Analyzing LatentQGAN: A Hybrid Quantum-Classical GAN with Autoencoders

Gregorio Piqué
gregorio.pique@edu.unifi.it

Abstract

Generative models have achieved significant success in capturing complex data distributions and generating realistic samples across various domains. However, training these models, particularly Generative Adversarial Networks (GANs), remains computationally demanding, especially for high-dimensional data. Quantum computing offers a promising avenue to address these challenges by leveraging principles such as superposition and entanglement. In this context, LatentQGAN integrates classical and quantum computing to enhance GAN training, enabling the generation of more expressive and complex data distributions beyond the capabilities of purely classical approaches.

1. Introduction

Generative models are a class of machine learning algorithms designed to learn the underlying probability distribution of a dataset, enabling them to generate new samples that resemble the training data. These models have many applications, like image synthesis, natural language processing, data augmentation, anomaly detection, and creative content generation. Early generative models, such as *Variational Autoencoders* (VAEs) [7], faced challenges in generating high-quality samples and accurately capturing intricate data distributions.

Generative Adversarial Networks (GANs) [3], employ a unique adversarial training process, where two neural networks are trained against each other: a generator creates synthetic data and a discriminator distinguishes between real and fake data. This competitive approach pushes both networks to improve, resulting in the generator producing increasingly realistic samples and the discriminator becoming more adept at identifying fakes. Compared to previous generative models, GANs demonstrated a remarkable ability to generate sharp, high-fidelity samples, leading to significant advances in image synthesis and other domains. However, GANs also suffer from limitations like unstable and challenging training, often requiring careful tuning of hyperparameters and network architectures. Other common issues are mode collapse, where the generator produces a limited variety of samples, and vanishing gradients.

Some of these issues might be partially solved by using different approaches based on the emerging and promising field of quantum computing: this uses principles from quantum mechanics to potentially solve problems too complex for classical computers. Quantum bits (qubits), unlike classical bits, can exist in multiple states at once (superposition), and can become correlated (entanglement). These properties allow for the development of potentially faster algorithms. Quantum machine learning (QML) explores how quantum computing can improve classical machine learning, by creating new models and techniques that learn more efficiently or solve problems beyond classical machine learning's capabilities. Potential advantages include faster training, better model accuracy, and the ability to work with larger, more complex data distributions.

The paper [15] presents *LatentQGAN*, a novel hybrid quantum-classical GAN architecture for image generation. This approach leverages the potential advantages of quantum computing for generative modeling, that may help in overcoming the difficulties and limitations of classical methods. This document provides a detailed study of the LatentQGAN model, including an introduction to the background theory relevant to the paper in Section 2, a brief presentation of the related work and existing similar methods in Section 3, an in-depth description of the LatentQGAN method in Section 4, and ultimately a presentation of experimental results and their discussion in Section 5.

2. Background Theory

This section introduces the background theory relevant to the paper, covering fundamental quantum computing concepts [6, 13] and the operational principles of classical GANs.

2.1. Quantum Mechanics and Computing

Quantum machine learning is a promising field that may offer significant advantages over classical approaches by leveraging the principles of quantum mechanics, a fundamental theory that describes the behavior of particles at the smallest scales. Unlike classical computing, which processes information using bits confined to either 0 or 1, quantum computing relies on *qubits*, which exhibit unique properties that enable more complex and efficient computations.

2.1.1 Relevant Properties of Quantum Mechanics

The key quantum mechanical principles that differentiate quantum computation from classical computation are *superposition*, *entanglement*, *quantum interference*, and *measurement-caused state collapse*.

Superposition shows the first (theoretical) advantage of the quantum approach on the classical one. A classical bit is either 0 or 1, but a qubit can exist in a superposition of both states. Mathematically, a qubit state can be represented as

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

where α and β are complex numbers or *amplitudes* satisfying $|\alpha|^2+|\beta|^2 = 1$, and $|0\rangle$ and $|1\rangle$ are the *basis states*. This means that until a measurement is performed, a qubit holds a combination of both states, allowing quantum algorithms to process multiple possibilities simultaneously.

Another fundamental property is entanglement, which allows two or more qubits to become entangled, meaning that their states are no longer independent. Altering one qubit instantaneously affects the state of the other. This property enables quantum computers to encode complex correlations that are too complex to represent classically.

