatedly running the same circuit and measuring its output. This makes the evaluation of quantum algorithms probabilistic and resource-intensive.

- **Gate Reversibility:** All quantum gates must be unitary, meaning they must preserve the total probability and be mathematically reversible. This requirement prevents the direct implementation of common classical logic operations such as AND or OR, which are not reversible. As a result, quantum algorithms must be carefully constructed using reversible logic, which adds to the complexity of circuit design.

- **No-Cloning Theorem:** A central property of quantum mechanics is that arbitrary quantum states cannot be copied. In practice, this means that quantum information cannot be duplicated for backup, error correction, or debugging as is common in classical computing. As a result, recovering a quantum state once it has been disturbed or lost is extremely challenging.

These fundamental principles shape both the strengths and constraints of quantum computation. While they enable powerful behaviors like superposition and entanglement, they also require new ways of thinking about computation, measurement and operations, especially in comparison to classical systems.

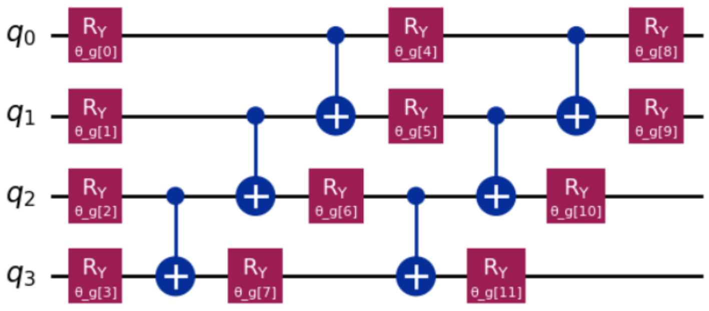
4.1.2 Practical Limitations

Beyond the fundamental constraints imposed by quantum mechanics, real-world quantum processors face a range of practical limitations that affect their performance and reliability. These limitations arise from the physical implementation of quantum systems, particularly those based on superconducting qubits. The main constrains are:

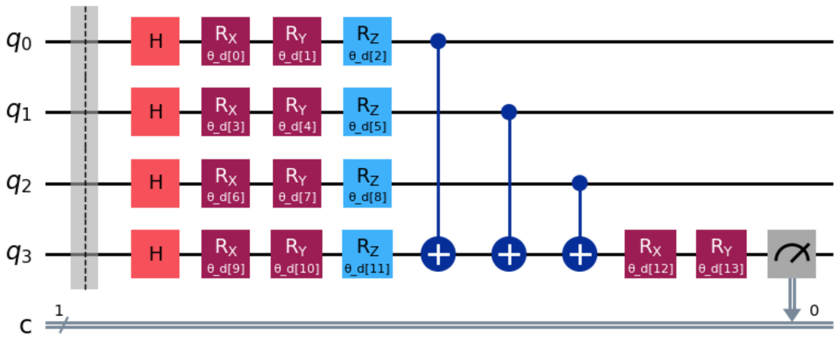
- **Gate Errors:** Quantum gates implemented on real hardware are not perfect. Each gate (such as single-qubit rotations or two-qubit CNOTs) introduces a small error, which accumulates as circuits become deeper. Two-qubit gates are particularly prone to higher error rates compared to single-qubit gates.

- **Readout Errors:** Measurement of qubit states is another source of noise. Even if a qubit is in a well-defined state before measurement, hardware imperfections may cause incorrect readout, leading to uncertainty in the recorded result.

- **Decoherence:** Qubits lose their quantum information over time due to interaction with the environment. This phenomenon, known as decoherence, is characterized by the T_1 (relaxation) and T_2 (dephasing) times, and limits the duration over which a quantum circuit can reliably run.



(a) Generator implementation using `RealAmplitudes` from Qiskit.



(b) Discriminator implementation using multiple rotation gates and one qubit measurement.

Figure 4.1: 4 qubit Variational Quantum Circuits for generator and discriminator.

- **Qubit Connectivity and Crosstalk:** In most quantum devices, not all qubits are directly connected. Executing operations between distant qubits may require additional SWAP gates, which further increase circuit depth and error. Moreover, operations on one qubit can unintentionally affect neighboring qubits due to crosstalk.

These factors collectively limit the depth and complexity of quantum circuits that can be reliably executed on current hardware. As a result, quantum circuits must be carefully designed to be hardware-efficient and resilient to noise, especially when targeting real quantum devices.

4.2 Components

An essential component in the architecture of a QGAN is the Variational Quantum Circuit (VQC), used as the generator or the discriminator (Figure 4.1). A VQC is a quantum circuit where certain gates have parameters that can be adjusted. These parameters are not fixed but are instead treated as variables that can be optimized to achieve a desired outcome.

An *ansatz* is a VQC that serves as a trial solution for the problem being addressed. These circuits define how the quantum state is transformed in order to approximate a target probability distribution [10]. When designing an ansatz for QGAN generators, one must strike a balance between:

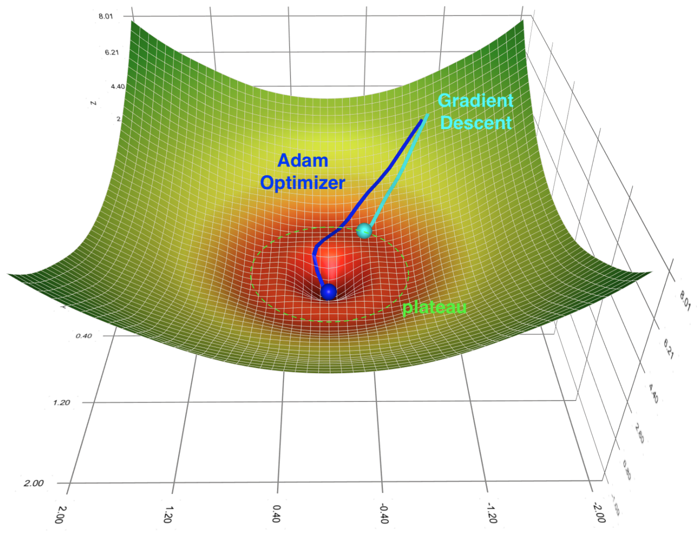


Figure 4.2: Barren plateau illustration from [2]. Simple gradient descent does not avoid barren plateaus. Therefore, optimizers such as Adam, that use techniques like adaptive learning rates or momentum, can contribute to escape such regions [3].

- **Expressivity:** The ability to represent complex distributions.
- **Trainability:** The presence of informative gradients during optimization.
- **Noise resilience:** Robustness under realistic, imperfect hardware conditions.

The use of multiple rotation gates and qubit connections may improve expressivity. However, they often require deeper and more complex circuits, which may not be practical for NISQ era devices. Additionally, increasing the depth of the circuit can also make the circuit more susceptible to hardware noise and barren plateaus.

The *barren plateau* problem consists of regions in the parameter space where gradients vanish, such as the one shown in Figure 4.2. These regions increase exponentially with the number of qubits or circuit depth [11]. This makes training extremely difficult, particularly on noisy hardware. In the context of QGANs, this issue is significant because effective training requires navigating a potentially non-convex landscape to minimize the divergence between real and generated distributions.

4.2.1 Ansatz Designs

To enable efficient training, ansatz are typically constructed using single-qubit rotation gates like RX, RY and RZ, and entangling gates such as CNOT or CZ. In practice, several types of ansatz are commonly employed in quantum machine learning and QGANs:

- **Hardware-Efficient Ansatz:** This type of ansatz is designed to be more practical and minimize the impact of hardware noise. Is typically composed of layers of parameterized single-qubit rotations and entangling gates that align with the hardware's native gate set. Therefore, it should consider the limitations of current quantum hardware, such as gate fidelity, qubit connectivity and gradient calculation [12]. A higher number of layers (repetitions) improves expressivity, which is the ability to represent complex states. On the contrary, a lower number of layer reduces gate count and depth, improving efficiency. For instance, Figure 4.1a is an example of this type of ansatz.

- **Problem-Inspired Ansatz:** A problem-inspired ansatz is tailored to the structure of a specific problem. One example is the Unitary Coupled Cluster (UCC) ansatz used in quantum chemistry [13]. These ansatzes can offer improved performance by embedding domain knowledge into the circuit. However, they often require deeper and more complex circuits, which may not be practical for NISQ era devices.

- **Adaptive Ansatz:** The adaptive ansatz is constructed iteratively by adding gates and entanglers based on some optimization criteria, such as the gradient magnitude of the loss function [14, 15]. This approach offers potential for efficient and expressive circuit design by focusing only on the most relevant gates. Nonetheless, adaptive methods introduce additional classical computational overhead and complexity in circuit optimization.

Looking ahead, as quantum hardware improves in coherence time, gate fidelity and qubit connectivity, the constraints of the NISQ era will begin to loosen. This will open the door to more expressive and deeper ansatz, including those that do not rely exclusively on Pauli-based generators.

4.3 Architectures

Since their introduction in 2018, various architectures and training methodologies have been proposed, each addressing specific challenges and exploiting unique advantages of quantum systems [16]. The two main designs are, on one hand, a hybrid architecture where a quantum generator is paired with a classical discriminator, and on the other hand, a fully quantum architecture where both the generator and discriminator are quantum models. Each design has distinct advantages, computational trade-offs and implementation strategies.

4.3.1 Hybrid QGAN

This is the most widely studied and practically efficient configuration [17, 18, 19, 20]. The generator is a variational quantum circuit, while the discriminator is a classical neural network. This configuration often follows classical GAN's architecture shown in Figure 3.1.

The quantum generator prepares a quantum state whose measurements return samples interpreted as fake data. These are passed to the classical discriminator, which assigns probabilities indicating whether samples are real or generated. A common method to connect the generator circuit to a classical neural network is `TorchConnector` from Qiskit Machine Learning library [21], which connects it to PyTorch [22]. This enables hybrid backpropagation and optimization using gradient descent.

Typically, a subset or all of the qubits are measured to produce bitstrings representing samples. The classical discriminator processes these bitstrings, which allows for efficient training loops with low overhead. This architecture has shown high performance in terms of convergence speed and scalability with the number of training parameters [19].

This architecture is generally more efficient in terms of the number of training loops and quantum circuit evaluations required per iteration, which is critical for practical implementations on NISQ devices [23, 24]. Moreover, it allows for an easy integration of the quantum generator with classical machine learning libraries.

4.3.2 Fully Quantum GAN

Other approaches implement both the generator and discriminator as variational quantum circuits [25, 26, 27, 28]. They are also known as fully quantum approaches, since data samples, neural networks and gradient calculation are quantum, but they still rely on classical optimization to update parameters.

The generator and discriminator are implemented as separate quantum circuits. The generator prepares a quantum state intended to mimic the distribution of real quantum data. The discriminator receives either the real or generated state and outputs a classification result by measuring the circuit (Figure 4.1b).

Usually, circuits are combined so only one qubit is measured to produce a classification result. While this facilitates gradient calculation, the fully quantum loop requires more and deeper quantum circuit evaluations per training step (example of implementation in Figure 5.3), making it less efficient overall compared to hybrid approaches. In addition, scaling these models is difficult due to the increased circuit noise and vanishing gradients, also making gradient estimation tools slower and less precise.

Nevertheless, the main advantage of a fully quantum QGAN is that it maintains a completely quantum training loop, which opens the possibility for achieving quantum advantage, especially in cases where the real data is also quantum in nature. Therefore, the discriminator circuit can directly apply quantum features such as entanglement and quantum correlations, potentially allowing it to capture patterns and distinctions that would be difficult for classical discriminators to recognize.

4.3.3 Variants

QGANs have been developed in many different forms. Some introduce new ideas, such as using entanglement to improve learning [29], or focusing on stable training with Wasserstein distance [30]. There are also style-based QGANs for generating complex data like particle physics events [31], and semi-supervised QGANs that combine classification and generation [32]. Some models use energy loss (dissipation) to simulate real physical systems [33, 34], and others mix quantum and classical parts for better performance [35]. Hamiltonian QGANs [36] use control theory ideas to improve how quantum states are generated.

Several of these models have been tested on real quantum hardware. QGANs have been run on superconducting processors [37, 38], trapped-ion devices [39], and even analog Rydberg atom computers [40]. Photonic chips have also been used to create QGANs with light-based circuits [41, 42]. Some experiments tested QGANs on different platforms to compare results, like using both IBM and IonQ devices [43]. These tests show that QGANs are flexible and can work on many kinds of quantum computers.

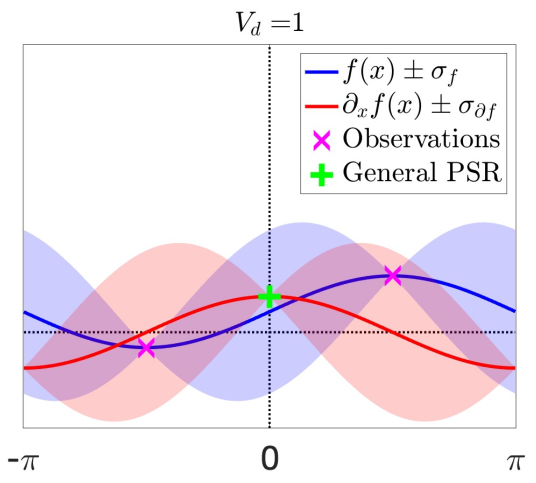


Figure 4.3: Illustration from [4] of the parameter-shift rule (PSR) for estimating the derivative of a quantum expectation value $f(x)$. The blue curve shows the function $f(x)$, and the red curve its derivative $\partial_x f(x)$. Shaded regions represent uncertainty bands. Magenta crosses indicate function evaluations at shifted parameters, used to compute the derivative via PSR. The green plus shows the resulting gradient estimate at $x = 0$, obtained without requiring analytic gradients or circuit modifications.

