Matrix Product State Based Quantum Generative Adversarial Networks for Financial Time Series Modelling

van Drooge, Lucas

Matrix Product State Based Quantum Generative Adversarial Networks for Financial Time Series Modelling

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Masters of Science
in
Physics

Author :	Lucas van Drooge
Student ID :	S2334356
Supervisor :	Jordi Tura
2^{nd} corrector :	Vedran Dunjko

Leiden, The Netherlands, November 21, 2025

Matrix Product State Based Quantum Generative Adversarial Networks for Financial Time Series Modelling

Lucas van Drooge

Huygens-Kamerlingh Onnes Laboratory, Leiden University
P.O. Box 9500, 2300 RA Leiden, The Netherlands

November 21, 2025

Abstract

This thesis develops a quantum-inspired framework for modelling financial time series by using Quantum Generative Adversarial Networks (QGANs) based on Matrix Product States (MPS). Traditional financial models often struggle to capture the complex, high-dimensional characteristics of market data, leading to shortcomings in predicting or simulating financial phenomena accurately. To overcome these limitations, we explore a simulated quantum-inspired framework that leverages the computational advantages of quantum systems. In our approach, MPS are used to represent quantum states efficiently while controlling entanglement through the so-called bond dimension. This controlled representation allows the QGAN to explore a high-dimensional state space with improved precision compared to classical GANs. We simulate these networks to assess their performance in reproducing the subtle statistical features inherent in financial time series. We show that tensor-network techniques can replicate key statistical features of financial time series, providing both a foundation for future experiments on quantum hardware and advancing quantum machine-learning methodologies.

Contents

Chapter 1

Introduction

Financial Time Series

As Michael Lewis aptly described, "The financial world is a jungle" [1] teeming with complexity, unpredictability, and hidden dynamics. A fundamental challenge of financial mathematics is modelling this blend of natural and social phenomena. Conversely, Christina Stead [2] put it this way: "Financiers are great mythomaniacs; their explanations and superstitions are those of primitive men; the world is a jungle to them," underscoring the ability of mathematics and models to structure this "jungle".

A good model should reproduce the characteristic properties of the system, enabling analysis and forecasting. A key tool for understanding financial markets is the analysis of financial time series. [3, 4]. Financial time series consist of repeated observations of the same variable over time, capturing data such as stock prices, market indices, and exchange rates. Recorded at constant intervals, these data points form the backbone of many economic models and forecasting [5]. Over the years, financial markets have become more complex as new participants have entered and a wider array of products has been traded. Consequently, financial time series now capture more intricate market behaviours. [6]. Financial markets are influenced by forces ranging from macroeconomic to micro-level trading activities, making the behaviour of financial time series largely unpredictable. Despite this unpredictability, they often exhibit key common properties.

These characteristics of financial time series are known as stylised facts. For example, volatility clustering, where large changes in price tend to be followed by further large changes, and small changes tend to be followed by small changes. Often, little to no linear autocorrelation is seen in returns, meaning past returns are generally not correlated with future returns. Assets in the financial market often exhibit price movements that are correlated with one another, forming communi-

ties or clusters that are not easily distinguished from each other through simple correlation measures. Detecting these communities can reveal insights into market structure and systemic risks. Additionally, financial time series deviate from the Gaussian distribution, exhibiting heavy tails and skewness, which challenge traditional econometric models and show an increase in the number of outliers in the market [7].

Predictive Modelling

In the twentieth century, economists and statisticians laid down the foundations of financial econometrics, aiming to grasp the complexities of market behaviour and economic cycles. For this, accurate modelling of financial time series is crucial for risk management [8], strategic trading, and regulatory policy [9], making it an important area of economic research. In light of the complexities of financial time series, analysts and researchers have continually sought more involved models to capture the intrinsic details of the financial data. It started with simpler models and gradually moved towards more complex systems as the understanding of financial markets and the technology became increasingly advanced.
In the mid-20th century, the introduction of autoregressive (AR) models, which specify that an output variable depends linearly on its own previous values and on a stochastic term [10], marked a significant advancement in economic forecasting [5]. Because of its simplicity and effectiveness in handling stable financial time series, the AR model became a standard tool in economic forecasting. These models, together with the Moving Average (MA) model, led to models known as ARMA (Autoregressive Moving Average) and autoregressive integrated moving average (ARIMA) models, which have a more complicated stochastic structure [4]. These models were better at handling the random fluctuations that are common in financial data, providing analysts with improved predictive accuracy. The introduction of Autoregressive Conditional Heteroskedasticity (ARCH) and Generalised ARCH (GARCH) models in the 1980s could account for changes in volatility over time, a common characteristic of financial markets. Engle's introduction of the (ARCH) model in 1982 and Bollerslev's (GARCH) model in 1986 allowed the variance of the current error term to be modelled as a function of the variances of previous periods' error terms [11, 12]. These models significantly improved the forecasting of volatile financial series, such as stock prices and foreign exchange rates. These models are linear in squares and cross products of the data. With the improvement of computational power, more advanced models such as stochastic volatility models and Dynamic Conditional Correlation (DCC) GARCH [13] have been introduced. These extended the capabilities of earlier methods by capturing both time-varying volatility and correlations between multiple assets. While no longer linear in their structure, many of these models remain computationally tractable

through univariate or two-step likelihood-based estimation procedures.
In recent decades, artificial intelligence (AI) has fundamentally transformed financial modelling. AI's integration into financial analysis signified a departure from traditional econometric methods to more data driven approaches. This transformation is driven by the capability of AI to analyse large amounts of data quickly and accurately, making it a very important tool in the high frequency, data-rich environment of modern financial markets [6].

The 2000s saw the rise of machine learning (ML) algorithms such as decision trees [14, 15], support vector machines [16], and ensemble methods like random forests. These tools enhanced predictive accuracy in tasks ranging from algorithmic trading to risk assessment. As big data technologies matured in the 2010s [17], deep learning techniques, particularly convolutional neural networks (CNNs) and recurrent neural networks (RNNs), gained prominence [18]. CNNs and RNNs are particularly adept at processing vast amounts of data quickly, making them ideal for finance, where large datasets and the ability to model over short time intervals can lead to more accurate predictions and better risk management. This capability to handle larger data sets effectively allows for capturing subtle patterns and anomalies that previous techniques might miss, providing a deeper understanding of the dynamics of financial markets [19, 20]. This led to a significant increase in research conducted into financial modelling (See Figure 1.1).

However, the increasing reliance on AI models has shifted the challenge from model design to data availability. High-quality financial datasets, which are essential for effective training, are often limited due to privacy concerns, proprietary restrictions, and noise. These limitations can impair model generalisability and increase the risk of overfitting, especially when training on small or unrepresentative samples. To address this, Synthetic Data Generation (SDG) has emerged as a promising solution. By using statistical methods, simulation models, or neural networks, SDG techniques aim to augment or replace real data with artificial samples that preserve the underlying statistical properties of financial time series. This not only mitigates privacy and access issues but also allows for more robust model training and validation under diverse conditions. A leading approach in SDG is the Generative Adversarial Network (GAN), which learns to replicate the distribution of a real dataset by generating new, statistically consistent samples. Despite their success, classical GANs face computational limits in capturing complex, high-dimensional distributions. This has spurred interest in quantum-enhanced variants.