Quantum states can also interfere with each other, by behaving like waves. This interaction can either strengthen or cancel out different quantum possibilities. By controlling this interference, quantum algorithms can enhance the probability of desired outcomes, potentially leading to significant speedups compared to classical methods for specific problems.

One last fundamental principle of quantum computing is that, unlike classical systems, where data exists in a well-defined state at all times, performing quantum measurements permanently alter the quantum states, causing them to "collapse" into a definite classical state, i.e., into one of the basis states (either $|0\rangle$ or $|1\rangle$). This collapse introduces a probabilistic element to quantum computations, making repeated sampling and circuit runs necessary for extracting useful information.

2.1.2 Advantages of Quantum Computation and Machine Learning

The aforementioned properties allow quantum computing and quantum machine learning to process information in ways that classical systems cannot efficiently replicate.

With superposition, quantum computers can encode and manipulate large amounts of information using relatively few qubits, making them well-suited for tasks that involve complex distributions or require efficient representation of high-dimensional data, particularly useful in QML. This is given by the exponential nature of quantum superposition, for which a system with n qubits can represent 2^n possible states simultaneously. Another key advantage of quantum computing is efficient sampling from probability distributions. While classical methods might require costly iterative techniques (like *Markov chain Monte Carlo* methods), quantum systems can generate samples directly through measurement. Additionally, quantum parallelism allows operations on all superposed states at once, enabling large-scale computations in a single step. Quantum entanglement further enhances computational capabilities by encoding richer correlations between qubits. This property can improve generative modeling and generalization.

The manipulation of qubits is performed with computational models called *quantum circuits*, consisting of a sequence of quantum gates which perform operations that have no direct classical equivalent. The Hadamard gate (H) creates superposition: it transforms a qubit from a definite computational basis state into an equal probability mixture of both, enabling quantum parallelism. Another essential class of operations is given by rotation gates (R_θ), which apply a rotation around a specified axis on the *Bloch sphere*. These gates are parameterized by an angle θ and introduce tunable quantum transformations. Additionally, the Controlled-Z (*C-Z*) gate is a two-qubit operation that applies a phase flip to the target qubit when the control qubit is in the $|1\rangle$ state. This gate is crucial for entanglement, allowing qubits to establish non-classical correlations that are fundamental to quantum algorithms and generative models.

Many QML approaches rely on *variational quantum circuits* (VQCs), which are parametrized trainable quantum models optimized with classical methods; these allow quantum-enhanced generative models, such as quantum GANs.

2.1.3 Challenges of Quantum-based Approaches

Despite the potential advantages, constructing quantum circuits and training models on real quantum computers remains a challenging task, primarily due to scalability issues and the limitations of current quantum hardware.

In the context of data generation, the challenge lies in representing high-dimensional data distributions, a complex task even in classical settings and particularly difficult for quantum systems. These difficulties are caused by the limited number of available qubits in quantum circuits and the inherent complexity of efficiently encoding classical data into quantum states [15].

Another challenge in training models on real quantum computers arises from the limitations of current hardware, including *qubit decoherence*, *gate errors*, and *connectivity constraints* [10]. Qubit decoherence occurs because qubits are highly sensitive to their environment, making them vulnerable to even minimal noise or disturbances. This can lead to the loss of quantum properties such as superposition and entanglement, ultimately degrading computational reliability. Additionally, quantum gates are prone to errors, introducing noise into computations. The more gates a circuit contains, the higher the likelihood of accumulated errors. To address this, *Quantum Error Correction* techniques have been developed to detect and correct errors without directly measuring the logical qubit, which would otherwise collapse the quantum state. These methods typically rely on redundancy, encoding a single logical qubit into multiple physical qubits [8, 9]. Connectivity constraints further complicate quantum computing, as qubits are not always fully interconnected due to the physical architecture of quantum processors. This means that some qubits can only interact with a limited set of others, requiring additional swap operations to allow interactions between non-adjacent qubits.

The *LatentQGAN* method, as presented in the studied paper, addresses these challenges by designing a quantum neural network that minimizes their impact as much as possible, for example, by using shallow circuits.