4.4 Quantum Gradient Computation

Training VQCs requires evaluating how changes in circuit parameters affect the output, so it hinges on the calculation of gradients. In classical machine learning, gradients are typically computed using backpropagation. However, quantum circuits, due to their unitary nature and measurement-based outputs, do not allow for backpropagation in the traditional sense. Instead, quantum gradients must be estimated through alternative techniques compatible with quantum computation.

Due to the NISQ era limitations, a widely used and efficient method is the *parameter-shift rule* [44]. It enables exact gradient computation under specific conditions:

Consider a quantum gate of the form $U(\theta) = e^{-i\theta G_{gate}}$, where G_{gate} is a Hermitian operator known as the generator[1] of the gate, and θ is a tunable parameter. When this gate acts on an initial state $|\psi_0\rangle$, it produces a new state $|\psi(\theta)\rangle = U(\theta)|\psi_0\rangle$. The expectation value of an observable $\hat{O}$ is given by:

$$f(\theta) = \langle\psi(\theta)|\hat{O}|\psi(\theta)\rangle \tag{4.1}$$

If the generator G has only two distinct eigenvalues, such as ± 1, the function $f(\theta)$ becomes sinusoidal with respect to θ (Figure 4.3). This sinusoidal form allows us to compute the derivative of $f(\theta)$ exactly using the parameter-shift rule:

$$\frac{\partial f(\theta)}{\partial \theta} = \frac{1}{2}\left[f\left(\theta + \frac{\pi}{2}\right) - f\left(\theta - \frac{\pi}{2}\right)\right] \tag{4.2}$$

[1]Avoid confusion with GAN's generator component G.

This formula involves running the quantum circuit twice, once with the parameter shifted forward and once shifted backward, and then combining the results. It enables gradient estimation that is both exact (for compatible gates) and hardware-friendly, requiring no internal modification of the quantum circuit. The practical implication of this rule is significant: for a quantum circuit with n trainable parameters, the full gradient can be obtained with $2n$ evaluations per training iteration. While this is more computationally expensive than backpropagation in classical neural networks, it provides an exact and stable approach for QGANs.

To apply this technique effectively, QGAN generators are typically implemented using hardware-efficient ansatz, composed of alternating layers of single-qubit Pauli-based rotation gates and fixed entangling gates. This approach provides gradient compatibility by supporting the parameter-shift rule, while maintaining noise resilience and execution efficiency.

In scenarios where a gate's generator has more than two eigenvalues or the parameter dependence is more complex, the parameter-shift rule no longer applies directly. In such cases, alternative methods like algebraic extensions of the parameter-shift rule [45] or finite differences [46] may be used, although they typically increase computational cost and noise sensitivity.

In future research, alongside expressive and deeper ansatz, alternative gradient estimation strategies may become more practical. These developments could reduce the impact of barren plateaus and enable richer function classes to be represented by QGAN generators or discriminators.

4.5 Applications and Future Directions

QGANs have gained attention for their potential to solve complex problems across a wide range of scientific and industrial domains. By leveraging quantum features, these models aim to enhance data generation, simulation and analysis, offering new possibilities beyond the capabilities of classical generative models.

In physics, QGANs have been employed to simulate high-energy processes and particle interactions. Notable applications include modeling particle detectors [47], generating gluon jets [48] and detecting anomalies in experimental data [49]. These tasks are computationally expensive in classical setups, and QGANs offer a path toward more efficient simulations.

In quantum chemistry and pharmaceutical research, QGANs have been applied to molecular structure generation and property prediction [50, 51]. These models facilitate the exploration of chemical space for new drug candidates and functional materials, benefiting from quantum encoding of structural information.

Beyond science, QGANs have demonstrated value in applied domains such as cybersecurity, where they are used to detect anomalous user behaviors and system intrusions [52, 53]. In finance, they are explored for generating synthetic financial data and improving decision models [54, 55, 56].

One of the most actively researched applications is image generation [16]. QGANs have been experimentally validated for generating and refining grayscale images [57], and more recent models have enhanced robustness on NISQ devices [58, 59]. Hybrid models have also been applied to hyperspectral image restoration [60], showing how QGANs can effectively operate on high-dimensional visual data with limited quantum resources.

Recent developments in QGAN research have focused on domain-specific applications and more scalable architectures [7]. These efforts aim to leverage the unique strengths of quantum models in representing complex data distributions. There is also growing interest in designing noise-resilient architectures that are compatible with current Noisy Intermediate-Scale Quantum hardware.

Future research is likely to concentrate on several key directions aimed at enhancing the scalability, reliability, and practicality of QGANs [8]. A primary focus will be on developing robust error mitigation techniques that can compensate for quantum noise without requiring full error correction, which remains out of reach for current devices. At the same time, designing more hardware-efficient circuit architectures will be essential for better compatibility with NISQ hardware. Additionally, hybrid quantum-classical models may help balance computational demands and improve performance under hardware constraints. Alongside these practical efforts, there is growing interest in establishing theoretical guarantees for QGANs, including their expressive power, convergence properties, and limits in terms of sample complexity. Together, these directions aim to make QGANs more stable, scalable, and effective in real-world applications.

CHAPTER 5

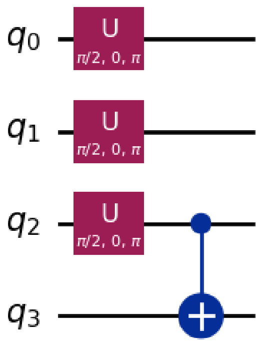
Fully Quantum GAN implementation

This work presents a fully quantum implementation of a GAN [61], based on the code from [1]. All main components, the generator, the discriminator and the real data encoding, are represented by quantum circuits. The objective of this implementation is to train a quantum generator to learn a target probability distribution defined by a quantum circuit that represents real data. Then, amplitude encoding and angle encoding methods are implemented to train classical data.

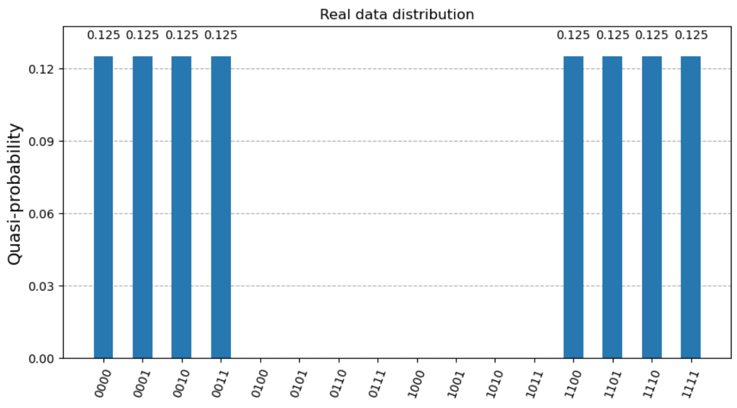
5.1 Components

This implementation includes three core quantum circuits. The first one prepares the quantum state that represents real data. The other two implement the generator and the discriminator:

- **Real Data Circuit:** This circuit encodes the real data into a quantum state based on a predefined probability distribution. Since any arbitrary distribution can be used for training, a simple and shallow distribution is preferred in this work to ensure hardware efficiency by reducing circuit depth and gate count. This circuit is presented in Figure 5.1.
- **Generator Circuit:** This circuit, shown in Figure 4.1a, corresponds to the generator component of the QGAN. It is implemented using the `RealAmplitudes` ansatz with two repetitions, chosen to balance expressivity and hardware efficiency. The circuit is composed of R_Y rotation gates and entangling CNOT gates arranged in a linear topology.
- **Discriminator Circuit:** This circuit, shown in Figure 4.1b, corresponds to the discriminator component of the QGAN. A custom architecture, inspired by the design proposed in [1], is used. It includes R_X, R_Y and R_Z rotation gates, along with CNOT gates that entangle each qubit with the last one. The final classification decision is obtained by measuring this last qubit.



(a) Implementation of the Real Data Circuit.



(b) Probability distribution that represents real data.

Figure 5.1: Real data representation via a quantum circuit.

5.2 Architecture

From the combination of the previous three circuits, two main circuits are constructed (Figure 5.2. The first circuit connects the real data circuit with the discriminator circuit. The second circuit connects the generated data with the discriminator.

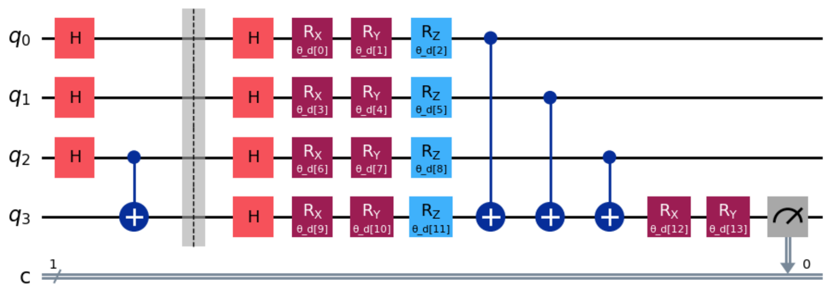
In order to update the parameters of both the generator and discriminator, the training process relies on Quantum Neural Networks (QNN) implemented using the `EstimatorQNN` class provided by Qiskit Machine Learning. This tool allows the efficient computation of expectation values and gradients from variational quantum circuits. In order to operate with it, `StatevectorEstimator` instance is used for simulation, and gradients are computed using the parameter-shift rule. We define the observable as:

$$O = Z \otimes I^{\otimes(n-1)}$$

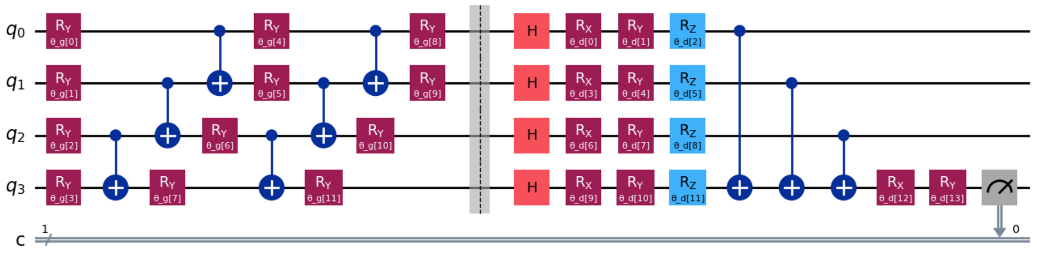
This operator measures the Pauli-Z expectation value on the first qubit. It is used in the `EstimatorQNN` to compute:

$$\langle\psi(\vec{\theta})|O|\psi(\vec{\theta})\rangle = \langle Z\rangle$$

which serves as the model output for training.



(a) Combination of the Real Data Circuit with the Discriminator Circuit.



(b) Combination of the Generator Circuit with the Discriminator Circuit.

Figure 5.2: Two main VQCs of the Fully Quantum GAN implementation.

The training loop is illustrated in Figure 5.3. This setup involves three QNN instances, each corresponding to one of the training steps in the QGAN:

- **Generator QNN ($G(\theta_G; \theta_D)$):** This instance computes gradients with respect to the generator's parameters (θ_G) while keeping the discriminator's parameters (θ_D) fixed. It is based on the circuit that combines the generator with the discriminator. This setup allows the generator to receive feedback from the discriminator and improve its ability to produce realistic samples.

- **Fake Data Discriminator QNN ($FD(\theta_D; \theta_G)$):** This instance computes gradients with respect to the discriminator's parameters (θ_D) while keeping the generator's parameters (θ_G) fixed. It is based on the circuit that combines the generator with the discriminator. This setup allows the discriminator improve classification accuracy of fake data.

- **Real Data Discriminator QNN ($RD(\theta_D)$):** This instance computes gradients with respect to the discriminator's parameters (θ_D). In this setup the generator is not involved. It is based on the circuit that combines the real data preparation circuit with the discriminator. This setup allows the discriminator improve classification accuracy of real data.

5.3 Loss functions

The original loss functions (Equation 3.1 and Equation 3.2) are reconstructed to align with the structure of the implementation and `EstimatorQNN`'s functionality, since `EstimatorQNN`

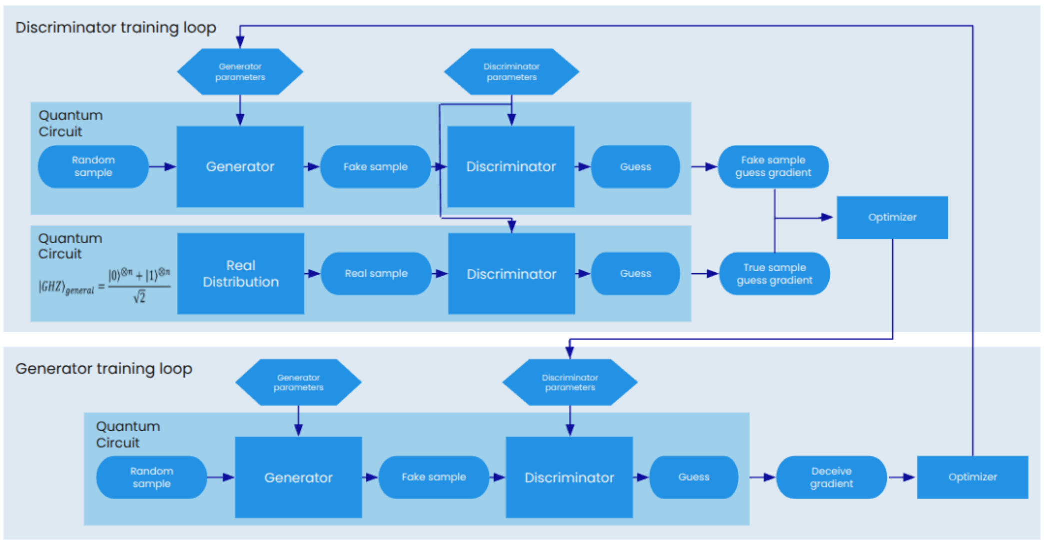


Figure 5.3: Fully Quantum GAN architecture.

returns an expectation value rather than classical probability. In this formulation, a key point is that the final qubit is measured in the Z basis, and the classifier's label depends on the measurement result. By convention, a measurement outcome of $|1\rangle$ corresponds to the real class and $|0\rangle$ to the fake class.