Quantum Computing in Financial Modelling

Quantum computing offers a fundamentally new method for computational finance by exploiting quantum mechanical phenomena such as superposition, entangle-

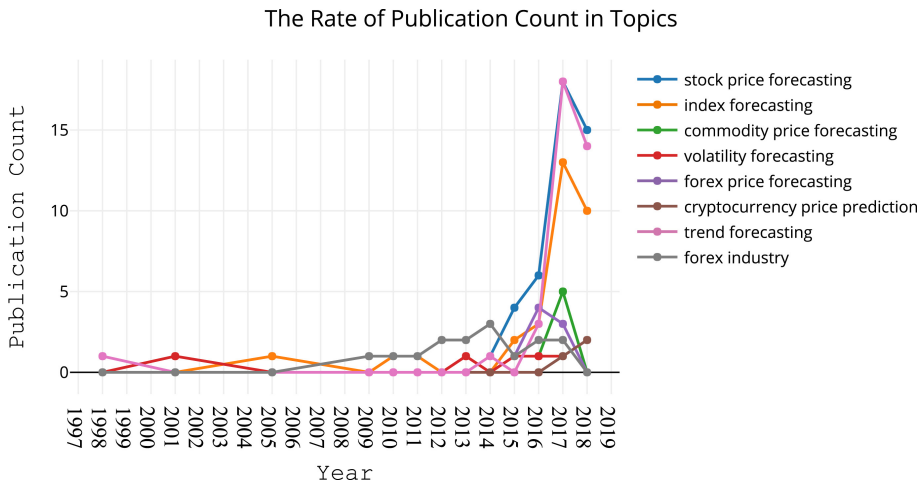


Figure 1.1: *The rate of publication counts for various implementation areas throughout the years. [6]*

ment, and interference. Unlike classical computing, which relies on bits as the basic units of data, 0s and 1s, quantum computing uses quantum bits, or qubits, which, due to the principle of superposition, can exist in combinations of 0 and 1 states. This, along with the principles of entanglement and quantum interference, enables quantum computers to process information in ways that can provide exponential speed-ups over classical computers for certain specialised tasks.

The theoretical foundations of quantum computing were laid in the early 1980s by physicist Richard Feynman, who proposed the idea that one quantum system could simulate another quantum system, something that a classical computer could not easily do [21]. These theoretical insights sparked experimental efforts, leading to rapid progress in building quantum hardware, with numerous breakthroughs achieved in recent years. For instance, there is an active race to build quantum processors with sufficiently low error rates to enable fault-tolerant operation, which is largely motivated by the promise of the technological disruption this machine is expected to bring [22–24]. While the broader field of econophysics has long applied analogies from statistical physics to model market phenomena [25, 26], the advent of quantum computing has more recently enabled genuine quantum algorithms for financial applications. Mapping the Black-Scholes-Merton PDE to a Schrödinger-type equation underpins quantum Monte-Carlo option-pricing methods [27–29]. Likewise, quantum linear-systems algorithms permit accelerated computation of portfolio risk metrics such as the covariance matrix [30, 31]. These developments mark a shift from purely conceptual mappings to practical quantum implementations in algorithmic trading, portfolio optimisation, and risk management.

While awaiting fault-tolerant quantum computers, researchers have developed methods to simulate quantum computers on classical hardware that can be used for research on the algorithm side and for testing potential use cases of quantum computers. One of the current challenges in simulating quantum systems with a high number of qubits on classical computers is the exponential growth of computational resources needed as the number of qubits increases. This exponential scaling arises because the Hilbert space dimension of a quantum system doubles with the addition of each qubit, leading to a dramatic increase in the memory and processing power needed to accurately represent and manipulate these states. As a result, classical simulations of quantum systems become impractical beyond a relatively small number of qubits. Matrix product states (MPS), however, can efficiently represent many quantum states with resources scaling linearly with the number of qubits and a bounded amount of entanglement.

In this thesis, I explore the use of MPS to simulate quantum states for financial time series modelling. Chapter 2 reviews the necessary background, including the stylised facts of financial time series and the theoretical principles of variational quantum circuits, introducing the concept of MPS and its implementation in our model. In Chapter 3, we define our research questions and explain their significance in the context of our work. Chapter 4 presents our results and addresses these research questions. Finally, Chapter 5 summarises our findings and discusses potential directions for future research.

Version of November 21, 2025– Created November 21, 2025 - 12:28

Chapter 2

Background

Following the motivation established in the Introduction, this chapter provides the core concepts used throughout the thesis. We begin with stylised facts of financial time series, develop the quantum formalism and the QGAN framework, and conclude with Matrix Product States (MPS), the tensor-network ansatz that enables scalable simulations for our experiments.

2.1 Financial Time Series

The financial time series category we will be looking at is known as indices. An index is a collection of a group of assets such as stocks or bonds that is used to track the performance of a specific sector of the broader market. The Standard and Poor's 500 (S&P500) is an example of one of these indices, see Figure 2.1. The S&P500 is a stock market index tracking the stock performance of 500 of the largest companies listed on stock exchanges in the United States. It is one of the most commonly followed equity indices and includes approximately 80% of the total market capitalisation of U.S. publicly traded companies. Market capitalisation (market cap) is the total value of a company's shares currently in circulation, calculated by multiplying the share price by the number of outstanding shares. It is used to gauge a company's size in the market. The S&P 500 index is a free-float weighted/capitalisation-weighted index. This means the price movement of the index can be and is currently dominated by a few companies that have a high market capitalisation. The S&P500 and financial time series in general exhibit distinct statistical properties known as stylised facts [32, 33]. Understanding these characteristics is crucial for developing accurate models and understanding market dynamics. Here, we discuss several well-documented stylised facts:

Figure 2.1: *S&P500 price history from 1950 to 2021.*

2.1.1 Non-Gaussianity

Traditional models in financial econometrics often rely on the assumption that financial time series, such as log returns, follow a normal (Gaussian) distribution. This assumption is convenient and also underpins many foundational results in finance, including the Efficient Market Hypothesis (EMH): if any predictable pattern existed, traders would exploit it until it disappeared due to the arbitrage that is to be had, driving the market back to randomness. However, real-world financial data frequently exhibit significant deviations from Gaussian distributions. Empirical studies consistently show that return distributions have heavy tails (higher likelihood of extreme outcomes), skewness (asymmetry), and volatility clustering. These features contradict the normal distribution assumption. Recognising non-Gaussianity is important when simulating or generating synthetic financial time data. Models that assume Gaussianity risk underestimating the probability of extreme events and thus misinterpret risk. In contrast, generative models that can model and replicate non-Gaussian features, including leptokurtic distributions (fat tails), offer a more realistic basis for risk analysis and trading strategies. We can see these fat tails in figure 2.2. It is empirically known that the probability distribution $P(r)$ consistently has a power-law decay in the tails.