2.2. GANs

Generative Adversarial Networks are two-player models consisting of a generator *G* and a discriminator *D* trained adversarially, where each attempts to outperform the other. The generator synthesizes data samples, while the discriminator tries to distinguish between genuine and generated examples, classifying them as 1 or 0 respectively. Training involves optimizing an objective function modeled as the binary cross-entropy loss, with both models seeking to optimize the same loss in opposite directions, leading to a min-max optimization process:

$$\min_G \max_D \mathcal{L}(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

In theory, training converges to an equilibrium where $p(y = 1|x) = p(y = 0|G(z)) = \frac{1}{2}$ meaning the generator has successfully learned the true data distribution p_{data}. At this point, it produces realistic synthetic data that consistently fool the discriminator, making it unable to distinguish between real and generated samples, with both being classified with equal probability. However, in practice, this ideal scenario is rarely achieved.

2.2.1 Training GANs

Training GANs is highly unstable and sensitive to various factors, often leading to failure when either one model dominates the adversarial game (typically the discriminator), or when *mode collapse* occurs, where *G* learns to produce only a limited subset of data, failing to capture the full diversity of the target distribution.

These failure scenarios are also influenced by the inherent imbalance in GANs, where the discriminator holds a significant advantage over the generator. This is primarily due to two factors: first, the discriminator's task of binary classification is considerably easier than the generator's one of synthesizing realistic data; second, *G* updates its parameters based on indirect feedback from *D*. As a result, the gradient signals received by *G* are often much weaker in magnitude compared to those of *D*, making training more challenging and potentially unstable [4].

The stability of GAN training can be improved through various techniques designed to mitigate the discriminator's advantage and facilitate more effective learning. One approach is reducing the sparsification of model gradients by avoiding operations that induce zero gradients, such as pooling layers or activation functions like *ReLU*, which outputs zero for negative inputs. Another widely used method is *label smoothing*, which involves assigning soft labels instead of binary values (e.g., using 0.1 instead of 0 for fake samples and 0.9 instead of 1 for real samples). This reduces the confidence of *D*, leading to more gradual learning and improved generalization.

Different optimization strategies are also employed to balance training dynamics. The discriminator is often trained using *SGD*, while *G* benefits from *Adam*, which typically accelerates convergence. Additionally, modifying the generator's loss function can prevent vanishing gradient issues. Instead of using the standard loss formulation, $-\log(1 - D(G(z)))$, a non-saturating loss, $-\log D(G(z)))$, is commonly adopted [2], as shown in Figure 1. This adjustment ensures that gradients remain strong in regions where the generator is still producing unrealistic samples, thus promoting more effective learning. While the new loss saturates when $D(G(z)) \approx 1$ (Figure 2), this occurs only when the generator has already learned to produce convincing outputs, aligning with the intended training dynamics.

3. Related Works

The practical use of GAN models often incorporates an autoencoder, a specific type of neural network composed of an encoder and a decoder [1]. The encoder extracts a latent representation of the input data by compressing it into a lower-dimensional space, while the decoder reconstructs the original input from this latent representation. The objective is to generate an output that closely resembles the original input. Autoencoders are commonly built using convolutional layers, which are particularly effective for processing image data. The encoder typically consists of convolutional blocks that progressively reduce the data dimen-

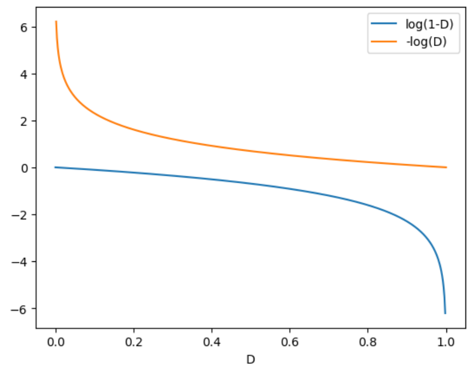

Figure 1. Standard loss (blue) and non-saturating one (orange). Both functions have similar structures and trends.

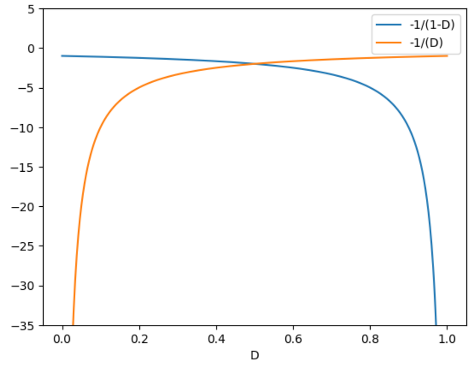

Figure 2. Standard loss gradients (blue) and non-saturating ones (orange). The former saturate where D recognizes synthetic data as fake, the latter where D is fooled by realistic generated samples, i.e. where no significant update is needed.