Since the expectation value of the Z operator on the state $|1\rangle$ is -1 and on the state $|0\rangle$ is $+1$, this implies that real samples should yield an expectation value close to -1, while fake samples should initially yield values closer to $+1$. Thus, when the discriminator is evaluated on real data, it should aim to minimize $\langle Z\rangle$, and when evaluated on generated (fake) data, it should aim to maximize $\langle Z\rangle$. Conversely, the generator aims to fool the discriminator into classifying fake data as real, which means minimizing $\langle Z\rangle$ on generated data.

Let $\langle Z\rangle_{RD}$ and $\langle Z\rangle_{FD}$ denote the loss values of evaluating the discriminator with real data and fake data respectively, and let $\langle Z\rangle_G$ denote the loss value of evaluating the generator. With these definitions, we can now express the loss functions as follows.

The discriminator loss function is reconstructed as

$$L_D = \langle Z\rangle_{RD} - \langle Z\rangle_{FD}$$

where $L_D \in [-2, 2]$. This function reflects the discriminator's goal of assigning lower expectation values (closer to -1) to real samples and higher ones (closer to $+1$) to fake samples.

In contrast, the generator loss function is reconstructed as

$$L_G = \langle Z\rangle_G$$

where $L_G \in [-1, 1]$. Minimizing this function encourages the generator to produce data that shifts the expectation value towards -1, making fake samples more likely to be classified as real by the discriminator.

From these definitions[1], the gradients used to update the model parameters are obtained as

$$\nabla_{\vec{\theta_D}} L_D = \nabla_{\vec{\theta_D}} \langle Z \rangle_{RD} - \nabla_{\vec{\theta_D}} \langle Z \rangle_{FD}$$
$$\nabla_{\vec{\theta_G}} L_G = \nabla_{\vec{\theta_G}} \langle Z \rangle_{G}$$

These equations define the gradient updates needed to optimize both networks. In practice, `EstimatorQNN` automatically computes these gradients using the parameter-shift rule.

To maintain consistency with the original classical GAN loss functions, where loss values typically lie within the interval $[-1, 0]$, a rescaling of the quantum losses is performed. The discriminator and generator losses are normalized using the following transformations:

$$L_D = \frac{(\langle Z \rangle_{RD} - \langle Z \rangle_{FD}) - 2}{4}$$

$$L_G = \frac{(\langle Z \rangle_{G}) - 1}{2}$$

where $L_D, L_G \in [-1, 0]$.

5.4 Adam Optimizer

The Adam optimizer (short for Adaptive Moment Estimation) is a widely used optimization algorithm in deep learning [3]. It is designed to combine the advantages of two other popular methods: AdaGrad and RMSProp.

This optimizer works by maintaining two moving averages of the gradients during training: The first one (m_t) builds upon the idea of momentum. It reflects the running average of past gradients and helps the optimizer follow the general direction of descent more consistently:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_{\theta_t} L_t$$

where $\beta_1 \in [0, 1)$ is the decay rate of this moving average, and L_t is the gradient of the loss function with respect to the parameters at time step t.

The second one (v_t) builds upon the idea of adaptive learning rates. It gives a sense of how large the gradients have been on average, and prevents parameters with large gradients from making overly aggressive updates:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left(\nabla_{\theta_t} L_t\right)^2$$

where $\beta_2 \in [0, 1)$ is the decay rate of this moving average.

Both m_t and v_t are initialized at zero. Hence, they are biased towards zero, especially during the initial time steps. To correct this, bias-corrected versions of these estimates are computed:

[1]It is important to emphasize that these formulations are based on the labeling convention where the outcome $|1\rangle$ corresponds to real data. If this convention changes, the signs in the loss functions would need to be adjusted accordingly.

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

The parameters θ are then updated by computing the gradient of the loss function at each time step t:

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$$

where α is the learning rate, and ϵ is a small constant added to prevent division by zero.

These moments are used to adaptively adjust the learning rate for each parameter during training. This allows Adam to perform well on problems with noisy gradients and barren plateaus, helping to maintain training stability and improve convergence.

For this model, the implementation from `tf.keras.optimizers` with default parameter values is used: learning rate $\alpha = 0.001$, exponential decay rate $\beta_1 = 0.9$, exponential decay rate $\beta_2 = 0.999$, and a small constant $\epsilon = 10^{-7}$ to avoid division by zero.

5.5 Performance Calculation

To evaluate how close the generator's output distribution is to the real data distribution, we use the Kullback–Leibler (KL) divergence:

$$D_{\text{KL}}(P_{\text{real}} \parallel P_{\text{model}}) = \sum_x P_{\text{real}}(x) \log\left(\frac{P_{\text{real}}(x)}{P_{\text{model}}(x)}\right)$$

This measure indicates the difference between the real data distribution P_{real} and the model (generator) distribution P_{model}. A lower value of KL divergence suggests that the generator is learning to approximate the real distribution accurately.

When evaluating the generator's performance, trained parameters are applied to the real data circuit and the generator, to then compute their probability distributions. To compute exact performance evaluation, probability distributions are retrieved by using the `Statevector` simulator, meaning that measurements are ideal and noiseless. Regardless, probability distributions can be obtained using quantum hardware by measuring the circuits using samplers, but the evaluated performance would be an approximation due to limited shot counts.

5.6 Training Procedure

The training procedure consists of iteratively updating the generator and discriminator parameters to minimize their respective loss functions. The algorithm of this approach is presented in Algorithm 5.1.

Before starting the loop, we initialize all parameters, optimizers and logging utilities. The generator and discriminator parameters are either loaded from saved values or initialized randomly.

In each training loop, the discriminator and the generator are updated multiple times, following the standard GAN architecture. For the discriminator, the gradient of the loss function is computed using two separate QNNs: one for fake data produced by the generator and one for real data samples. The overall gradient is obtained by subtracting the two. This gradient is then applied using the optimizer to adjust the discriminator parameters.

The generator is trained by applying the gradient of the loss function computed using the QNN constructed from the combined generator-discriminator circuit. In this case, parameters are swapped so the discriminator parameters are fixed and the generator parameters are the weights. The optimizer uses this gradient to update the generator parameters.

To monitor the generator's performance, the Kullback-Leibler (KL) divergence between the generator's output distribution and the target real distribution is computed at each step. When the KL divergence reaches a new minimum, the current generator parameters are stored as the best performing set so far.

Training statistics, including the generator and discriminator loss, KL divergence and best performance to date, are saved at the end of each iteration to a file for later analysis. Th If the training is interrupted, a final block ensures that parameters and logs are safely stored. The training current state, including optimizer state, is stored for further evaluation.

5.7 Quantum Encoding Methods

In order to process classical data such as images using quantum circuits, the data must first be transformed into a quantum-compatible format. This is achieved through encoding methods, which allow classical information to be represented as quantum states. In the context of quantum generative models like QGANs, this step is crucial, as the generator must learn to produce quantum states that reflect real data distributions. Therefore, it is essential to encode classical information into a quantum state in such a way that meaningful classical information can later be recovered through measurement.

Two common encoding strategies in quantum machine learning are *amplitude encoding* and *angle encoding*. Each has its advantages, trade-offs, and practical limitations, particularly in the context of training variational quantum circuits.

5.7.1 Amplitude Encoding

Amplitude encoding is a method used to load classical data into a quantum state. In this approach, the values of a data vector are represented directly by the amplitudes of a quantum state. For a normalized classical vector $\vec{x} = (x_0, x_1, \ldots, x_{2^n-1})$, the corresponding quantum state is:

$$|\psi_{\vec{x}}\rangle = \sum_{i=0}^{2^n-1} x_i \, |i\rangle \, ,$$

where $|i\rangle$ denotes the computational basis states of n qubits.

The main advantage of amplitude encoding is its high storage efficiency. Since N classical values can be encoded into just $\log_2 N$ qubits, it is possible to represent large datasets using relatively few qubits. This compact representation is particularly attractive for near-term quantum devices, where the number of available qubits is limited. Additionally, once encoded, data can be processed in parallel due to quantum superposition, potentially offering exponential speedups.

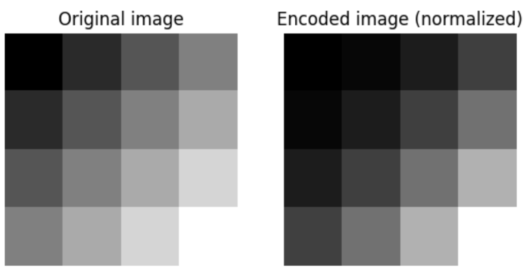


Figure 5.4: Amplitude encoding data corruption example.

However, classical data must be modified so it can be represented as a quantum state vector. Firstly, classical data size must be exactly a number power of 2, which may require padding the data with zeros. Secondly, it has to be normalized so that the sum of squared values equals one. This can lead to distortion or corruption of the original information, as shown in Figure 5.4.

Despite its efficiency, amplitude encoding is often computationally more expensive. Loading this data into amplitudes typically requires complex quantum circuits, increasing gate count and depth. Moreover, each sample requires a separate quantum state preparation circuit. This means that each training loop requires transpilation for each sample, which is highly impractical on current hardware.

Furthermore, a recent study shows that as the number of features increases, quantum states encoded via amplitude encoding tend to concentrate in a small region of the Hilbert space [62]. This phenomenon reduces the diversity of quantum states that can be effectively represented, thus limiting the expressive power of quantum models. This concentration effect contributes to the emergence of barren plateaus and becomes more pronounced when training on large datasets or high-dimensional inputs. This suggests that it may suffer in terms of learning capability and scalability.

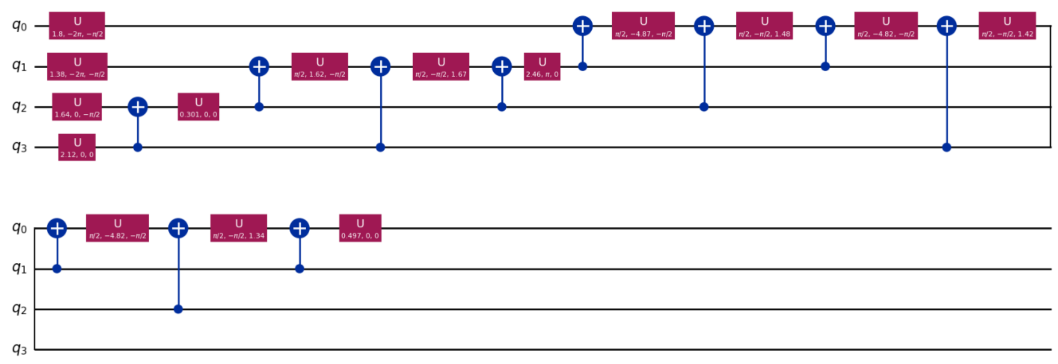
Consequently, due to its impracticality in respect to the use of multiple real samples to train the model, this encoding method is incorporated in the Fully Quantum GAN implementation by substituting the specific real data circuit from the main implementation (Figure 5.1a) with the function `prepare_state` from `qiskit.circuit.library`. This function creates a quantum circuit that prepares a quantum state whose amplitudes match the input parameter values after normalizing them to form a valid probability distribution. An example is presented in Figure 5.5.

5.7.2 Angle Encoding

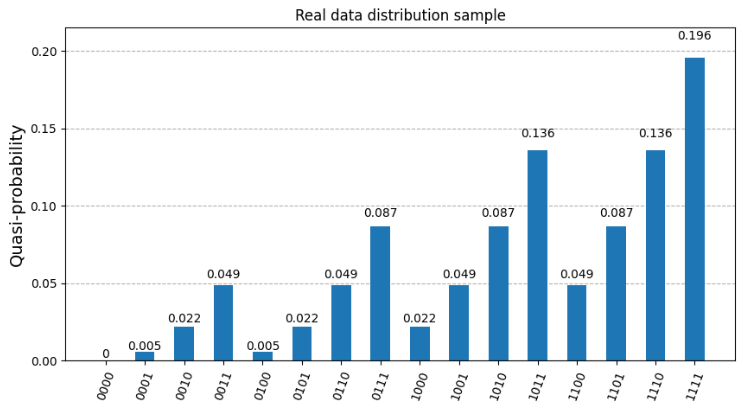
Angle encoding is a method for loading classical data into a quantum circuit by mapping each data value to the angle of a quantum rotation gate. Specifically, classical features are encoded into rotation angles of single-qubit gates, such as $RY(\theta)$, applied to individual qubits (Figure 5.6). For a classical vector $\vec{x} = (x_0, x_1, \ldots, x_{n-1})$, the corresponding quantum state is:

$$|\psi_{\vec{x}}\rangle = \bigotimes_{i=0}^{n-1} RY(x_i)\,|0\rangle^{\otimes n}\,.$$

where each component $x_i \in [0, \pi]$ is used to rotate qubit i.



(a) Created quantum circuit to produce a specific probability distribution.



(b) Real sample encoded into a probability distribution.