$$P(r) \propto |r|^{-\alpha}. \tag{2.1}$$

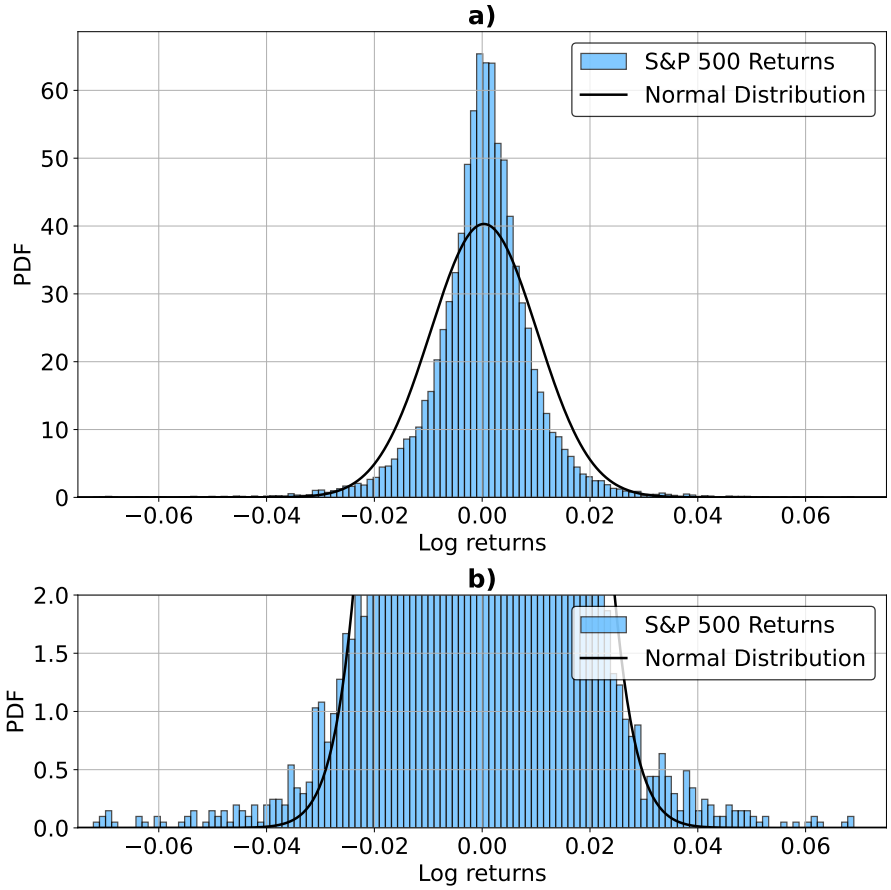


Figure 2.2: ***a)*** *S&P 500 daily return distribution compared with a Gaussian fit.* ***b)****Zoomed in S&P 500 daily return distribution on the "fat tails".*

where $P(r)$ is the probability density function of the return r, and α is a positive exponent that determines the thickness of the tails. Empirical studies often find values of α between 3 and 5 for various financial assets. A smaller α implies heavier tails, i.e., a higher likelihood of observing extreme returns. These non-Gaussian features are not just statistical curiosities, but they reflect underlying market dynamics such as herding behaviour and feedback loops. So accurately modelling them is critical for risk management and derivative pricing.

2.1.2 Absence of Linear Autocorrelation

Another key stylised fact of financial time series is the absence of significant linear autocorrelation in returns. In contrast to many natural systems where past values can help predict future ones, financial returns often appear almost uncorrelated over time, especially at daily or longer lags; a lag refers to the time interval between two successive data points. Autocorrelation quantifies the similarity between a time series and a lagged version of itself:

$$\rho_{\mathrm{id}}\left(\tau\right) = \mathrm{corr}\left(r_t, r_{t+\tau}\right) \tag{2.2}$$

where ρ_{id} is the autocorrelation for lag τ, r_t returns at time t. Here we assume weak stationarity, which means that the mean and variance are time invariant, and we take the sample average over t, which ensures that ρ depends only on the lag τ, not on any specific time index. $\mathrm{corr}(\mathrm{X}, \mathrm{Y})$ denotes the Pearson correlation and is given by:

$$\mathrm{corr}(\mathrm{X}, \mathrm{Y}) = \frac{\mathrm{cov}(\mathrm{X}, \mathrm{Y})}{\sigma_{\mathrm{X}}\sigma_{\mathrm{Y}}} = \frac{\langle(\mathrm{X} - \langle\mathrm{X}\rangle)(\mathrm{Y} - \langle\mathrm{Y}\rangle)\rangle}{\sigma_{\mathrm{X}}\sigma_{\mathrm{Y}}} \tag{2.3}$$

where σ_{X} and σ_{Y} are the standard deviations of X and Y.

By analogy, daily rainfall often exhibits positive autocorrelation. However, in financial markets, this predictive power vanishes. This is because of the Efficient Market Hypothesis we discussed previously. Nonetheless, small autocorrelations can appear over very short time scales due to market effects. For instance, during periods of market crisis or panic, negative returns cluster together, leading to short occurrences of autocorrelation. But overall, the absence of linear autocorrelation is a characteristic of a liquid and competitive financial market. The absence of linear dependence does not result in total randomness. Some non-linear patterns and volatility clustering still remain, as we will now see.

2.1.3 Volatility Clustering

Although financial returns themselves exhibit little to no linear autocorrelation, this does not mean that financial markets have no temporal structure. One of the

most prominent and well-documented forms of temporal dependence is volatility clustering [34, 35]. This refers to the empirical observation that large magnitude returns tend to be followed by further large magnitude returns, while periods of low volatility tend to persist. In other words, significant price swings cluster in time, with turbulent periods followed by more turbulence and calm periods followed by calm periods. A natural way to quantify this is through the autocorrelation of absolute or squared returns, such as $|r_t|$, rather than the returns themselves. These quantities capture the magnitude of fluctuations and show a strong, positive autocorrelation that decays slowly over time:

$$corr(|r_t|, |r_{t+\tau}|) > 0 \tag{2.4}$$

for τ ranging from a few minutes to several weeks.

2.1.4 The Leverage Effect

Another key temporal feature of financial time series is the leverage effect, which describes a negative correlation between past returns and future volatility. In practical terms, this means that when asset prices drop, volatility tends to increase, a phenomenon particularly evident in equity markets (stock markets). This effect is often attributed to a feedback mechanism: a fall in a company's stock price increases its debt-to-equity ratio, also known as financial leverage, which in turn raises the perceived risk and expected future volatility. The leverage effect can be quantified through the correlation between past returns and the square of future returns, expressed as:

$$L(k) = corr\{(x_{t+k})^2, x_t\}. \tag{2.5}$$

where x_t is the return at time t and x_{t+k}^2 is the future variance at lag k. We see that $L(k)$ often takes negative values that decay towards zero as k increases, confirming that negative returns are typically followed by higher volatility. This effect is asymmetric. Price drops tend to trigger a much stronger increase in volatility than the decrease caused by similarly sized price gains. The leverage effect is also market dependent. For example, a clear negative correlation is observed in indices like the SP500 and DAX, whereas in some emerging markets like China a positive correlation known as the anti-leverage effect has been reported. [36]
As with the other stylised facts, this effect disappears if the return series is randomly shuffled, underscoring its reliance on the temporal ordering of observations.

Having explored the intrinsic nature of financial time series and their underlying statistical properties in Section 2.1, we can see why modelling financial time series may not be as straightforward as we thought. This motivates the search for an expressive and flexible modelling technique. In the following section, we lay

the groundwork for one such approach by introducing core principles of quantum theory, ultimately setting the stage for the use of Quantum Generative Adversarial Networks (QGANs) in financial modelling.