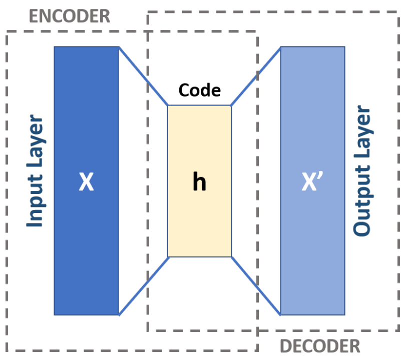

Figure 3. Structure of an autoencoder.

sionality, whereas the decoder employs transposed convolutional layers to perform upsampling and restore the original dimensions. Figure 3 [16] illustrates the typical architecture of an autoencoder. Training involves minimizing the reconstruction error, which quantifies the difference between the input and the output of the model, often measured using the mean squared error (MSE) loss.

Autoencoders are often integrated into GAN models, as seen in *LatentGAN* [11], which benefits from the computational efficiency and reduced overfitting associated with training on lower-dimensional representations. In this approach, a trained autoencoder first compresses the input data into a latent representation, which serves as the target for the GAN generator. The generator produces lower-dimensional outputs that the discriminator evaluates as real or fake. To generate new images, the generator's output is passed through the decoder, which reconstructs the image by mapping it back to the pixel space.

Several studies have explored the advantages of implementing GAN models using quantum computing. One such example is *Q-PatchGAN* [5], which employs a hybrid architecture consisting of a quantum generator and a classical discriminator. The generator is composed of multiple identical quantum circuits, each responsible for generating a specific patch of the output image. While this approach demonstrated promising results on low-dimensional images, such as 8×8 MNIST samples, it faced challenges related to scalability and mode collapse when applied to higher-dimensional data [14].

Another approach is *MosaiQ* [14], which aims to overcome the scalability limitations of previous methods by incorporating principal component analysis (PCA) for dimensionality reduction. This hybrid model consists of a quantum generator, composed of a stack of identical quantum circuits, and a classical discriminator. The generator synthesizes data in a lower-dimensional space, starting from randomly sampled noise, while the discriminator distinguishes between real and generated data by processing both the generator's output and PCA-reduced real images. To generate new images, the generator performs a forward pass, followed by a PCA-inverse transformation to restore the data to the high-dimensional pixel space. Although this approach mitigates scalability and mode collapse issues, its training remains restricted to simulators due to the large number of iterations required to achieve high-quality results [15].

4. LatentQGAN

The following section presents the framework of LatentQGAN, which consists of two separately trained models: an autoencoder, and a hybrid q-GAN model. The structure of the general model is shown in Figure 4.

4.1. Autoencoder

The autoencoder is a CNN-based neural network composed of an encoder and a decoder. The encoder compresses input data using a series of convolutional layers with ReLU activations, followed by a block of linear layers and a final normalization layer, producing an output of shape