Figure 5.5: Amplitude encoding for the real sample shown in Figure 5.4.

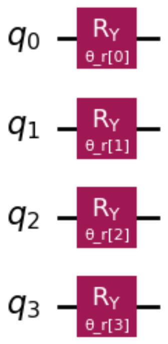


Figure 5.6: Angle encoding Variational Quantum Circuit.

The main advantage of angle encoding is its simplicity and hardware efficiency. It does not require complex state preparation circuits or global normalization, and it supports straightforward gradient-based training. While each qubit encodes only one classical feature (unlike amplitude encoding), angle encoding produces circuits that are typically shallower and less prone to noise, making it more practical for current noisy intermediate-scale quantum (NISQ) devices.

According to comparative studies, despite being less qubit-efficient, angle encoding

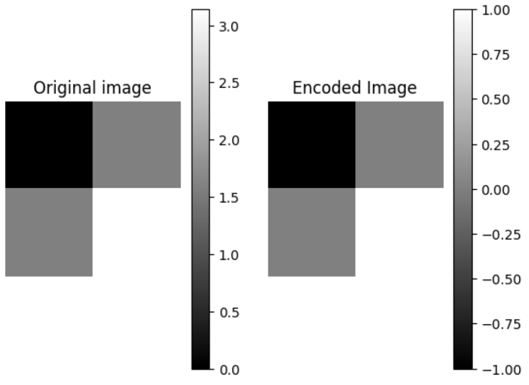


Figure 5.7: Angle encoding for a real data sample.

often matches or exceeds the predictive performance of amplitude encoding in classification tasks, while requiring two to three orders of magnitude less execution time [63].

The integration of angle encoding into the Fully Quantum GAN is achieved by replacing the real data circuit from the original implementation with a variational quantum circuit (VQC), where the circuit parameters represent classical data values. Specifically, the circuit applies RY rotations to each qubit based on the input features. These qubits are then measured using an estimator with Z observables, producing expectation values in the range $[-1, 1]$, which reflect the encoded information (Figure 5.7).

As real data is no longer encoded into a probability distribution, KL divergence is no longer appropriate for evaluating the performance of the generator. Instead, the model's performance must be assessed by comparing the generated outputs with the real data samples, which are represented as images. This is done using a simplified version of the Structural Similarity Index Measure (SSIM) [64]. The function computes global mean, variance, and covariance values to estimate structural similarity between the two images. The final score is transformed into a distance metric, where a lower value indicates a closer match.

This approach enables the model to process a different real data sample in each training iteration by simply updating the parameters of the VQC. As a result, the Fully Quantum GAN can be trained on an entire dataset, allowing it to learn more detailed and complex relationships between samples. This dynamic encoding makes the training process more flexible and expressive while maintaining compatibility with near-term quantum hardware.

```
1  input: Number of training iterations $N$ for the main loop, $N_G$ for the generator
   and $N_D$ for the discriminator
2  input: Learning rates $\alpha_G$ for the generator and $\alpha_D$ for the discriminator
3  input: Generator $G(\theta_G; \theta_D)$, Fake Data Discriminator $FD(\theta_D; \theta_G)$ and Real Data
   Discriminator $RD(\theta_D)$
4  for $N$ training steps
5      for $N_D$ training steps
6          Forward pass (real): compute discriminator expectation value
               $\langle Z \rangle_{RD} = RD(\theta_D)$
7          Forward pass (fake): generate sample and compute expectation value
               $\langle Z \rangle_{FD} = FD(\theta_D; \theta_G)$
8          Compute discriminator loss:
               $L_D = \frac{(\langle Z \rangle_{RD} - \langle Z \rangle_{FD}) - 2}{4}$
9          Backward pass: compute discriminator gradients
               $\nabla_{\theta_D} L_D = \nabla_{\theta_D} \langle Z \rangle_{RD} - \nabla_{\theta_D} \langle Z \rangle_{FD}$
10         Update discriminator parameters with Adam optimizer (Section 5.4):
               $\theta_D \leftarrow \theta_D - \alpha_D \cdot Adam(\nabla_{\theta_D} L_D)$
11     rof
12     for $N_G$ training steps
13         Forward pass: generate sample and evaluate discriminator
               $\langle Z \rangle_G = G(\theta_G; \theta_D)$
14         Compute generator loss:
               $L_G = \frac{\langle Z \rangle_G - 1}{2}$
15         Backward pass: compute generator gradient
               $\nabla_{\theta_G} L_G = \nabla_{\theta_G} \langle Z \rangle_G$
16         Update generator parameters with Adam optimizer (Section 5.4):
               $\theta_G \leftarrow \theta_G - \alpha_G \cdot Adam(\nabla_{\theta_G} L_G)$
17     rof
18     Calculate performance by computing KL divergence
19 rof
```

Algorithm 5.1: Training algorithm for the Fully Quantum GAN Implementation.

CHAPTER 6

Experimental setups

This chapter presents the experimental framework used to evaluate the Fully Quantum GAN (QGAN) implementation in terms of performance, scalability, and practical feasibility. The evaluation begins with a description of the three execution environments used for training and testing: a noiseless simulator, a noise-aware simulator mimicking real hardware, and actual IBM quantum hardware. The design of the experiments includes the selection of quantum circuit configurations, qubit counts, and the classical datasets used for amplitude and angle encoding strategies. To ensure consistent and meaningful evaluation of these experiments, a set of performance metrics is established. Lastly, the chapter discusses important changes made during the evaluation phase, such as architectural adjustments, to obtain significant results when increasing the number of qubits.

6.1 Execution environments

To evaluate the performance and robustness of the Fully Quantum GAN implementation, experiments are conducted across three distinct environments: noiseless (ideal) simulation, noisy simulation based on a real quantum backend and real quantum hardware. Each setup is configured to match different levels of quantum noise and resource constraints, offering insight into how well the system performs under increasingly realistic conditions.

6.1.1 Noiseless Simulation

In the noiseless setup, quantum circuits are simulated using the `StatevectorEstimator` provided by `qiskit.primitives`. This simulator assumes perfect quantum operations with no noise, decoherence, or measurement error. The quantum neural networks (QNNs) for the generator and discriminator are implemented using `EstimatorQNN` with the precision parameter explicitly set to 0.0, indicating that exact expectation values are used during training.

This idealized environment serves as a theoretical baseline, showing the best possible performance of the model without hardware-induced imperfections.

6.1.2 Noisy Simulation

To approximate realistic execution on quantum hardware, a noisy simulator backend is used. The backend is set to `FakeSherbrooke()` from `qiskit_ibm_runtime.fake_provider`, which mimics the behavior and noise characteristics of IBM's Sherbrooke quantum processor. For execution, the `EstimatorV2` interface is used to estimate expectation values, and `EstimatorQNN` with the default precision value[1] (0.015625) is used to calculate gradients.

Circuit transpilation and optimization are performed using a preset pass manager from `qiskit.transpiler.preset_passmanagers`, configured with the target backend and the highest optimization level (level 3). This process adapts each circuit to the hardware's native layout while aiming to reduce noise. At this optimization level, the pass manager aims to minimize gate count and depth as much as possible, at the cost of a longer compilation time. In fact, the last Z gate from the discriminator quantum circuit (Figure 4.1b) was removed due to optimization, thereby adapting the circuit to an optimized design.

This setup provides a realistic evaluation of performance under hardware noise and transpilation effects, without the cost and real limitations of actual quantum hardware.

6.1.3 Real Quantum Hardware: IBM Sherbrooke

To train the Fully Quantum GAN on real hardware, the same implementation used for the noisy simulation is employed. However, instead of using `FakeSherbrooke()`, the real quantum backend `ibm_sherbrooke` is selected. Access to the device is provided via the `QiskitRuntimeService` interface.

IBM Sherbrooke is a superconducting quantum computer developed by IBM and hosted at the PINQ^2 facility in Bromont, Quebec, in collaboration with the Institut quantique at the Université de Sherbrooke [65]. This system is part of IBM's open access program, meaning that researchers and students can use it through the IBM Quantum platform.

This backend consists of a superconducting transmon processor hosted in the IBM Quantum System One computer. The processor is based on the Eagle R3 design [66], which features 127 qubits. Its architecture is optimized for stability, coherence and scalability of quantum operations. It uses a *heavy-hex* lattice layout, where each qubit connects to two or three neighbors in a hexagonal tiling (Figure 6.1). This design balances improved entangling capability with reduced hardware complexity and lower crosstalk compared to fully connected layouts.

A key innovation in this system is the use of Echoed Cross-Resonance (ECR) gates for two-qubit entanglement, chosen to improve calibration stability while lowering leakage and coherent errors [67]. Single-qubit gates are implemented via fast microwave pulses (SX gates), and readout is performed using dedicated resonators.

Error rates and coherence times directly affect circuit performance. These are presented in Table 6.1. As can be observed, two-qubit gate fidelity remains the primary bottleneck in circuit scalability due to both error rates and gate duration.

However, Sherbrooke's use of ECR gates and stable calibration has enabled modest improvements in quantum volume and error-resilient performance when scaling to programs with tens of qubits. These specifications make Sherbrooke a robust platform for exploring

[1]The precision value reflects the quantization of expectation values due to finite sampling.

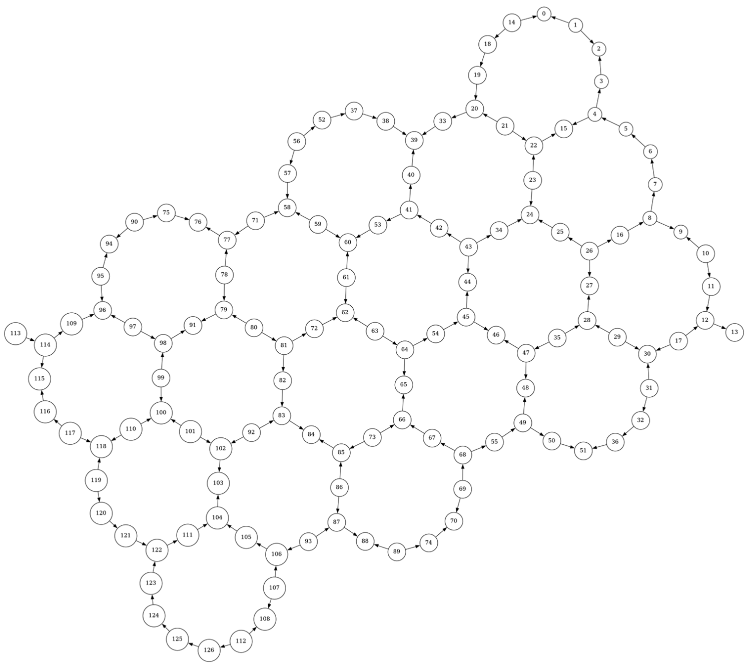


Figure 6.1: Hexagonal layout of Eagle R3 quantum processor.

Metric	Min	Median	Max
Relaxation time T_1 (μs)	76.3	261.6	484.1
Dephasing time T_2 (μs)	16.4	188.1	636.6
Qubit frequency (GHz)	4.455	4.794	5.058
Anharmonicity (GHz)	-0.32	-0.31	-0.27
Readout assignment error	0.0034	0.0201	0.1797
Pr(meas 0 | prep 1)	0.0049	0.0185	0.3237
Pr(meas 1 | prep 0)	0.0019	0.0166	0.1338
Readout length (ns)	1216	1216	1216
ID gate error	1.078×10^{-4}	2.195×10^{-4}	2.238×10^{-3}
RZ gate error	0.0	0.0	0.0
SX gate error	1.074×10^{-4}	2.195×10^{-4}	2.238×10^{-4}
Pauli-X gate error	1.078×10^{-4}	2.195×10^{-4}	2.238×10^{-3}
ECR gate error	0026	0059	0.0365
ECR gate time (ns)	341.3	533.3	881.8

Table 6.1: Key Per-Qubit Metrics for IBM Sherbrooke (Eagle R3, 127 qubits).

mid-depth quantum circuits while mitigating error accumulation due to its balance of low error rates and strong qubit connectivity.

This experimental setup exposes the model to real-world noise, gate errors and hardware limitations. It provides a more accurate evaluation of how it performs outside of ideal or approximated simulations, reflecting the challenges of working with current quantum devices.

6.2 Design of the experiments

To comprehensively assess the performance and behavior of the Fully Quantum GAN across various conditions, a series of evaluations were conducted. These experiments are designed to test the scalability, robustness, and applicability of the model in both theoretical and practical scenarios.

6.2.1 Different Environment Evaluation

To assess the practical performance of the Fully Quantum GAN under different operational conditions, the model was executed using 4 qubits in three environments: noiseless simulation, noisy simulation, and real quantum hardware. This configuration serves as a baseline for evaluating how quantum noise and hardware constraints impact training quality, efficiency, and output.

6.2.2 Scalability Evaluation

To understand how the Fully Quantum GAN scales with increasing quantum resources, the model was evaluated using 4, 8, and 16 qubits in each execution environment. Each environment provides a distinct perspective on the challenges and feasibility of scaling quantum generative models.

- **Noiseless simulation:** Using an ideal simulator without noise effects, the scalability of the model can be studied in isolation from hardware imperfections. As the number of qubits increases, the generator and discriminator circuits grow in size and complexity, requiring longer training times and more memory for `Statevector` calculations.
- **Noisy simulation:** By introducing realistic noise in simulation, we can evaluate an approximation of how error rates, decoherence and crosstalk affect model performance at different scales. As qubit count increases, deeper circuits lead to greater cumulative error, which can degrade training stability and output quality. This setup highlights the limitations of scaling on near-term quantum devices.

These experiments collectively provide a multi-perspective understanding of the scalability of quantum GANs, identifying both algorithmic and hardware-specific bottlenecks as the system grows.