2.2 Foundations of Quantum Theory

We first introduce the mathematical notation and fundamental concepts of quantum theory needed for the quantum inspired method used in this thesis. Classical computers make use of so-called bits. Bits are the basic units of information and can be represented as being either a 0 or a 1. What is special about the information units quantum computers use, qubits, is that they can be in the state 0 or a 1, but also, more generally, a complex linear combination of the two.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{2.6}$$

By the Born rule, measuring the state $|\psi\rangle$ yields $|0\rangle$ with probability $|\alpha|^2$ and $|1\rangle$ with probability $|\beta|^2$. Since there are only two outcomes, we must satisfy the normalisation condition $|\alpha|^2 + |\beta|^2 = 1$, ensuring that the total probability is unity [37]. In equation 2.6 we also introduced the "bra-ket" notation, which is widely used in quantum theory and quantum information. [38]. The bra-ket notation is used to denote quantum states. A ket is of the form $|v\rangle$. Mathematically, it denotes a vector, $\boldsymbol{v}$, in a (complex) vector space V (Hilbert Space), and physically, it represents a state of a specific quantum system. A bra is of the form $\langle f|$. Mathematically, it denotes a linear form $f : V \to \mathbb{C}$, which is a linear map that maps each vector in V to a complex number. In this thesis, we work exclusively in finite dimensional Hilbert spaces. Thus, every bra $\langle f|$, which is just the conjugate transpose of a ket $|f\rangle$, when applied to any ket $|v\rangle$, returns the complex number $\langle f \mid v\rangle \in \mathbb{C}$.

In quantum algorithms, we want to manipulate qubits much like we would manipulate bits on a classical circuit, for instance, with an AND gate. A quantum logic gate (or simply quantum gate) is a basic quantum gate operating on a certain number of qubits. Quantum gates are unitary operators and are described as unitary matrices relative to some orthonormal basis. A unitary matrix is a square matrix U with complex entries whose inverse is given by its own conjugate transpose. In quantum computing, unitary matrices play a central role because the evolution of a quantum state $|\psi\rangle$ is given by a unitary operator U. When we apply U to $|\psi\rangle$, it produces another valid quantum state $U|\psi\rangle$. Because U is a unitary transformation, it preserves the norm of vectors. In quantum mechanics, the norm of a state vector is related to its total probability (which must remain 1). Thus, unitarity ensures probabilities remain consistent under quantum evolution. Thus, the probabilistic interpretation of quantum states is preserved. This

is why only unitary transformations are valid gates in quantum computing. Every quantum gate is represented by a unitary matrix with respect to some orthonormal basis. In most quantum-computing contexts, that basis is the computational basis, meaning that for a d level system (a qubit when $d = 2$, or a qudit when $d > 2$) we take the reference states $|0\rangle, |1\rangle, \ldots, |d-1\rangle$ as our basis vectors. A gate that acts on n qubits (a register) is represented by a $2^n \times 2^n$ unitary matrix, and the set of all such gates with the group operation of matrix multiplication is the unitary group $U(2^n)$. Some notable examples are the Pauli gates and the CNOT gate. The Pauli matrices (X, Y, Z) or $(\sigma_x, \sigma_y, \sigma_z)$ act on a single qubit. The Pauli X, Y and Z equate, respectively, to a rotation around the x, y and z axes of the Bloch sphere by π radians. The Bloch sphere is a geometric representation of a qubit's state, where any pure state is mapped to a point on the surface of a unit sphere in three-dimensional space. In this representation, the north and south poles typically correspond to the basis states $|0\rangle$ and $|1\rangle$, respectively.

The Pauli- X gate is the quantum equivalent of the NOT gate for classical computers when looking at the standard basis $|0\rangle, |1\rangle$. It is called a bit-flip as it maps $|0\rangle$ to $|1\rangle$ and $|1\rangle$ to $|0\rangle$. Similarly, the Pauli- Y maps $|0\rangle$ to $i|1\rangle$ and $|1\rangle$ to $-i|0\rangle$ with $i = \sqrt{-1}$. Pauli Z leaves the basis state $|0\rangle$ unchanged and maps $|1\rangle$ to $-|1\rangle$. This is why the Pauli Z is called a phase-flip. The corresponding matrices are:

$$X = \sigma_x = \text{NOT} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$
$$Y = \sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$
$$Z = \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

The square of a Pauli matrix is the identity matrix.

$$X^2 = Y^2 = Z^2 = I, \qquad XY = iZ,\ YZ = iX,\ ZX = iY, \qquad XYZ = iI.$$

Controlled gates act on 2 or more qubits, where one or more qubits act as a control for some operation. Like the other single-qubit Pauli operators, the Pauli-Y gate is an element $U(2)$ of the group of all 2×2 unitary matrices, which is defined as

$$U(2) = \{\, U \in \mathbb{C}^{2\times 2} \mid U^\dagger U = I \,\}, \tag{2.7}$$

so basically all 2 by 2 unitary matrices. However, since a global phase does not affect any measurement statistics and thus is not physically observable, we can equivalently restrict our attention to the special unitary group SU(2), which consists of unitaries with

$$SU(2) = \{\, U \in U(2) \mid \det(U) = 1 \,\}. \tag{2.8}$$

SU(2) is a double cover of SO(3), which means that there is a homomorphism in which each rotation in SO(3) corresponds to two opposite points in SU(2). Concretely, U and $-U$ in SU(2) map to the same physical rotation in SO(3). Particles with spin $\frac{1}{2}$ (electrons, qubits) transform according to 2-component spinors, not ordinary 3-vectors. These spinors live in the fundamental representation of SU(2). As we saw, SU(2) only includes unitary matrices with determinant 1, so it drops any overall phase factor $e^{i\phi}$. Since global phases have no effect on measurement outcomes. For example, the controlled-Z gate

$$\mathrm{CZ} = \mathrm{diag}(1,\, 1,\, 1,\, -1) \tag{2.9}$$

belongs to SU(4) (the multi-qubit extension of SU(2)) and implements a relative π-phase only when both qubits are $|1\rangle$. The controlled NOT gate (or CNOT or CX), which we use extensively, acts on 2 qubits and performs the NOT operation on the second qubit only when the first qubit is $|1\rangle$, and otherwise leaves it unchanged. With respect to the basis $|00\rangle, |01\rangle, |10\rangle, |11\rangle$, it is represented by the Hermitian unitary matrix:

$$\mathrm{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

A Hermitian matrix is a square matrix H with complex entries that equals its own conjugate transpose: $H = H^\dagger$ and by the spectral theorem can be diagonalized as $H = U\Lambda U^\dagger$. Where U is unitary, and Λ is a real diagonal matrix whose entries λ_i are the eigenvalues of H. In quantum computing, Hermitian matrices represent observables, quantities like energy or spin, whose eigenvalues are the only possible measurement outcomes. As we saw before, a quantum state $|\psi\rangle$ is a unit-norm vector in a system's finite-dimensional Hilbert space. Measurements are described by a set of Positive Operator-Valued Measure elements $\{\hat{M}_m\}$. An operator on a finite-dimensional Hilbert space is positive semidefinite:

$$\langle\phi|M|\phi\rangle \geq 0 \text{ for all } |\phi\rangle. \tag{2.10}$$

Or in spectral decomposition

$$M = \sum_i \lambda_i |u_i\rangle\langle u_i|, \tag{2.11}$$

where every eigenvalue λ_i is non-negative, so no outcome probability can ever be negative. The elements $\{\hat{M}_m\}$ also satisfy

$$\sum_m M_m = I. \tag{2.12}$$

Which ensures that the probabilities of all possible outcomes sum to one when following the Born rule [39]:

$$\sum_m p(m) = \sum_m \langle\psi|M_m|\psi\rangle = \langle\psi|I|\psi\rangle = 1 \tag{2.13}$$

As we will be measuring Pauli's on our circuit, each M_m is a projector P_m onto an eigenspace. These are the special case of a projective measurement. For projective measurements, each M_m is an orthogonal projector, whose rank determines the degeneracy of the corresponding eigenvalue. So, if an eigenvalue is non-degenerate, its eigenspace is one-dimensional. The corresponding projector P has rank 1. If an eigenvalue is degenerate with multiplicity d, then its eigenspace is d-dimensional. This all ensures that outcomes are real (from Hermiticity), probabilities are non-negative and sum to one, and that all physically observable information resides in the relative phases and eigenvalues of the operators, not in any unobservable global phase.