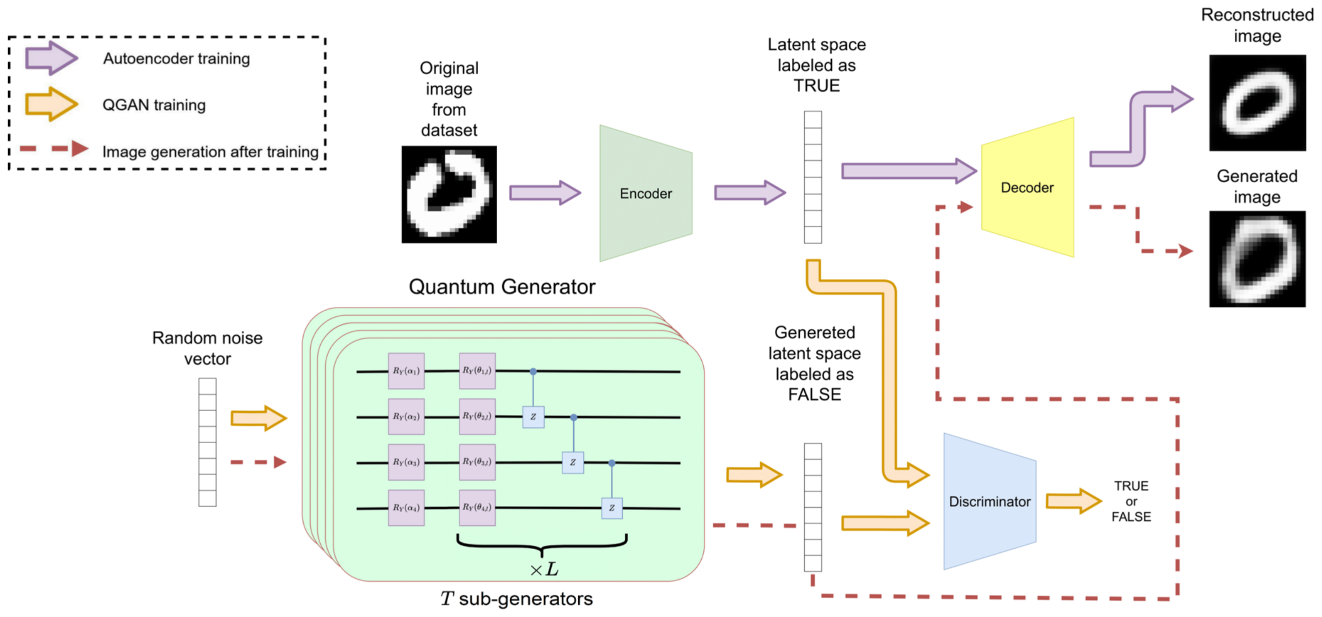

Figure 4. Structure of the LatentQGAN model.

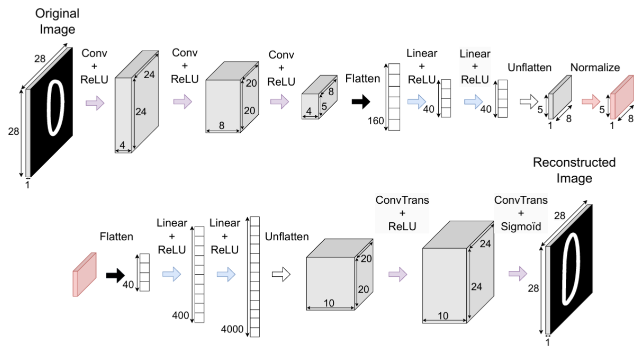

Figure 5. Structure of the autoencoder.

$(5 \times 8 \times 1)$. The decoder takes this latent representation and reconstructs the original input through a sequence of linear layers, transposed convolutions, and a final sigmoid activation, restoring the image to its original shape, such as $(28 \times 28 \times 1)$. An important step in the encoder is the *Normalize* operation, which applies per-line normalization to ensure that for each row i, the sum of its elements satisfies $\sum_{j=1}^{2^{N_G}} z_{i,j} = 1$, where 2^{N_G} represents the row's dimension. This normalization ensures compatibility between the encoder's and the quantum generator's outputs, then processed by the discriminator. The autoencoder's architecture is illustrated in Figure 5.

4.2. Quantum-GAN

The q-GAN model is formed by a classical discriminator and a quantum generator. The discriminator consists of three fully connected linear layers with ReLU activations, followed by a final sigmoid function. As discussed in Section 2.2.1, ReLU activations are often replaced with non-sparsifying alternatives, such as LeakyReLU, to improve gradient flow and stability during training.

The quantum generator follows the structure of *Q-PatchGAN* [5], consisting of a set of T identical subcircuits, each responsible for generating a portion of the output data. This design enables the use of shallower quantum circuits, which, as discussed in Section 2.1.3, reduces the chance of errors during execution. Each LatentQGAN sub-generator operates with 4 qubits, one of which serves as an *ancilla* qubit. The generator begins with an input layer of rotation gates that encode the random noise α, producing the state $|z\rangle = \bigotimes_{i=1}^{N} R_Y(\alpha_i)$. This state is then processed through L parametrized layers, each consisting of $R_Y(\theta_i)$ rotation gates applied to each qubit i and controlled-Z gates sequentially applied between adjacent qubits. Each layer l is parameterized by a weight vector θ_l, representing the trainable model parameters that are optimized during training.