6.2.3 Image Generation with Quantum GANs

The practical application of QGANs to image generation was explored through two different encoding strategies:

- **Amplitude encoding:** Images were encoded into quantum states where pixel intensities determine the state amplitudes. This method was tested with images of size 4×4 and 16×16, corresponding to 4 and 8 qubits respectively. These evaluations were conducted under noiseless simulation.

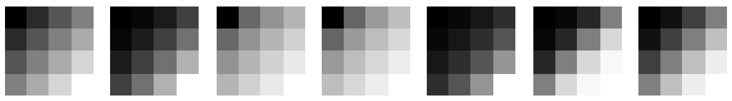


Figure 6.2: Image training dataset for quantum encoding methods.

- **Angle encoding:** Image pixel values were mapped to rotation angles in variational quantum circuits, using RY gates. Datasets of sizes 2×2 and 2×4 were used for 4 and 8 qubit setups, respectively. All angle encoding evaluations were also performed in noiseless simulation.

The dataset used for training consisted of grayscale gradient images, where each image smoothly transitions from darker to lighter shades of gray in different intensities (Figure 6.2. This type of dataset provides continuous variation in pixel intensity, allowing the model to learn a complex relation across the images.

6.3 Evaluation Metrics

The following metrics were used to evaluate the models:

- **Training time per epoch:** Measures in seconds the computational cost and efficiency of training loops in each setup.
- **Loss values:** Generator and discriminator losses were tracked across epochs. All loss values were rescaled to lie within the range $[-1, 0]$ for consistency across evaluation methods.
- **Performance metrics:**
  - *KL Divergence:* Used to measure the similarity between the generated and target probability distributions. This is applicable to distribution learning tasks and amplitude-encoded image generation.
  - *SSIM Distance:* Applied to angle-encoded image tasks to assess the similarity between generated and real images. The distance was computed as $1 - \text{SSIM}$ so that 0 is best, matching the behavior of the KL Divergence metric.
- **Improvement metric:** Measures the overall improvement of the training performance by calculating the difference between the first epoch's performance and the best epoch's performance.

This comprehensive evaluation framework allows for detailed insights into model training behavior, learning stability, hardware compatibility, and encoding-specific performance.

6.4 Changes During Evaluation

During the evaluation phase of the Fully Quantum GAN, several important challenges emerged as the number of qubits increased. While the model performed reasonably well

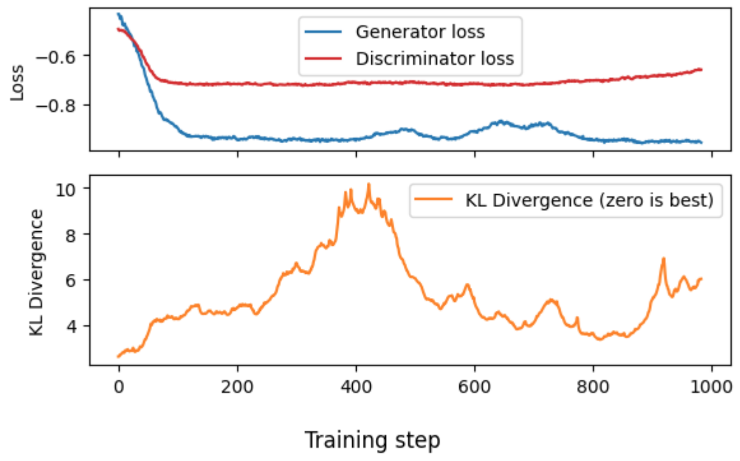


Figure 6.3: Training instability in 8 qubit noisy simulation.

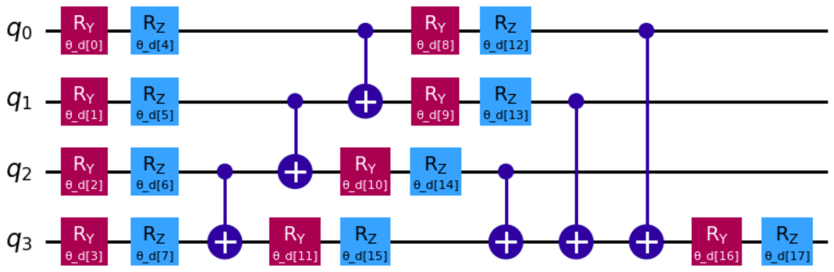


Figure 6.4: Redesigned discriminator incorporating `EfficientSU`.

with 4 qubits, training stability and convergence degraded significantly when scaling up to 8 or 16 qubits. In many of these cases, it was observed that the generator consistently produced outputs that the discriminator could not effectively differentiate. This is a common effect in Generative Adversarial Networks (explained in Section 3.4), that leads to training instability and unsatisfactory results as shown in Figure 6.3.

This issue was traced back to limitations in the initial discriminator design (Figure 4.1b). The original discriminator architecture was optimized for small-scale experiments with only 2 to 3 qubits. It featured a simple layout with limited connectivity between qubits, which proved insufficient to capture the more complex patterns and correlations present in higher-dimensional quantum states. As a result, the discriminator lacked the expressive power needed to provide meaningful feedback to the generator during training.

To address this, the discriminator circuit was redesigned to increase its expressivity. This involved using Qiskit's `EfficientSU` ansatz with one repetition to add more entangling gates and improve the qubit connectivity (Figure 6.4). In turn, this allowed the discriminator to represent more complex decision boundaries while maintaining a similar gate count. To match the discriminator's number of parameters, generator circuit was also increased in gates by using `RealAmplitudes` with three repetitions.

However, these improvements came at the cost of greater circuit depth that increased noise and computation time. Despite these trade-offs, the updated discriminator significantly improved performance across evaluations with higher qubit counts.

It is important to note that the 4-qubit experiments were conducted using the original, low-complexity discriminator circuit. However, for experiments involving more than 4 qubits, the redesigned discriminator was used.

CHAPTER 7

Analysis of the results

This chapter presents a detailed analysis of the results obtained from the experimental evaluations described earlier. Each experiment is assessed using the defined metrics: training time per epoch, loss values, and performance scores (KL Divergence or SSIM Distance). The discussion aims to interpret model behavior, identify trends, and understand the impact of different settings such as execution environments, qubit scaling, and encoding methods.

7.1 Different Environment Evaluation

Let us begin by analyzing the results regarding the evaluation across different execution environments. As presented in Table 7.1, the noiseless simulation provides a reference under ideal conditions, where the model does not suffer from any hardware-related noise. In this environment, the Fully Quantum GAN converged relatively quickly in 278 epochs and achieved the best performance value of 0.1089. This shows that the model can learn stably and efficiently when free from physical constraints. In the noisy simulation, realistic quantum errors and expectation value approximation (due to finite sampling) are introduced. These factors increased the training time per epoch and slightly reduced performance. The model still improved significantly, but required more training (480 epochs) to converge and reached a lower performance of 0.2719. The real hardware setup showed the most challenging conditions. Performance was affected by real-world noise, calibration fluctuations and measurement errors. As a result, the model required 689 epochs to reach its best performance, which was 0.6921. Despite these difficulties, the model was able to improve during training in all three environments, meaning that the model has a certain level of robustness even under limited and noisy conditions.

Analyzing the performance metric graph in Figure 7.1, is visible the presence of peaks in the training progress. This volatility indicates moments during training where the generator struggles to accurately reproduce the real data distribution. These fluctuations are expected in variational training processes. Despite temporary increases in the metric value, the overall trend shows a progressive reduction, suggesting that the model is actively adjusting its parameters to minimize the performance metric. This behavior demonstrates that the generator is learning from the feedback provided by the discriminator and improving its

Metric	Noiseless Sim.	Noisy Sim.	Real Hardware
Training time per epoch (s)	0.3	22.2	165.0
Best performance	0.1089	0.2719	0.6921
Best epoch	278	480	689
Improvement	4.6824	4.5193	4.0992

Table 7.1: Comparison of metrics for 4 qubit training across environments.

Metric	4 qubits	8 qubits	16 qubits
Noiseless Simulation			
Training time per epoch (s)	0.3	7.7	32.6
Best performance	0.1089	0.3938	0.5370
Best epoch	278	676	701
Improvement	4.6824	2.4610	4.4812
Noisy Simulation			
Training time per epoch (s)	22.2	28.7	-
Best performance	0.2696	1.2421	-
Best epoch	940	929	-
Improvement	4.5216	1.6127	-

Table 7.2: Scalability evaluation across different qubit counts and execution environments.

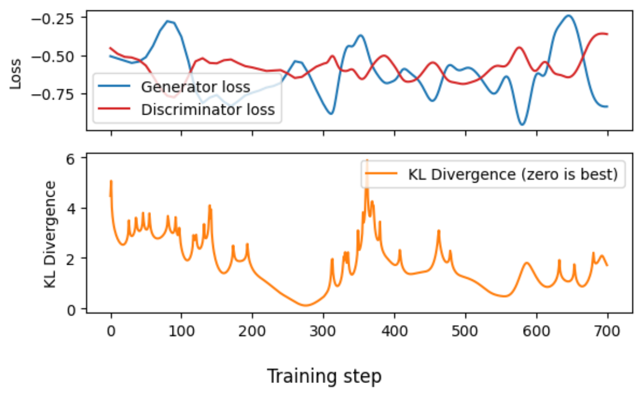
ability to generate samples closer to the target distribution over time. This is apparent in the results shown in Figure 7.2.

Training time in real hardware is also significantly affected by hardware access latency. One of the main issues was job queuing, as submitted tasks often faced long pending times before execution began. Although the actual circuit execution on the quantum processor was relatively fast, the total turnaround time was significantly extended due to the time required to configure the backend. In some cases, runtime errors caused jobs to fail entirely, requiring re-submission and further waiting. After approximately 32 hours of cumulative usage, the workload was deprioritized due to time quota limitations. This resulted in endless waiting times, therefore having to stop execution in real hardware at epoch number 689.

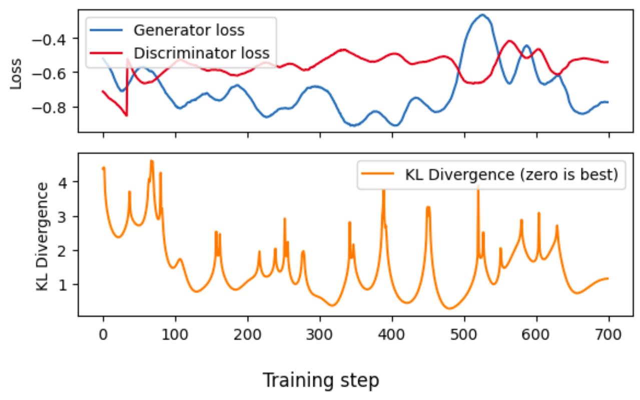
7.2 Scalability Evaluation

Now, we analyze how the Fully Quantum GAN scales when increasing the number of qubits by examining the metrics presented in Table 7.2. As expected, increasing the number of qubits significantly affects both training time and model performance. In noiseless simulation, training time per epoch increases from 0.3 seconds for 4 qubits to 7.69 seconds for 8 qubits, and up to 32.6 seconds for 16 qubits. This is due to the exponential growth of circuit complexity and the number of parameters to optimize. Despite this, the model is still able to learn effectively, although the best performance gradually degrades from 0.1089 with 4 qubits to 0.537 with 16 qubits. This reduction in performance may be due to the increasing difficulty of exploring and optimizing a larger state space.

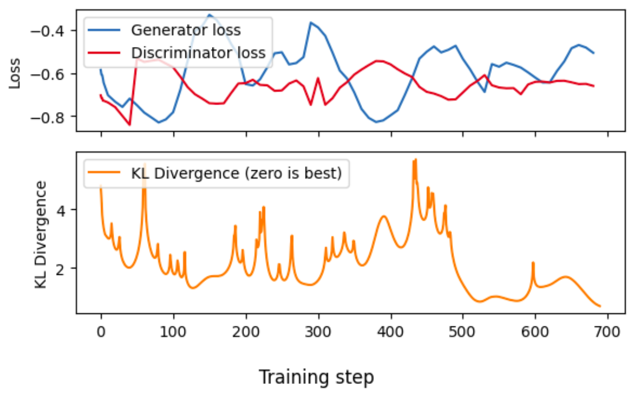
In noisy simulation, the challenges become more pronounced. Training time per epoch increases moderately from 22.2 to 28.7 seconds when scaling from 4 to 8 qubits. However, performance degrades sharply, with the best value worsening from 0.2696 to 1.2421. This



(a) Noiseless simulation execution.



(b) Noisy simulation execution.

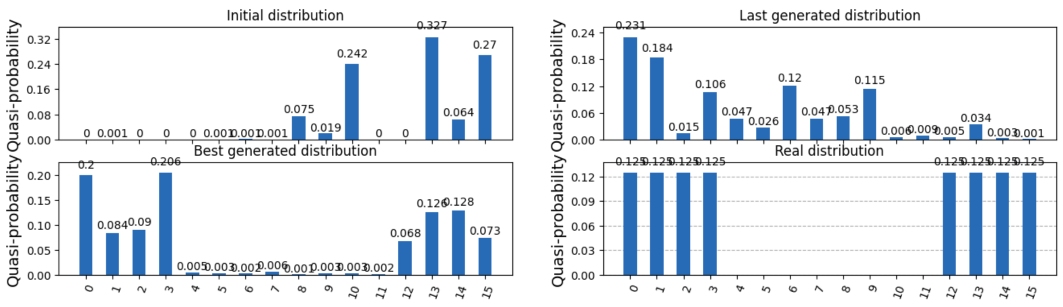


(c) Real hardware execution.