When measuring a quantum state, the quantum state collapses thereafter. It collapses to

$$\frac{M_m|\psi\rangle}{\sqrt{p(m)}} \tag{2.14}$$

which lies in the eigenspace associated with M_m. You only ever observe one of the eigenvalues, and the system is left in the corresponding eigenspace of that eigenvalue. State collapse or wave function collapse is the cornerstone of the measurement postulate in quantum mechanics. It is the idea that while quantum systems can exist in a combination of states (superposition), the act of measuring a particular property forces the system to become one outcome, represented by one of the eigenstates of the measurement operator.

Building on this theoretical framework, Section 2.3 transitions into the practical application of these ideas by exploring Quantum Computational Techniques. We will introduce Quantum Generative Adversarial Networks, which we use as the practical application of quantum mechanics in financial modelling.

2.3 Quantum Computational Techniques

Quantum computing, with its potential to accelerate certain computational tasks that classical computers struggle with, has recently sparked a lot of interest. Quantum computers leverage superposition and entanglement to solve certain problems with lower complexity than the best known classical algorithms, offering potential polynomial or even exponential runtime advantages. For integer factorisation, for

example, the fastest classical algorithms run in sub-exponential but still super-polynomial time. For instance, the general number field sieve [40] factors a length n bit integer N in time

$$T(N) \;=\; \exp\Big(\big((\tfrac{64}{9})^{1/3}+o(1)\big)\,(\log N)^{1/3}\,(\log\log N)^{2/3}\Big). \tag{2.15}$$

In contrast, Shor's quantum algorithm runs in polynomial time in n [41]. This has a great effect on areas such as cryptography, particularly in decrypting data encrypted with cryptosystems (such as RSA), which rely on the difficulty of this task.
Quantum computing might also show a transformative approach to financial modelling, leveraging the unique aspects of quantum theory to enhance the generation and analysis of financial data. The built-in probabilistic nature of quantum mechanics identifies itself with the stochastic elements of financial markets. Unlike classical systems that require a distinct variable for each possible outcome, a quantum register can represent an entire probability distribution in superposition with only a handful of qubits, enabling far more compact handling of complex uncertainties. This can lead to more accurate synthetic data for applications like algorithmic trading and portfolio optimisation [42] where neural networks are trained on this synthetic data. An example of such a quantum algorithm is the Quantum Generative Adversarial Networks (QGANs), which exploit quantum superposition, entanglement, and interference to model complex probabilistic systems more efficiently and accurately than classical computing methods.

2.3.1 QGAN

The QGAN is inspired by the classical version; the Generative Adversarial Network (GAN) [43], which consists of two neural networksâthe generator and the discriminatorâthat engage in a zero-sum game. The generator's role is to create data that is indistinguishable from real data, while the discriminator's role is to detect whether the data it receives is real (the original real dataset) or fake (created by the generator). The generator's input is a random noise signal, and it then transforms this signal into data instances that resemble the real data as closely as possible. This transformation is guided by back propagation signals derived from the discriminator's assessments, effectively teaching the generator to improve its data generation capabilities. On the other side, the discriminator evaluates the authenticity of each data instance it receives, classifying it as either real or fake. This model is trained on a dataset of genuine data to establish a baseline of authenticity, which it uses to benchmark the synthetic outputs of the generator (see Figure 7.1).

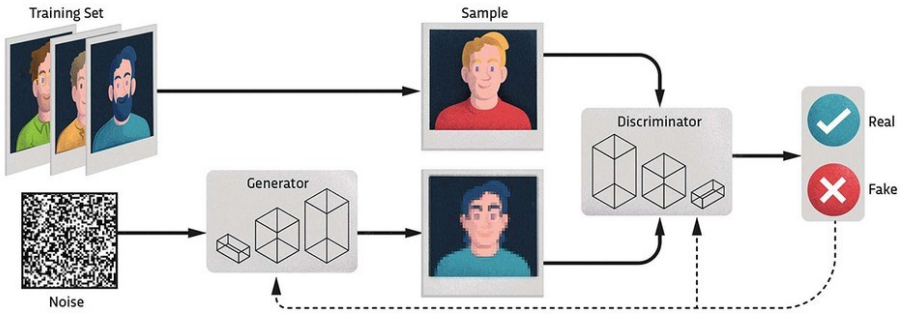


Figure 2.3: *GAN architecture showing a generator creating an artificial image from noise and the discriminator trying to figure out if it is a real image or not.[44]*

The QGAN's training objective can be formulated as the following optimisation problem:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}\Big[\log D(x)\Big] + \mathbb{E}_{z\sim p_z(z)}\Big[\log\Big(1-D(G(z))\Big)\Big]. \tag{2.16}$$

where $\min_G \max_D V(D,G)$ represents the min-max optimisation problem where the generator (G) aims to minimise the objective function against an adversary, the discriminator (D), which tries to maximise it.

$D(x)$ is the discriminator's estimate that a real data instance x is genuine, outputting a probability between 0 and 1 where $x \sim p_{\text{data}}(x)$ denotes that x is sampled from the real data distribution p_{data}. $G(z)$ is the generator's output when given a noise variable z. The generator maps the latent vector z (sampled from a noise distribution, typically Gaussian or uniform) $z \sim p_z(z)$ to the data space to produce synthetic data that mimics the real distribution.

$\mathbb{E}_{x\sim p_{\text{data}}(x)}\Big[\log D(x)\Big]$ is the expected value of the logarithm of the discriminator's predictions on real data x. The logarithm function amplifies the effect of errors when $D(x)$ is far from 1. The discriminator's goal is to maximise this value, effectively getting better at identifying real data. $\mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]$ This is the expected value of the logarithm of one minus the discriminator's predictions on generated data $G(z)$. Here, the generator's goal is to minimise this term by fooling the discriminator into believing that $G(z)$ is real, making $D(G(z))$ close to 1, thereby making $\log(1-D(G(z)))$ go to $-\infty$.

In a QGAN setup, a quantum circuit acts as either the generator and/or the discriminator. Here, the generator and/or discriminator is a parameterised quantum circuit: we adjust its gates so that when we measure its output qubits, we obtain classical samples whose distribution matches the real data. The discrimi-

nator then takes those classical samples along with the real ones and learns to tell them apart. In this way, although the end result is purely classical data, we train the quantum circuit to encode the data's probability law into its measurement statistics. Unlike approaches such as Lloyd & Weedbrook's quantum state GAN, where the goal is to reproduce an unknown quantum state, our QGAN only outputs classical samples [45]. The novelty is that a relatively small quantum circuit can flexibly store and generate complex classical distributions by exploiting its high dimensional Hilbert space. Our QGANs use variational quantum circuits (VQC). Variational or parametrised quantum circuits $U(\theta)$ are quantum algorithms that depend on free parameters θ. These parameters can then be optimised by classical post processing during the training. These circuits allow for adjustments in their parameters through an optimisation process that aims to minimise a cost function. The classical discriminator consists of a classical neural network. It processes the data samples and labels them either as being real or generated. Notably, the hyperparameters of the networks, i.e., the number of nodes and layers, need to be carefully chosen to ensure that the discriminator does not overpower the generator and vice versa. In the original GAN paper, Goodfellow et al. [46] show that if the discriminator becomes too optimal, its loss saturates and provides vanishing gradients to the generator, which prevents learning. Later work on the Wasserstein GAN by Arjovsky et al. [47] introduced a new critic loss that helps maintain useful gradients even when the discriminator is strong, directly addressing this bottleneck. For more information, read Appendix 7.1.1. But in short, the Wasserstein distance is given by