After the parametrized layers, the final quantum state of each sub-generator G_t is given by $|\phi_t(z)\rangle = \sum_{i=0}^{2^N-1} \beta_i|i\rangle$. Before generating the latent representation of a fake data sample, two key operations are performed: *post-selection* and normalization. Post-selection introduces non-linearity into the quantum model by simulating a partial measurement of the generator's output. This step is crucial for enabling complex tasks such as image generation, which would not be possible using only linear transformations.

Post-selection is used because a (direct) partial measurement would collapse the quantum state: the entire state is measured, but only the results where the ancillary qubit is in the $|0\rangle$ state are kept. Finally, a normalization step ensures that the remaining values still represent a valid probability distribution, meaning they sum to 1. The remaining values have a shape of $(5 \times 8 \times 1)$ and are classified by the discriminator, as they represent the latent code of a fake image.

5. Experimental Tests

This section presents the settings of the experiments conducted by [15] and the corresponding results.

5.1. Settings

This subsection presents the experiments settings, in terms of used models, datasets, evaluation metrics, and other details.

5.1.1 Models and Data

The experiments were conducted on both simulators and real quantum devices, specifically the *127-qubit IBM Eagle quantum processor*, using 2048-shot experiments and *M3* error mitigation [8]. The proposed model was tested on the MNIST dataset, which consists of 28×28 grayscale images of handwritten digits. The autoencoder was trained first using the MSE loss, compressing the data into a lower-dimensional latent space (5×8). The quantum generator consists of $T = 5$ sub-circuits, each with $N = 4$ qubits. The variational quantum circuit includes $L = 7$ parametrized layers, leading to a total of $5 \cdot 7 \cdot 4 = 140$ trainable parameters. The authors selected this configuration as an optimal trade-off between quantum computational efficiency (minimizing quantum errors) and model performance (improving the quality of generated samples). The discriminator is implemented as a FCNN with 3681 trainable parameters. The q-GAN was trained using the SGD optimizer and the BCE loss (although the typical choice of optimizer and loss function follows the reasoning discussed in Section 2.2.1).

5.1.2 Parameter-shift Rule

The gradients of the quantum generator are computed using the *parameter-shift* rule [12], an analytical method for calculating gradients with respect to gate parameters, enabling the training of parametrized quantum circuits. This approach estimates gradients by evaluating the circuit at slightly (but macroscopically) shifted parameter values.

Other gradient computation techniques, such as numerical differentiation and automatic differentiation, are not suitable for quantum circuits [12]. Numerical differentiation approximates gradients by applying infinitesimally small shifts to function inputs. However, this method is impractical in quantum computing due to noise and high error rates in current hardware. Automatic differentiation, widely used in classical deep learning, is challenging in quantum settings because intermediate derivatives cannot be stored and reused. This limitation arises because quantum states cannot be measured without collapsing, preventing access to intermediate computations.

The parameter-shift rule, instead, is expressed as $\partial_\theta f = r(f(\theta + s) - f(\theta - s))$, which looks similar to the *finite difference rule* in numerical differentiation, but differs by using a macroscopic shift s typically set as $s = \pi/2$, and a scaling factor of $r = \frac{1}{2}$. This is particularly applicable to quantum circuits with Pauli rotations because they follow trigonometric functions such as $\sin$ and $\cos$, which makes this an exact method [12].

5.1.3 Evaluation Metric

The quality of the generated images was evaluated using the *Fréchet Inception Distance* (FID), a high-level, full-reference metric commonly used to assess the performance of generative models [4]. FID measures the similarity between the distribution of generated images and that of real images. Instead of comparing individual samples, it calculates the mean and covariance of the feature representations extracted from both generated and real images. These feature representations are obtained using a convolutional neural network, typically an *Inception* model. Since FID is a distance metric, lower values indicate better performance, as they suggest that the generated images are more similar to real images. However, FID does not always align perfectly with human perception or semantic similarity [15]. LatentQGAN is compared to other models presented in Section 3, including QPatchGAN, MosaiQ, the classical LatentGAN, and a *RandomDecoder* model. The RandomDecoder generates images by feeding random noise into the trained decoder.