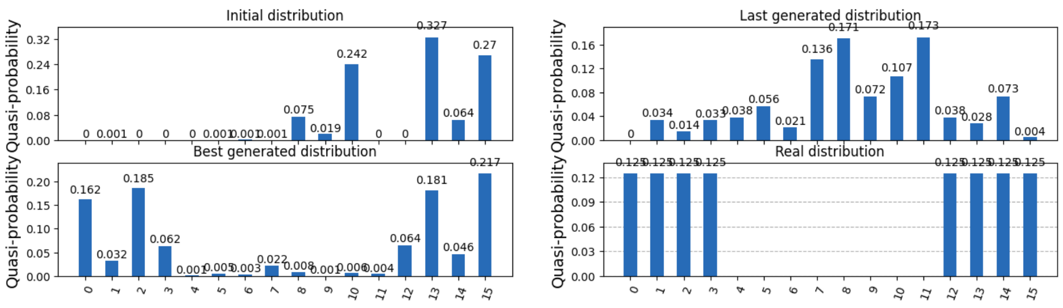
Figure 7.1: 4 qubit performance in three different execution environments.

suggests that as circuits grow deeper, noise effects such as gate errors, decoherence, and qubit crosstalk accumulate and significantly impact learning. The model's ability to improve also drops, as seen in the lower improvement score. This shows current hardware limitations make large-scale QGANs challenging to train without error mitigation.

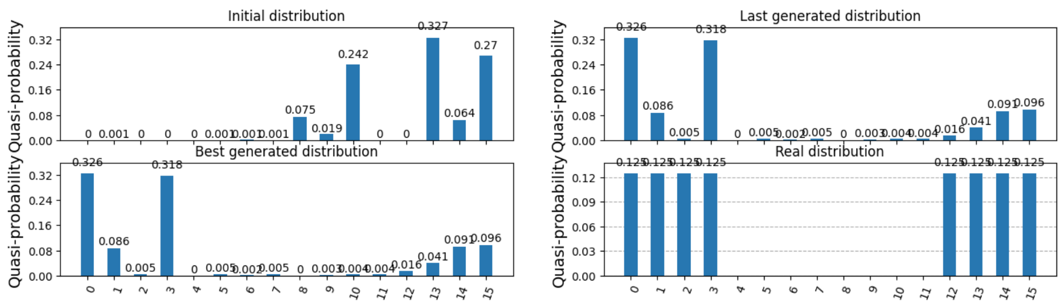
In regard to learning rate, in Figure 7.5 the 8-qubit noisy simulation displays less smooth training curves compared to the other setups shown in Figure 7.3, primarily due to the



(a) Noiseless simulation execution.



(b) Noisy simulation execution.

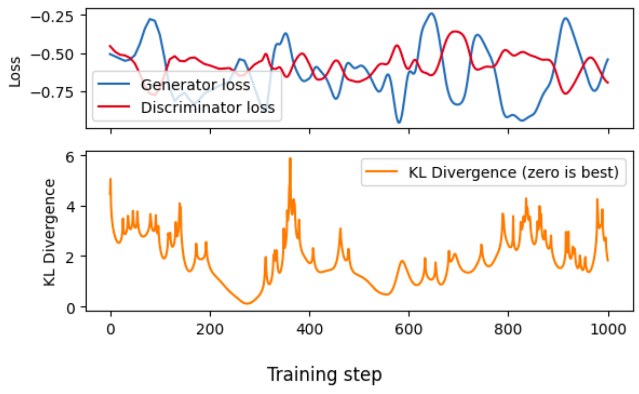


(c) Real hardware execution.

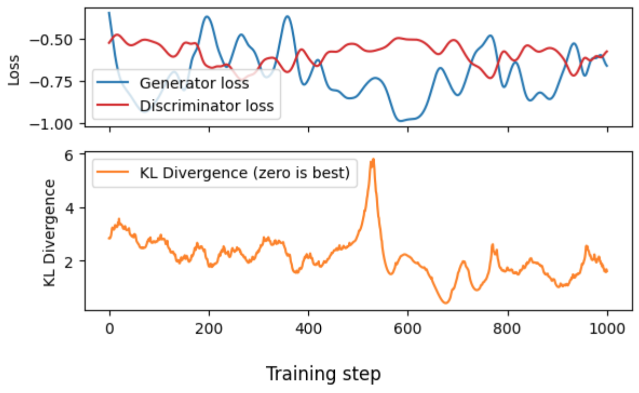
Figure 7.2: 4 qubit probability distribution results in three different execution environments.

accumulation of quantum noise during circuit execution. Errors such as decoherence, gate infidelities, and readout inaccuracies introduce fluctuations that distort the expected gradient flow. As a result, the training progress becomes irregular, with the performance metric exhibiting abrupt changes instead of a gradual, consistent improvement. This deviation from ideal behavior reflects how noise interferes with the optimization trajectory, making it more difficult for the model to converge steadily toward better results. This is noticeable in Figure 7.6 when compared with results obtained in noiseless simulation from Figure 7.4.

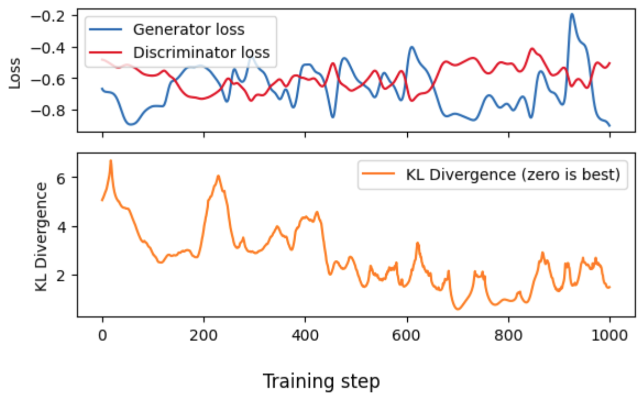
The 16-qubit noisy simulation was not completed due to significant computational and practical constraints. Noisy simulation is extremely resource-intensive compared to noiseless simulation due to output estimation via shots. Moreover, as shown in the results, learning a larger and noisy probability distribution would require many more epochs to reach reasonable convergence, especially under the instability introduced by quantum noise.



(a) 4 qubit execution.



(b) 8 qubit image execution.



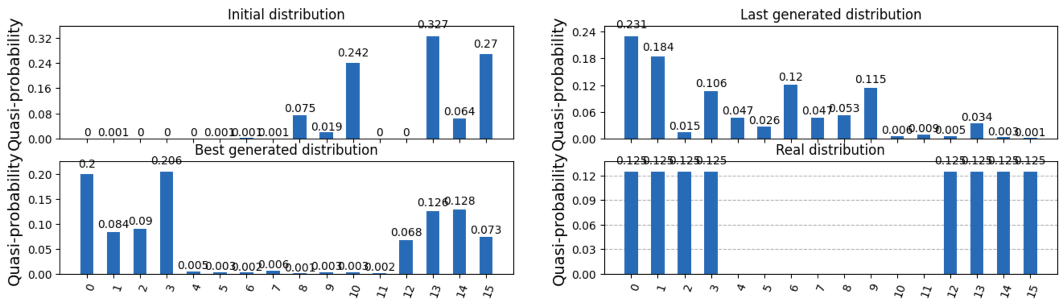
(c) 16 qubit image execution.

Figure 7.3: Noiseless execution performance with different number of qubits.

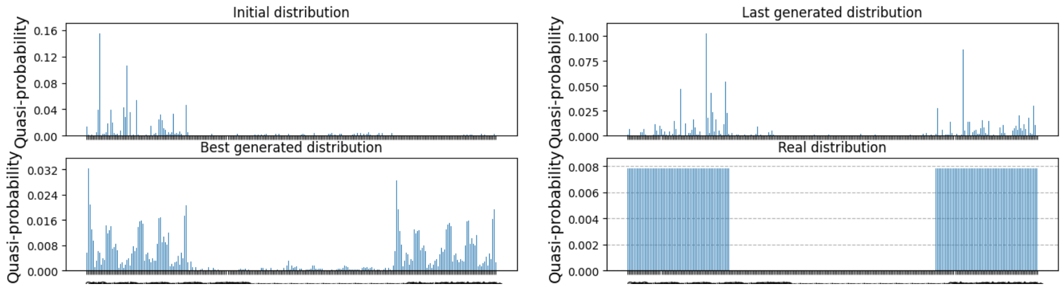
7.3 Image Generation with Quantum GANs

In this experiment, we evaluate the model's ability to learn and generate classical images using amplitude and angle encoding. All metrics are then collected in Table 7.3.

Amplitude encoding method has demonstrated strong representational power, producing promising results when learning a single target image. As displayed in Figure 7.8, the



(a) 4 qubit execution.



(b) 8 qubit execution.

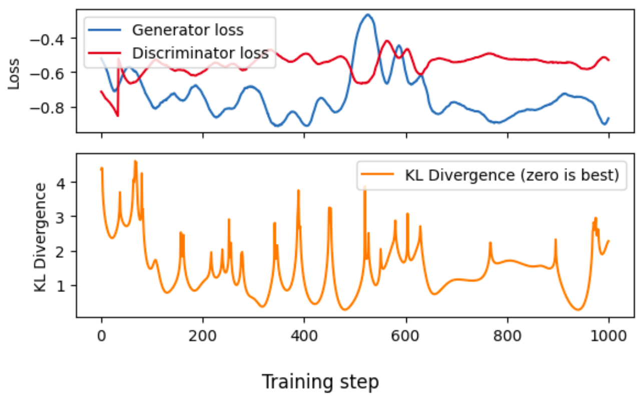
Figure 7.4: Noiseless execution results with different number of qubits.

Metric	**4 qubits**	**8 qubits**
Amplitude Encoding		
Training time per epoch (s)	0.5	6.6
Best performance (KL Div.)	0.0934	0.5171
Best epoch	959	558
Improvement	1.4875	2.3780
Angle Encoding		
Training time per epoch (s)	0.2	0.8
Best performance (SSIM Dist.)	0.3762	0.2779
Best epoch	83	491
Improvement	0.8941	1.2235

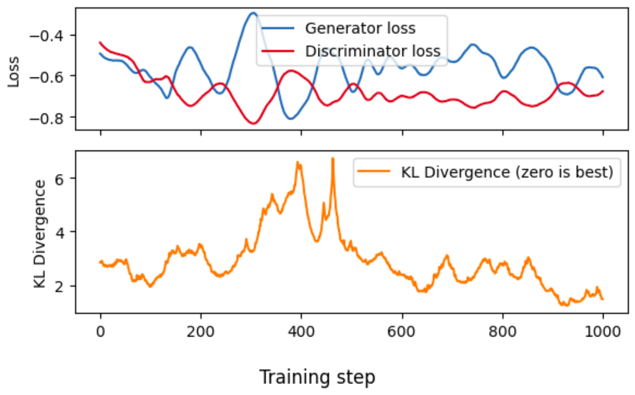
Table 7.3: Evaluation of QGAN image generation using different encoding methods.

improvement in generated images is notable. The performance metric also increased from 0.0934 for 4 qubits to 0.5171 for 8 qubits, suggesting that learning accuracy degrades with scale. As presented in Figure 7.7, the best epoch dropped from 959 to 558, while the improvement score rose from 1.4875 to 2.3780, indicating a faster but less precise convergence. This was expected since a larger data size normally requires longer training to acquire similar results.

However, this encoding training was significantly slower compared to other methods, primarily due to the complexity of the quantum circuit used for state preparation. This is reflected in the increase in training time per epoch from 0.5 seconds with 4 qubits to 6.6 seconds with 8 qubits. As the number of qubits increased, the depth and gate count of the real circuit grew rapidly, making both simulation and transpilation computationally



(a) 4 qubit execution.



(b) 8 qubit execution.

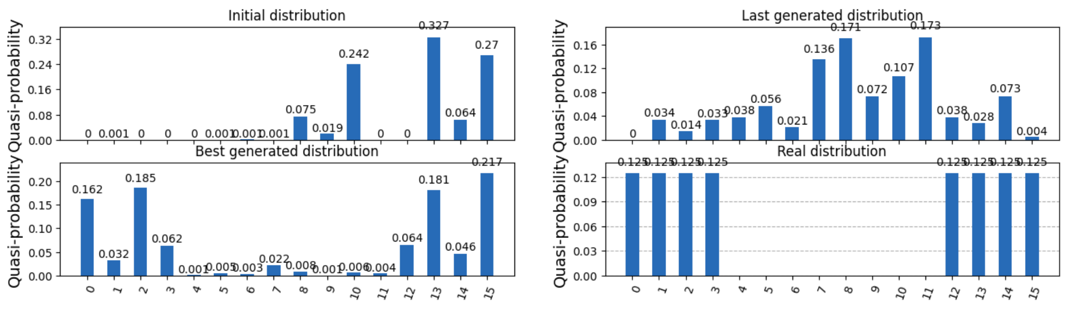
Figure 7.5: Noisy execution of performance with different number of qubits.

expensive.

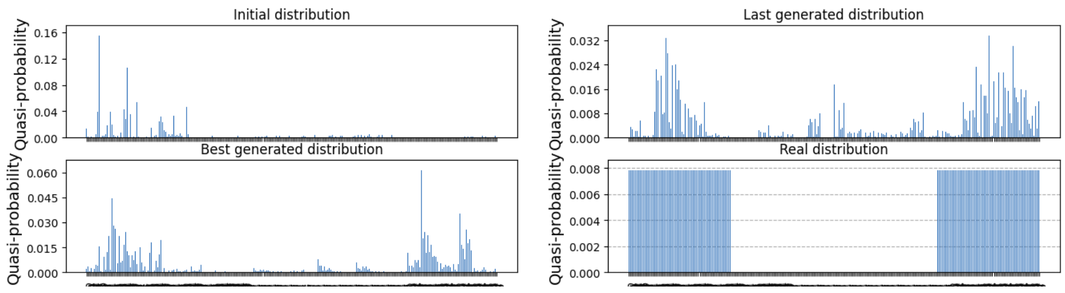
While effective in capturing target images, this method proved impractical for training on datasets, since each new image would require a full circuit recompilation and transpilation at every training iteration, greatly increasing execution time and resource usage.