$$W_1\left(p_{\text{real}}, p_{\text{gen}}\right) = \inf_{\gamma \in \Pi\left(p_{\text{real}}, p_{\text{gen}}\right)} \mathbb{E}_{(x,y)\sim\gamma}[\|x-y\|], \tag{2.17}$$

where $\Pi\left(p_{\text{real}}, p_{\text{gen}}\right)$ is the set of joint probability distributions on $R \times R$ with the same marginals as our empirical distribution P, and Q and (x, y) are the samples from this distribution [32]. This formula might seem intractable to calculate. However, in the particular case of the Wasserstein-1 distance for 1-dimensional samples, it is equal to the area between the two marginal cumulative distribution functions given by

$$W_1\left(p_{\text{real}}, p_{\text{gen}}\right) = \int_{-\infty}^{\infty} |F_{p_{real}}(x) - F_{p_{gen}}(x)| dx \tag{2.18}$$

with $F_{p_{real}}(x)$ and $F_{p_{real}}(x)$ the marginal cumulative distribution functions of P_{real} and P_{gen} respectively. [48] We therefore use it as our primary GAN loss, which we shall refer to throughout as the "Wasserstein loss". In all subsequent experiments, the Wasserstein loss was adopted as the primary training objective for the QGAN, unless explicitly stated otherwise. See Equation 2.16.

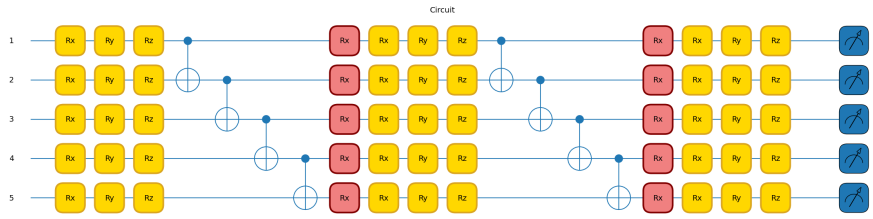


Figure 2.4: *Graphical representation of the quantum circuit used in our research.*

Quantum circuits used in QGANs incorporate building blocks such as parameterised rotations and controlled gates to create entanglement and manage superposition, further enhancing their ability to represent and manipulate high-dimensional data effectively. [49] We use a combination of these gates to obtain the quantum circuit we use in our research. See Figure 2.4 for a graphical representation of the quantum circuit we use. The example circuit shows a two-layer, 5 qubit version of our quantum circuit. The yellow gates represent the parameterised gates that are trained by the QGAN. The red gates represent the noise-uploading gates. The blue two-qubit gates represent the CNOT gates used for entanglement between the qubits. The final blue squares represent the measurement on the individual qubits. Our circuit uses noise-reuploading with trainable scaling parameters. Data-reuploading is the process of including multiple encoding layers in our circuit, usually sandwiched between rotation and entangling layers. It has been shown that this approach can increase the expressivity of PQCs [50]. One layer consists of 3 parametrised gates consisting of x, y, and z axis rotations of the qubit. Then, there are entanglement gates between neighbouring qubits. Then finally, there are the noise reuploading gates in x rotations. At the end of the whole quantum circuit, another round of parameterised gates is applied, and then the measurements are done. After executing the circuit, we measure the expectation values of a predefined set of Pauli operators, which form a classical output vector. In simulation, these expectation values are computed directly from the final state. On real quantum hardware, they must instead be estimated by repeated measurements (sampling) of the circuit outcomes to approximate the true expectation values. The output represents a time series data of $2N$ time steps, where N is the number of qubits in the circuit.

Rather than a Quantum Circuit Born Machine where you sample bit-strings directly from $|\psi\rangle\langle\psi|$, our generator implements the Expectation Value Sampler (EVS) framework [51]. Here one draws a classical latent vector $z \sim P_z$ and feeds it into the parameterized quantum circuit $U(z;\theta)$ on n qubits. Instead of measuring computational basis outcomes, we measure a fixed set of observables $\{M_i\}_{i=1}^{d}$ and

form our output:

$$x_i = \langle 0|U^\dagger(z;\theta)\, O_i\, U(z;\theta)|0\rangle, \quad i = 1,\ldots,d, \qquad x \in \mathbb{R}^d. \tag{2.19}$$

Unlike a Circuit Born Machine that samples bitstrings, the EVS framework outputs continuous expectation values directly.
As we explore more of the quantum toolbox, it is apparent that the efficiency of these quantum algorithms is tied to their ability to manage and manipulate large quantum systems constrained by current technological limits. Going into Section 2.4, we explore Matrix Product States (MPS), an important concept in the representation of quantum states that offers a solution to the scalability challenges we observed with high-dimensional quantum systems. This section will also detail how MPS are utilised to simulate large quantum systems under suitable conditions.

2.4 Matrix product states

As explained earlier, simulating quantum states faces exponential scaling as each additional qubit doubles the Hilbert space, dramatically increasing the memory and processing power required to represent and manipulate the state. To grasp the exponential scaling, let's look at storing a classical state and a quantum state [52]. Say we want to store a classical spin configuration of the Ising Hamiltonian on N spins:

$$H = J\sum_i \sigma_i\sigma_{i+1} \tag{2.20}$$

with $\sigma \in \{\pm 1\}$. So we have to store a string of N bits. In this classical case, the number of states that the system can take is 2^N and the memory to store a single state is N. So, the state will be easy to store, while it might be hard to find the ground state, the Hamiltonian's eigenstate with the lowest energy eigenvalue. Now let us look at the quantum state and consider the transverse field Ising Hamiltonian:

$$H = J\sum_i \sigma_i^z\sigma_{i+1}^z + \hbar\sum_i \sigma_i^x, \tag{2.21}$$

where σ^x and σ^z are the Paulis. So a state can be written as

$$|\psi\rangle = \sum_{\sigma_1,.,\sigma_n} \psi_{\sigma_1,\ldots,\sigma_n}\, |\sigma_1..\sigma_n\rangle \tag{2.22}$$

where $|\sigma_1..\sigma_n\rangle$ are basis states and each σ_i is equal to 0 or 1 for qubits (d=2). The coefficients $\psi_{\sigma_1,\ldots,\sigma_n}$ are complex numbers that define the amplitude of each basis state in the quantum state. In addition to exponentially many basis states, the system can also be in arbitrary superpositions of these states. This exponential

scaling makes it practically impossible to work on an exact solution for increasing system sizes. We want to use states that are efficient to optimise and store, which leads us to tensor networks and, in particular, Matrix Product States (MPS).

As we saw in equation 2.22, the full-state $\psi_{\sigma_1,\ldots,\sigma_n}$ has one complex amplitude for each of the 2^n basis configurations. In other words, its storage cost, and hence both memory and classical simulation time, grow exponentially with n. To understand MPS and why it is valuable, we first need to understand how singular value decomposition (SVD) is used for compression. Any matrix $M = \mathbb{C}^{l\times n}$ can be written as

$$M = U\Lambda V^{\dagger}, \tag{2.23}$$

where Λ is an $l \times n$ diagonal matrix with the singular values of M and the columns of U and the columns of V are called left-singular vectors and right-singular vectors of M respectively, with U being a $l \times l$ and V a $n \times n$. In the decomposition of Equation 2.23 , the squared singular values σ_i^2 sum to $||M||_F^2$, the total "energy" of the matrix. Thus, the largest singular values and their associated left and right singular vectors identify the directions along which M has the most variance ("information"). Truncating to the top k singular values yields the best rank k approximation in Frobenius norm, preserving most of the original matrix's structure. The Frobenius norm of an $m \times n$ matrix $M = [M_{ij}]$ is defined as the square root of the sum of the squares of all its entries:

$$\|M\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |M_{ij}|^2} = \sqrt{\sum_k \sigma_k^2}. \tag{2.24}$$

Hence, we can disregard the small singular values and corresponding singular vectors to compress the original matrix. Now we can apply this SVD to quantum states.