5.2. Results

The experiments, conducted on both simulators and real quantum devices, show that LatentQGAN achieves its best FID between 350 and 700 iterations, with poor image quality before 350 and a decline beyond 700 due to mode collapse. In simulations, LatentQGAN consistently outperforms QPatchGAN and MosaiQ, achieving similar or better image quality with fewer parameters (140 vs. 240) and requiring 676 times fewer iterations than MosaiQ. While other QGAN models generate high-dimensional images effectively on simulators (e.g., MNIST, FashionMNIST, SAT4), their higher accuracy comes at the cost of 10–70 times more parameters, making them untrainable on current NISQ devices. While the autoencoder plays a crucial role in data compression, the generator is essential for learning meaningful patterns in the latent space; without it, models like the RandomDecoder fail to produce coherent images. Additionally, the classical LatentGAN (with the same parameter count) struggles to match LatentQGAN's performance, highlighting the importance of its quantum generator architecture and training process (see Figure 6 and Figure 7).

Results obtained from real quantum hardware show slightly lower accuracy and quality compared to simulations due to noise, but the generated images remain visually recognizable and competitive (Figures 7 and 8).

6. Conclusion

This project involved an in-depth study of the LatentQGAN model for image generation, leveraging the potential advantages of a quantum computing approach. The method uses an autoencoder to project data to and from a lower-dimensional latent space. This enables more efficient handling of high-dimensional data and facilitates the training of quantum models on simulators and real NISQ devices. The results achieved with this approach are promising. Future work may involve extending the method to other tasks, such as time series processing and generation or anomaly detection, as well as exploring the possibility of a fully-quantum implementation.

References

[1] D. Bank, N. Koenigstein, and R. Giryes. Autoencoders, 2021.

[2] W. Fedus, M. Rosca, B. Lakshminarayanan, A. M. Dai, S. Mohamed, and I. Goodfellow. Many paths to equilibrium: Gans do not need to decrease a divergence at every step, 2018.

[3] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks, 2014.

[4] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018.

[5] H.-L. Huang, Y. Du, M. Gong, Y. Zhao, Y. Wu, C. Wang, S. Li, F. Liang, J. Lin, Y. Xu, R. Yang, T. Liu, M.-H. Hsieh, H. Deng, H. Rong, C.-Z. Peng, C.-Y. Lu, Y.-A. Chen, D. Tao, X. Zhu, and J.-W. Pan. Experimental quantum generative adversarial networks for image generation. *Physical Review Applied*, 16(2), Aug. 2021.

[6] A. J., A. Adedoyin, J. Ambrosiano, P. Anisimov, W. Casper, G. Chennupati, C. Coffrin, H. Djidjev, D. Gunter, S. Karra, N. Lemons, S. Lin, A. Malyzhenkov, D. Mascarenas, S. Mniszewski, B. Nadiga, D. O'malley, D. Oyen, S. Pakin, L. Prasad, R. Roberts, P. Romero, N. Santhi, N. Sinitsyn, P. J. Swart, J. G. Wendelberger, B. Yoon, R. Zamora, W. Zhu, S. Eidenbenz, A. Bärtschi, P. J. Coles, M. Vuffray, and A. Y. Lokhov. Quantum algorithm implementations for beginners. *ACM Transactions on Quantum Computing*, 3(4), July 2022.

[7] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2013.

[8] P. D. Nation, H. Kang, N. Sundaresan, and J. M. Gambetta. Scalable mitigation of measurement errors on quantum computers. *PRX Quantum*, 2(4), Nov. 2021.

[9] E. Novais and H. U. Baranger. Decoherence by correlated noise and quantum error correction. *Physical Review Letters*, 97(4), July 2006.

[10] J. Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, Aug. 2018.

[11] O. Prykhodko, S. Johansson, P.-C. Kotsias, E. Bjerrum, O. Engkvist, and H. Chen. A de novo molecular generation method using latent vector based generative adversarial network, 2019.

[12] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, and N. Killoran. Evaluating analytic gradients on quantum hardware. *Physical Review A*, 99(3), Mar. 2019.

[13] M. Schuld and F. Petruccione. *Machine Learning with Quantum Computers*. Springer Nature, 10 2021.

[14] D. Silver, T. Patel, W. Cutler, A. Ranjan, H. Gandhi, and D. Tiwari. Mosaiq: Quantum generative adversarial networks for image generation on nisq computers, 2023.

[15] A. Vieloszynski, S. Cherkaoui, O. Ahmad, J.-F. Laprade, O. Nahman-Lévesque, A. Aaraba, and S. Wang. Latentqgan: A hybrid qgan with classical convolutional autoencoder, 2024.