Angle encoding has offered a more scalable and flexible approach for image generation. It allows more efficient training with image datasets by simply modifying gate parameters, avoiding the need for full circuit recompilation. This has reduced transpilation overhead and enabled faster training, even as the number of qubits increased. In this case, training time per epoch grew modestly from 0.2 seconds for 4 qubits to 0.8 seconds for 8 qubits. Although the representational capacity is more limited compared to amplitude encoding, especially for high-resolution images, angle encoding has demonstrated stable training. As illustrated in Figure 7.10, the model was able to capture essential image features effectively while being trained with a dataset of multiple images.

Small fluctuations are shown in Figure 7.9, since this volatility in the discriminator loss and performance metrics represents the change of real data sample (image) each iteration. Interestingly, the SSIM distance improved from 0.3762 for 4 qubits to 0.2779 for 8 qubits. This improvement is attributed to the enhanced expressivity of the discriminator circuit



(a) 4 qubit execution.



(b) 8 qubit execution.

Figure 7.6: Noisy execution of results with different number of qubits.

used in the 8-qubit training. The original discriminator, designed for low-qubit setups, struggles to capture complex relationships within the dataset, leading to training instability as shown in Figure 7.9a. In contrast, the updated circuit enabled the discriminator to effectively handle dataset sample relations, even with the increased data complexity at 8 qubits. The best epoch moved from 83 to 491, and the improvement score rose from 0.8941 to 1.2235, reflecting the model's improved ability to learn more intricate data structures.

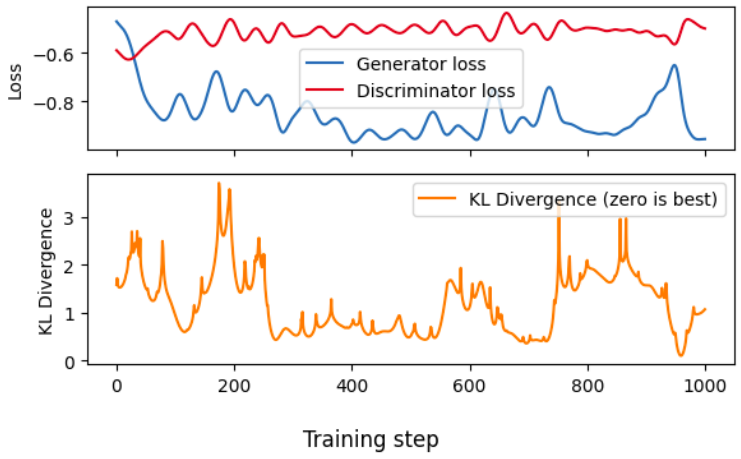
Overall, the efficiency and compatibility of angle encoding with variational learning made it better suited for dynamic training scenarios, suggesting that angle encoding is a promising approach worth conducting a more thorough investigation.

Despite the relative advantages of angle encoding, training with 16 qubits in a noiseless execution environment still proved to be computationally demanding. Both encoding methods faced significant performance bottlenecks at this scale due to increased circuit complexity, leading to face longer execution times and limited model responsiveness. These challenges highlight the current limitations of training fully quantum models on high-dimensional inputs and emphasize the need for more hardware-efficient circuit designs and improved noise resilience.

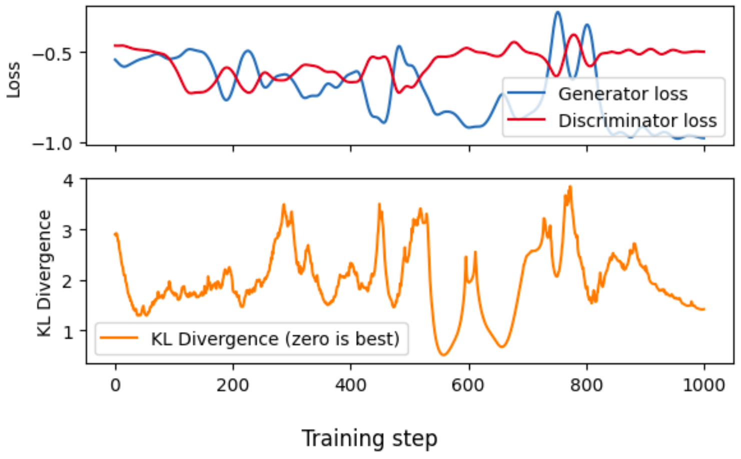
7.4 Summary of the results

This chapter has provided an in-depth analysis of the performance of the Fully Quantum GAN across different configurations and conditions. Key findings show that while noiseless simulations allow stable and efficient convergence, noisy simulations and real hardware introduce substantial training challenges due to quantum noise and hardware limitations.

Scalability tests reveal that increasing the number of qubits significantly raises circuit



(a) 4×4 image (4 qubit) execution.

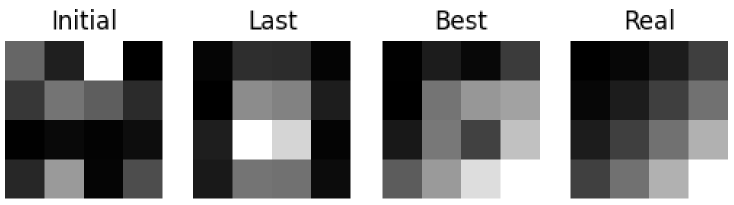


(b) 16×16 image (8 qubit) execution.

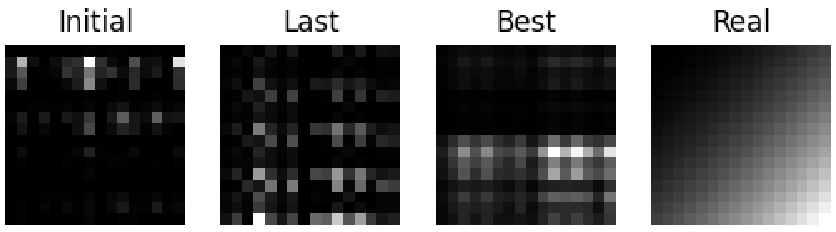
Figure 7.7: Amplitude encoding performance with different image sizes.

complexity, training time, and susceptibility to error, making optimization more difficult in both simulation and real-world conditions. Nonetheless, with improved discriminator design, the model was still capable of learning from more complex data.

Regarding quantum encoding, the comparison of the different methods demonstrated that while amplitude encoding yields high-quality results for fixed targets, it becomes impractical for datasets due to heavy circuit compilation requirements. In contrast, angle encoding showed strong potential for scalable and dynamic training, achieving lower SSIM distance and greater training stability when paired with a more expressive discriminator. Overall, the results highlight the importance of architectural adaptability, encoding choices, and hardware-aware design in advancing the practical training of quantum generative models.

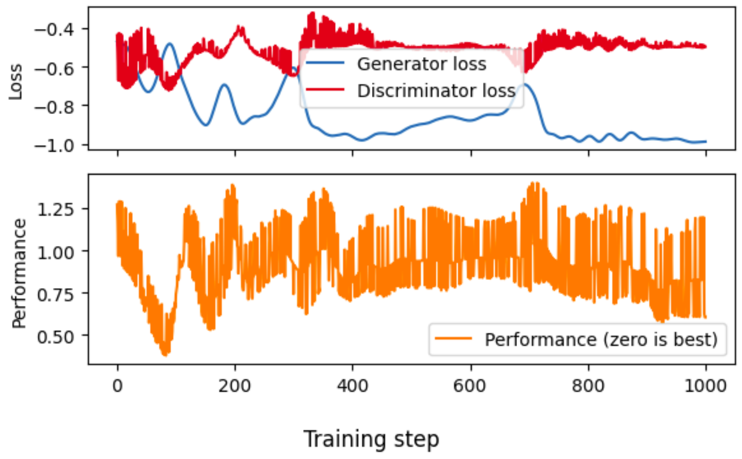


(a) 4×4 image (4 qubit) execution.

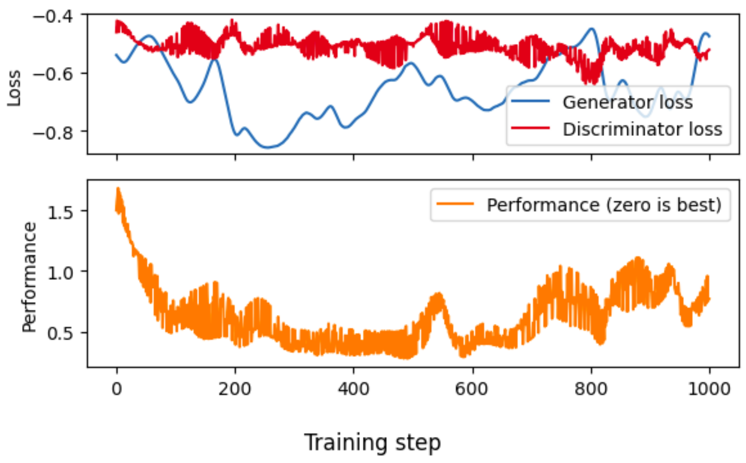


(b) 16×16 image (8 qubit) execution.

Figure 7.8: Amplitude encoding results with different image sizes.

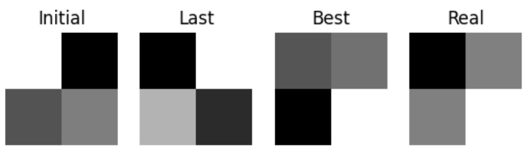


(a) 2×2 dataset (4 qubit) execution.

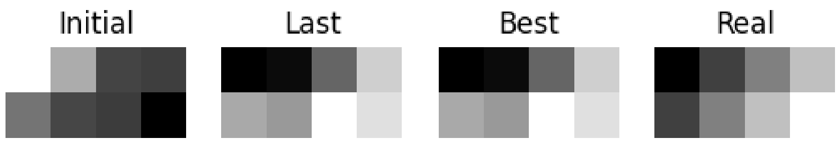


(b) 2×4 dataset (8 qubit) execution.

Figure 7.9: Angle encoding performance with different image sizes.



(a) 2×2 dataset (4 qubit) execution.



(b) 2×4 dataset (8 qubit) execution.

Figure 7.10: Angle encoding results with different image sizes.

CHAPTER 8

Conclusions and Future Work

In this work, a Fully Quantum Generative Adversarial Network was designed, implemented, and evaluated using IBM Quantum hardware, with both the generator and discriminator modeled as variational quantum circuits. The architecture was adapted for execution on current Noisy Intermediate-Scale Quantum devices, incorporating classical optimization techniques and exploring two classical-to-quantum encoding methods: amplitude and angle encoding. The model was tested across three environments (noiseless simulation, noisy simulation and real quantum hardware) to assess its robustness, scalability, and practicality.

Results show that while noiseless simulations demonstrate successful training convergence and effective data generation, noisy and hardware executions reveal challenges such as circuit depth sensitivity, noise-induced instability, and the limitations of quantum encoding fidelity. Nonetheless, the fully quantum approach proved viable for simple generative tasks and highlighted the trade-offs involved in working with real quantum devices. These findings support the potential of QGANs for future quantum machine learning applications.

Regarding future work, several directions can be pursued to enhance the capabilities and applicability of Fully Quantum GANs. One key avenue is the exploration of alternative ansatz designs specifically tailored for generative modeling and quantum classification tasks, with a focus on improving expressivity and noise resilience while maintaining hardware efficiency. Future research should also target real-world applications by training QGANs on domain-specific datasets, such as those from quantum chemistry, high-energy physics or healthcare, where the generation of structured data or anomaly detection could benefit from quantum-enhanced modeling.

Most importantly, further evaluations should prioritize execution on real quantum hardware, especially on cutting-edge platforms. In this context, the upcoming quantum computer to be installed in Donostia, the IBM Quantum System Two powered by the IBM Quantum Heron processor, represents a unique and strategic opportunity. Being based in Donostia, it would be highly beneficial to access this local infrastructure to conduct larger scale and lower noise QGAN experiments, enabling deeper analysis of quantum model performance in practical and scalable scenarios. This future access to next-generation quantum systems will be essential to test and refine QGAN architectures in more realistic and demanding settings.

Bibliography

[1] Davis Clarke and Frank Harkins. Qiskit notebooks quantum machine learning qgan 2023. See pages ix, 7, and 21.

[2] Baidu Inc. Institute for Quantum Computing. Qnn research tutorial for barren plateaus. See pages ix, 14.

[3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. See pages ix, 14, and 25.

[4] Samuele Pedrielli, Christopher J. Anders, Lena Funcke, Karl Jansen, Kim A. Nicoli, and Shinichi Nakajima. Bayesian parameter shift rule in variational quantum eigensolvers, 2025. See pages ix, 17.

[5] Yuxuan Du, Xinbiao Wang, Naixu Guo, Zhan Yu, Yang Qian, Kaining Zhang, Min-Hsiu Hsieh, Patrick Rebentrost, and Dacheng Tao. Quantum machine learning: A hands-on tutorial for machine learning practitioners and researchers, 2025. See page 1.

[6] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014. See pages 1, 5.

[7] Tuan A. Ngo, Tuyen Nguyen, and Truong Cong Thang. A survey of recent advances in quantum generative adversarial networks. *Electronics*, 12(4), 2023. See pages 1, 19.