There are three ways to represent wave function as an MPS: left-canonical decomposition, right-canonical decomposition, and mixed-canonical decomposition [53]. We will show the left-canonical decomposition. The full state $\psi_{\sigma_1,\ldots,\sigma_N}$ is an Nth order tensor, a $d \times d \times \cdots \times d$ array with d^N components. To begin the MPS decomposition, we first separate the leftmost index σ_1, which encodes the physical degrees of freedom at the first site.

$$\begin{aligned} \psi_{\sigma_1,(\sigma_2\ldots,\sigma_n)} &= \sum_{\alpha_1} U_{\sigma_1,\alpha_1} D_{\alpha_1,\alpha_1} V^{\dagger}_{\alpha_1,(\sigma_2\ldots,\sigma_n)} \\ = \sum_{\alpha_1} U_{\sigma_1,\alpha_1} \psi_{\alpha_1,(\sigma_2\ldots,\sigma_n)} &= \sum_{\alpha_1} A^{\sigma_1}_{\alpha_1} \psi_{\alpha_1,(\sigma_2\ldots,\sigma_n)} \end{aligned} \tag{2.25}$$

Here $A^{\sigma_1}_{\alpha_1}$ contains the complex amplitudes that map each physical state σ_1 of the first site to each virtual (bond) index α_1, encoding both the local state probabilities

and its entanglement with the remainder of the chain. We got it by decomposing the matrix U into d row vectors A^{σ_1}. The dimension of the matrices D is called the bond dimension (χ). So the state vector, equation 2.22, now looks like:

$$|\psi\rangle = \sum_{\{\sigma\}} \sum_{\alpha_1} A^{\sigma_1}_{\alpha_1} \psi_{\sigma_1,(\sigma_2\ldots,\sigma_n)} \, |\sigma_1..\sigma_n\rangle \tag{2.26}$$

Now we continue with the second site. We group σ_2 and α_1 and $\psi_{\alpha_1,(\sigma_2\ldots,\sigma_n)}$ as $\psi_{(\alpha_1,\sigma_2),(\sigma_3\ldots,\sigma_n)}$. With SVD:

$$\psi_{(\alpha_1,\sigma_2),(\sigma_3\ldots,\sigma_n)} = \sum_{\alpha_2} U_{(\alpha_1,\sigma_2),\alpha_2} D_{\alpha_2,\alpha_2} V^{\dagger}_{\alpha_2,(\sigma_3\ldots,\sigma_n)} = \sum_{\alpha_2} A^{\sigma_2}_{(\alpha_1,\alpha_2)} \psi_{\alpha_2,(\sigma_3\ldots,\sigma_n)}, \tag{2.27}$$

we get

$$|\psi\rangle = \sum_{\{\sigma\}} \sum_{(\alpha_1,\alpha_2)} A^{\sigma_1}_{\alpha_1} A^{\sigma_1}_{\alpha_1,\alpha_2} \psi_{\alpha_2,(\sigma_3\ldots,\sigma_n)} \, |\sigma_1..\sigma_n\rangle \, . \tag{2.28}$$

Continuing this, we get the full state represented as a product of matrices.

$$|\psi\rangle = \sum_{\{\sigma\}} \sum_{(\alpha_1,\ldots,\alpha_{N-1})} A^{\sigma_1}_{\alpha_1} A^{\sigma_2}_{\alpha_1,\alpha_2} \ldots A^{\sigma_{N-1}}_{\alpha_{N-2},\alpha_{N-1}} A^{\sigma_N}_{\alpha_{N-1}} \, |\sigma_1 \ldots \sigma_N\rangle \, . \tag{2.29}$$

When we want the exact decomposition, the maximal dimensions of the A matrices is $(1 \times d), \left(d \times d^2\right), \ldots, \left(d^{N/2-1} \times d^{N/2}\right), \left(d^{N/2} \times d^{N/2-1}\right), \ldots, \left(d^2 \times d\right), (d \times 1)$ going from the leftmost site to the rightmost site. We expect the result to be a scalar, and because of this configuration of row and column vectors, this results in a scalar. Now we did not gain anything yet in terms of memory usage when representing this exact decomposition.

By capping the bond dimension at a fixed value independent of the system size n we sacrifice being able to exactly represent every possible quantum state. But now our MPS uses only $\mathcal{O}\left(nd\chi^2\right)$ parameters with system size n, local dimension d and bond dimension χ. Whereas full-state needs $\mathcal{O}(d^n)$ amplitudes. Consequently, our memory usage now scales linearly with growing system sizes instead of exponentially. In essence, χ dictates the capacity of the MPS to capture the entanglement between different parts of the system. This trade-off between efficiency and accuracy is a central feature of the MPS approach.

We can also present this idea visually, see Figure 2.5 The horizontal connections between the tensors U are the matrix multiplications in the equation above. They are contractions over the so-called virtual indices. The physical indices refer to the indices that connect the tensors in the MPS to the actual physical system. Each tensor in the MPS is associated with a site in the physical system, and the physical index of each tensor corresponds to the possible states that the quantum

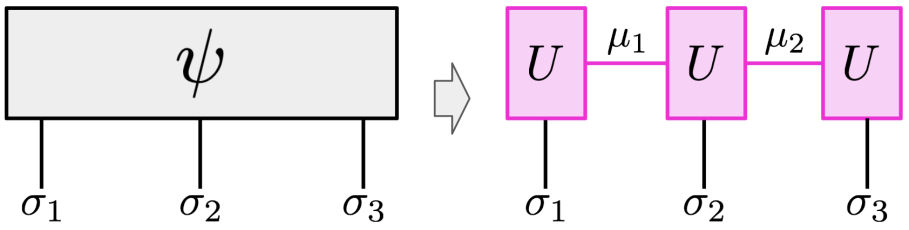


Figure 2.5: *Graphical representation of defining matrix product state with $\mu_{1,2}$ the virtual indices and σ_i the physical indices.*

system at that site can occupy. In the qubit case, this would be the 0 and 1. The vertical lines are the physical indices of the original state, showing how each tensor is linked to the state space of the physical system. So the MPS now has the following indices: $(virtual_{left}, physical, virtual_{right})$. The sizes of the virtual bonds exponentially grow from 2 to $\mathcal{O}(2^{n/2})$ until the middle of the chain. So the bond dimensions grow as

$$2, 4, 8, \ldots, 2^{\lfloor n/2 \rfloor}, \tag{2.30}$$

and then symmetrically decrease back to 2. To understand this, consider an MPS that models a linear chain of n qubits. Each qubit is linked to its neighbour by virtual bonds whose dimensions determine the capacity of the MPS to encode entanglement between these qubits. At the start and end of the chain, the bond dimensions are minimal. However, as we move towards the centre of the chain, the bond dimensions increase exponentially because the entanglement is now captured across an increasing number of qubits. This exponential growth in bond dimension continues until we reach the midpoint of the chain. Here the bond dimension peaks at the $\mathcal{O}(2^{n/2})$ point. This peak shows the maximum entanglement entropy that the MPS needs to capture across half of the system. So MPS are efficient for systems with low to moderate entanglement. Highly entangled states in large quantum systems require a big increase in computational resources because of this growth in bond dimensions. Because all the matrices U from the SVD are left-normalized.