[16] Wikipedia. Autoencoder. https://en.wikipedia.org/wiki/autoencoder. (09.02.2025).

7. Appendix

The project also included implementing the LatentQGAN model proposed in the studied paper. The implementation is written in Python and uses the IBM open-source SDK called *Qiskit*. Below is a simple demo example demonstrating how the trained model can generate new data and evaluate it using the discriminator. A possible result is shown in Figure 9. The source code is publicly available in the Git repository at `https://github.com/Pikerozzo/qml_latentqgan`.

```python
import QGenerator
import Discriminator
import AutoEncoder

# create q-GAN (or import pre-trained one)
netQG, netD = load_qgan()

# create autoencoder (or import pre-trained one)
autoencoder = load_autoencoder()

# sample random noise for generating 2 images
noise_shape = (2,netQG.n_circuits,netQG.n_qubits)
noise = torch.rand(noise_shape)

# perform forward pass of q-generator
result = netQG(noise)
```

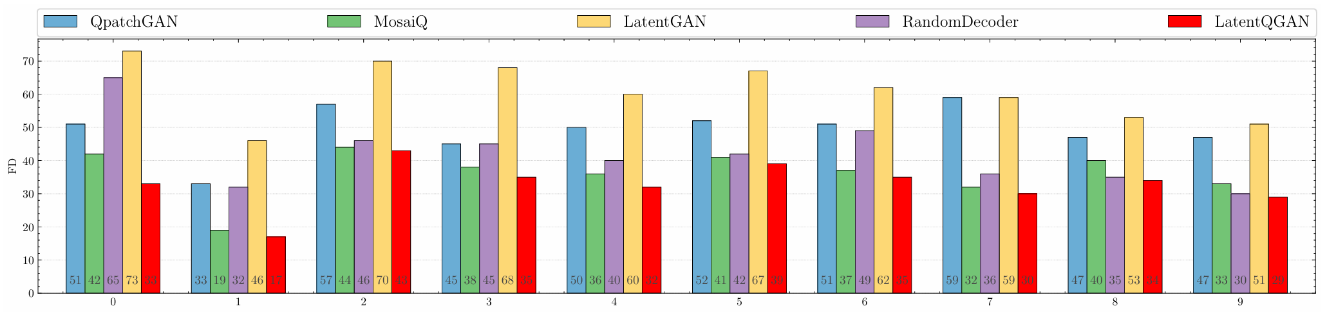

Figure 6. FID comparison of results from tested models, with LatentQGAN on simulators.

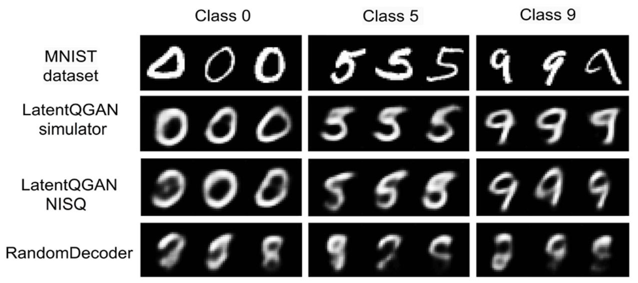

Figure 7. Visual comparison of results from LatentQGAN on simulator and quantum computer, and from RandomDecoder.

	Class 0	***Class 5***	***Class 9***
Quantum simulator	33	39	29
Quantum computer	38	43	34

Figure 8. FID comparison on simulator vs. quantum computer. Results on the quantum computer are obtained from a single run using identical noise vectors and initial parameters as those used in simulator training.

```

# evaluate results with discriminator
disc_output = netD(result)

# map to pixel-space using the decoder
generated_imgs = autoencoder.decode(result)

# show resulting images
```

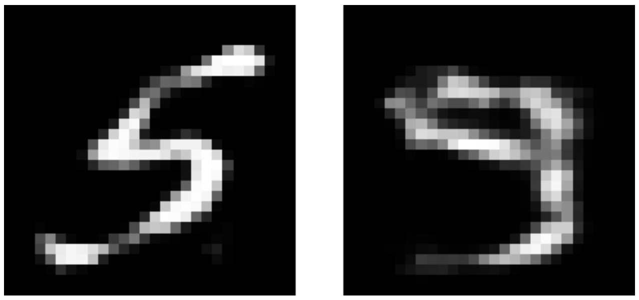

Figure 9. Demo result.