[8] Divya Saxena and Jiannong Cao. Generative adversarial networks (gans survey): Challenges, solutions, and future directions, 2023. See pages 8, 9, and 19.

[9] Md Mashrur Arifin, Md Shoaib Ahmed, Tanmai Kumar Ghosh, Ikteder Akhand Udoy, Jun Zhuang, and Jyh haw Yeh. A survey on the application of generative adversarial networks in cybersecurity: Prospective, direction and open research scopes, 2024. See page 9.

[10] Junhan Qin. Review of ansatz designing techniques for variational quantum algorithms, 2022. See page 13.

[11] Martín Larocca, Supanut Thanasilp, Samson Wang, Kunal Sharma, Jacob Biamonte, Patrick J. Coles, Lukasz Cincio, Jarrod R. McClean, Zoë Holmes, and M. Cerezo. Barren plateaus in variational quantum computing. *Nature Reviews Physics*, 7(4):174–189, March 2025. See page 14.

[12] Lorenzo Leone, Salvatore F.E. Oliviero, Lukasz Cincio, and M. Cerezo. On the practical usefulness of the hardware efficient ansatz. *Quantum*, 8:1395, July 2024. See page 15.

[13] Abhinav Anand, Philipp Schleich, Sumner Alperin-Lea, Phillip W. K. Jensen, Sukin Sim, Manuel Díaz-Tinoco, Jakob S. Kottmann, Matthias Degroote, Artur F. Izmaylov, and Alán Aspuru-Guzik. A quantum computing view on unitary coupled cluster theory. *Chemical Society Reviews*, 51(5):1659–1684, 2022. See page 15.

[14] Ankit Kulshrestha, Xiaoyuan Liu, Hayato Ushijima-Mwesigwa, Bao Bach, and Ilya Safro. Qadaprune: Adaptive parameter pruning for training variational quantum circuits, 2024. See page 15.

[15] Sukin Sim, Jonathan Romero, Jérôme F Gonthier, and Alexander A Kunitsa. Adaptive pruning-based optimization of parameterized quantum circuits. *Quantum Science and Technology*, 6(2):025019, March 2021. See page 15.

[16] Mohammadsaleh Pajuhanfard, Rasoul Kiani, and Victor S. Sheng. Survey of quantum generative adversarial networks (qgan) to generate images. *Mathematics*, 12(23), 2024. See pages 15, 18.

[17] Christa Zoufal, Aurélien Lucchi, and Stefan Woerner. Quantum generative adversarial networks for learning and loading random distributions. *npj Quantum Information*, 5(1), November 2019. See page 15.

[18] Jonathan Romero and Alan Aspuru-Guzik. Variational quantum generators: Generative adversarial quantum machine learning for continuous distributions, 2019. See page 15.

[19] Haozhen Situ, Zhimin He, Yuyi Wang, Lvzhou Li, and Shenggen Zheng. Quantum generative adversarial network for generating discrete distribution. *Information Sciences*, 538:193–208, October 2020. See pages 15, 16.

[20] Abhinav Anand, Jonathan Romero, Matthias Degroote, and Alan Aspuru-Guzik. Noise robustness and experimental demonstration of a quantum generative adversarial network for continuous distributions. *Advanced Quantum Technologies*, 4(5), April 2021. See page 15.

[21] M. Emre Sahin, Edoardo Altamura, Oscar Wallis, Stephen P. Wood, Anton Dekusar, Declan A. Millar, Takashi Imamichi, Atsushi Matsuo, and Stefano Mensa. Qiskit Machine Learning: an open-source library for quantum machine learning tasks at scale on quantum hardware and classical simulators. *arXiv e-prints*, page arXiv:2505.17756, May 2025. See page 15.

[22] Okuyan Boga et al. Qiskit machine learning tutorial: Pytorch qgan implementation. See page 15.

[23] Albha O'Dwyer Boyle and Reza Nikandish. A hybrid quantum-classical generative adversarial network for near-term quantum processors, 2023. See page 16.

[24] Sahil Nokhwal, Suman Nokhwal, Saurabh Pahune, and Ankit Chaudhary. Quantum generative adversarial networks: Bridging classical and quantum realms, 2023. See page 16.

[25] Seth Lloyd and Christian Weedbrook. Quantum generative adversarial learning. *Physical Review Letters*, 121(4), July 2018. See page 16.

[26] Pierre-Luc Dallaire-Demers and Nathan Killoran. Quantum generative adversarial networks. *Physical Review A*, 98(1), July 2018. See page 16.

[27] Jinfeng Zeng, Yufeng Wu, Jin-Guo Liu, Lei Wang, and Jiangping Hu. Learning and inference on generative adversarial quantum circuits. *Physical Review A*, 99(5), May 2019. See page 16.

[28] S. E. Rasmussen and N. T. Zinner. Multiqubit state learning with entangling quantum generative adversarial networks. *Physical Review A*, 106(3), September 2022. See page 16.

[29] Murphy Yuezhen Niu, Alexander Zlokapa, Michael Broughton, Sergio Boixo, Masoud Mohseni, Vadim Smelyanskyi, and Hartmut Neven. Entangling quantum generative adversarial networks, 2021. See page 16.

[30] Shouvanik Chakrabarti, Yiming Huang, Tongyang Li, Soheil Feizi, and Xiaodi Wu. Quantum wasserstein generative adversarial networks, 2019. See page 16.

[31] Carlos Bravo-Prieto, Julien Baglio, Marco Cè, Anthony Francis, Dorota M. Grabowska, and Stefano Carrazza. Style-based quantum generative adversarial networks for monte carlo events. *Quantum*, 6:777, August 2022. See page 16.

[32] Kouhei Nakaji and Naoki Yamamoto. Quantum semi-supervised generative adversarial network for enhanced data classification. *Scientific Reports*, 11(1), October 2021. See page 16.

[33] Kerstin Beer and Gabriel Müller. Dissipative quantum generative adversarial networks, 2021. See page 16.

[34] He Wang and Jin Wang. Dissipation-driven quantum generative adversarial networks, 2024. See page 16.

[35] Runqiu Shu, Xusheng Xu, Man-Hong Yung, and Wei Cui. Variational quantum circuits enhanced generative adversarial network, 2024. See page 16.

[36] Leeseok Kim, Seth Lloyd, and Milad Marvian. Hamiltonian quantum generative adversarial networks, 2024. See page 16.

[37] Ling Hu, Shu-Hao Wu, Weizhou Cai, Yuwei Ma, Xianghao Mu, Yuan Xu, Haiyan Wang, Yipu Song, Dong-Ling Deng, Chang-Ling Zou, and Luyan Sun. Quantum generative adversarial learning in a superconducting quantum circuit. *Science Advances*, 5(1), January 2019. See page 16.

[38] He-Liang Huang, Yuxuan Du, Ming Gong, Youwei Zhao, Yulin Wu, Chaoyue Wang, Shaowei Li, Futian Liang, Jin Lin, Yu Xu, Rui Yang, Tongliang Liu, Min-Hsiu Hsieh, Hui Deng, Hao Rong, Cheng-Zhi Peng, Chao-Yang Lu, Yu-Ao Chen, Dacheng Tao, Xiaobo Zhu, and Jian-Wei Pan. Experimental quantum generative adversarial networks for image generation. *Physical Review Applied*, 16(2), August 2021. See page 16.

[39] Samwel Sekwao, Jason Iaconis, Claudio Girotto, Martin Roetteler, Minwoo Kang, Donghwi Kim, Seunghyo Noh, Woomin Kyoung, and Kyujin Shin. End-to-end demonstration of quantum generative adversarial networks for steel microstructure image augmentation on a trapped-ion quantum computer, 2025. See page 16.

[40] Nicholas S. DiBrita, Daniel Leeds, Yuqian Huo, Jason Ludmir, and Tirthak Patel. Recon: Reconfiguring analog rydberg atom quantum computers for quantum generative adversarial networks. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '24, page 1–9. ACM, October 2024. See page 16.

[41] Haoran Ma, Liao Ye, Xiaoqing Guo, Fanjie Ruan, Zichao Zhao, Maohui Li, Yuehai Wang, and Jianyi Yang. Quantum generative adversarial networks in a silicon photonic chip with maximum expressibility. *Advanced Quantum Technologies*, September 2024. See page 16.

[42] Tigran Sedrakyan and Alexia Salavrakos. Photonic quantum generative adversarial networks for classical data. *Optica Quantum*, 2(6):458, December 2024. See page 16.

[43] Julien Baglio. Data augmentation experiments with style-based quantum generative adversarial networks on trapped-ion and superconducting-qubit technologies, 2024. See page 16.

[44] David Wierichs, Josh Izaac, Cody Wang, and Cedric Yen-Yu Lin. General parameter-shift rules for quantum gradients. *Quantum*, 6:677, March 2022. See page 17.

[45] Artur F. Izmaylov, Robert A. Lang, and Tzu-Ching Yen. Analytic gradients in variational quantum algorithms: Algebraic extensions of the parameter-shift rule to general unitary transformations. *Physical Review A*, 104(6), December 2021. See page 18.

[46] Shi Jin, Nana Liu, and Yue Yu. Time complexity analysis of quantum difference methods for linear high dimensional and multiscale partial differential equations. *Journal of Computational Physics*, 471:111641, December 2022. See page 18.

[47] Su Yeon Chang, Sofia Vallecorsa, Elías F. Combarro, and Federico Carminati. Quantum generative adversarial networks in a continuous-variable architecture to simulate high energy physics detectors, 2021. See page 18.

[48] Rey Guadarrama, Sergei Gleyzer, Mariia Baidachna, Kyoungchul Kong, Konstantin T. Matchev, Katia Matcheva, Isabel Pedraza, Gopal Ramesh Dahale, and Haydee Hernández-Arellano. Quantum generative adversarial networks for gluon initiated jets generation, 2025. See page 18.

[49] Elie Bermot, Christa Zoufal, Michele Grossi, Julian Schuhmacher, Francesco Tacchino, Sofia Vallecorsa, and Ivano Tavernelli. Quantum generative adversarial networks for anomaly

detection in high energy physics. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, page 331–341. IEEE, September 2023. See page 18.

[50] Prateek Jain and Srinjoy Ganguly. Hybrid quantum generative adversarial networks for molecular simulation and drug discovery, 2022. See page 18.

[51] Po-Yu Kao, Ya-Chu Yang, Wei-Yin Chiang, Jen-Yueh Hsiao, Yudong Cao, Alex Aliper, Feng Ren, Alan Aspuru-Guzik, Alex Zhavoronkov, Min-Hsiu Hsieh, and Yen-Chu Lin. Exploring the advantages of quantum generative adversarial networks in generative chemistry. *Journal of Chemical Information and Modeling*, 63(11):3307–3318, May 2023. See page 18.

[52] Daniel Herr, Benjamin Obert, and Matthias Rosenkranz. Anomaly detection with variational quantum generative adversarial networks. *Quantum Science and Technology*, 6(4):045004, July 2021. See page 18.

[53] Minghua Pan, Bin Wang, Xiaoling Tao, Shenggen Zheng, Haozhen Situ, and Lvzhou Li. Detection and evaluation of abnormal user behavior based on quantum generation adversarial network, 2023. See page 18.

[54] Amine Assouel, Antoine Jacquier, and Alexei Kondratyev. A quantum generative adversarial network for distributions, 2021. See page 18.

[55] Mingyu Lee, Myeongjin Shin, Junseo Lee, and Kabgyun Jeong. Mutual information maximizing quantum generative adversarial network and its applications in finance, 2023. See page 18.

[56] Santanu Ganguly. Implementing quantum generative adversarial network (qgan) and qcbm in finance, 2025. See page 18.

[57] Samuel A. Stein, Betis Baheri, Daniel Chen, Ying Mao, Qiang Guan, Ang Li, Bo Fang, and Shuai Xu. Qugan: A quantum state fidelity based generative adversarial network. In *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*, page 71–81. IEEE, October 2021. See page 18.

[58] Cheng Chu, Grant Skipper, Martin Swany, and Fan Chen. Iqgan: Robust quantum generative adversarial network for image synthesis on nisq devices, 2023. See page 18.

[59] Daniel Silver, Tirthak Patel, William Cutler, Aditya Ranjan, Harshitta Gandhi, and Devesh Tiwari. Mosaiq: Quantum generative adversarial networks for image generation on nisq computers, 2023. See page 18.

[60] Chia-Hsiang Lin and Si-Sheng Young. Hyperking: Quantum-classical generative adversarial networks for hyperspectral image restoration, 2025. See page 19.

[61] Beñat Burutaran Maiz. Fully quantum gan implementation. Available at `https://github.com/bburuta/Qiskit-IBM`. See page 21.

[62] Xin Wang, Yabo Wang, Bo Qi, and Rebing Wu. Limitations of amplitude encoding on quantum classification, 2025. See page 28.

[63] Minati Rath and Hema Date. Quantum data encoding: A comparative analysis of classical-to-quantum mapping techniques and their impact on machine learning accuracy, 2023. See page 30.

[64] Jim Nilsson and Tomas Akenine-Möller. Understanding ssim, 2020. See page 30.

[65] Katia Moskvitch. Ibm and pinq2 unveil utility-scale quantum computer in québec. See page 34.

[66] Muhammad AbuGhanem. Ibm quantum computers: evolution, performance, and future directions. *The Journal of Supercomputing*, 81(5), April 2025. See page 34.

[67] Matthias Steffen, Jerry Chow, Sarah Sheldon, and Doug McClure. A new eagle in the poughkeepsie quantum datacenter: Ibm quantum's most performant system yet. See page 34.