$$\sum_{\sigma_i} A^{\sigma_i,\dagger} A^{\sigma_i} = \mathbb{I}. \tag{2.31}$$

Contracting a site of the MPS from the left is equal to the identity, and the norm of the MPS is also equal to the identity. Going from left to right is, as we mentioned, just a choice, and you can also go from right to left and obtain a right-canonical state by keeping the right-orthonormal $V^\dagger$ of the SVD. When computing an expectation value, we take some observable O and we set up the MPS so that left of the site we want to measure, this observable is left-orthonormal and to the right of this site is right-orthonormal resulting in the mixed-canonical form. Now

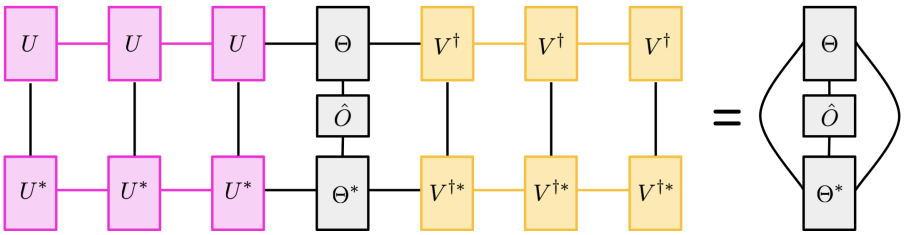


Figure 2.6: *Graphical representation of the contraction $<\psi|\hat{O}|\psi>$ for local expectation values reduces to contractions on just a single site, because all other contractions are just the identity.*

when calculating the expectation value of the observable reduces to contracting the MPS down to just one site, because all other contractions are just the identity. See figure 2.6.

Bond Dimension and Entanglement

MPS are a tensor-network ansatz that efficiently parametrises one-dimensional quantum states whose bipartite entanglement satisfies an area law. What is meant by this is states where the entanglement entropy across any cut is bounded. One can determine and quantify the entanglement between the two parties of a bipartite state. Separating a full system $|\psi\rangle$ and a sub-system ρ_{sub}, the von Neumann entanglement entropy is given by

$$S(\rho_{\text{sub}}) = -\text{tr}\left[\rho_{\text{sub}} \log\left(\rho_{\text{sub}}\right)\right]. \tag{2.32}$$

In our MPS, the singular values of the bonds encode the entanglement of bipartitions between all sites to the left of the bond and all sites to the right of the bond:

$$S(\rho_{1:i}) = S(\rho_{i+1:n}) = -\sum_{\mu_i=1}^{\chi} \Lambda_{\mu_i}^2 \log\left(\Lambda_{\mu_i}^2\right). \tag{2.33}$$

So, for a bond dimension, χ, the maximal entanglement entropy we can get is when an equal distribution of singular values:$\Lambda_i^2 \equiv 1/\chi$. So the entanglement entropy is bounded by

$$S(\rho_{1:i}) \leq \log(\chi) \tag{2.34}$$

(log base 2 for qubits).

Bond dimension analysis

Because the bond dimension directly controls the accuracy of the MPS approximation, we must select it to balance expressiveness and efficiency. We can inspect

this by comparing the full-state simulation and the MPS simulation with fidelity.

$$F(\rho, \sigma) = \mathrm{Tr}\left(\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}}\right)^2 \tag{2.35}$$

While the fidelity is symmetric and bounded between 0 and 1, it does not satisfy the triangle inequality and so is not a metric space in the strict sense [54, 55]. But it has a simple closed form, an operational interpretation, and is monotonic. Fidelity is closely related to the standard trace distance metric $D(\rho, \sigma) = \frac{1}{2}\|\rho - \sigma\|_{\mathrm{tr}}$ by the Fuchs-van de Graaf inequalities [56]

$$1 - \sqrt{F(\rho, \sigma)} \leq D(\rho, \sigma) \leq \sqrt{1 - F(\rho, \sigma)}. \tag{2.36}$$

This relationship helps us define that high fidelity implies a small trace distance. We use finite-size scaling, which is a standard method in tensor network simulations. The idea is to run the same simulation with an increasing bond dimension and check if it converges to the exact value.

Python implementation

At the current progress of quantum computing, actual quantum computers with the necessary scale and fidelity for broad application remain beyond reach as they are very expensive and do not support high qubit numbers. Consequently, we rely on simulations of quantum computing on classical systems to test and develop quantum algorithms. However, it's important to note that simulations may not capture all the nuances of a real quantum system, which could affect the reliability of experimental results. We leverage Quimb's [57] efficient (MPS) backend together with JAX's [58] "just-in-time" compilation and automatic differentiation to build and train the quantum GAN. QuimbMPS allows us to express parameterised circuits as tensor networks, automatically managing bond-dimensionality, compression, and contraction, while the *backend* = *jax* flag ensures that every gate application and expectation value measurement is traced by JAX. We then execute multi-step WGAN-GP training loops with GPU acceleration. We use the stochastic Adam optimiser to optimise our generator and critic networks. As this algorithm has an adaptive learning rate, it requires less tuning compared to methods like stochastic gradient descent. This work was performed using the compute resources from the Academic Leiden Interdisciplinary Cluster Environment (ALICE) provided by Leiden University. This in combination with the python package Ray allows for parallelised training of models [59]. The code is available in the Github repository: `Quantum-Finance`

2.4.1 Pre- and Postprocessing

Before training the QGAN, the financial time series require careful transformation to ensure stability and comparability across experiments. The following pre- and post-processing steps were applied.

Data normalization

All log returns were standardised to zero mean and unit variance:

$$r_t^{\text{norm}} = \frac{r_t - \mu_r}{\sigma_r}, \tag{2.37}$$

where r_t are the log returns, and μ_r, σ_r the mean and standard deviation. This helps with stability and ensures equal weighting across training samples.

Inverse Lambert-W transformation

To address the heavy tails present in financial returns, we applied an inverse Lambert-W transformation, which Gaussianises the distribution while retaining the option of recovering the original scale. This helps the generator model distributions with more stable gradients during training. After this step, a second normalisation was performed to re-standardise the transformed values. Clipping was applied to avoid extreme outliers introduced by the inverse mapping.

Rolling window segmentation

Since the QGAN requires a dataset of samples rather than a single time series, we segmented the data into overlapping windows of fixed length m, shifted with stride s. Each window forms an individual training sample, providing the model with temporal context.

Post-processing

After training, the generated sequences from the QGAN were mapped back to the original scale by inverting the normalisation and Lambert-W transformations.

Version of November 21, 2025– Created November 21, 2025 - 12:28

Chapter 3

Research Questions

In this chapter, we outline the central research questions that guide this thesis. The study of MPS within the Quantum Generative Adversarial Network offers an opportunity to explore and expand the exciting new world of quantum-inspired financial modelling. The specific focus on circuit architecture and training parameters is driven by the need to develop efficient and accurate quantum simulations that can be effectively applied in financial analysis.

The research questions are designed to investigate the relationship between MPS representation, circuit complexity, and the performance of quantum simulations in modelling complex financial systems. Each question targets specific aspects of QGAN deployment, from the technical details of quantum state simulation to the practical implications of model training and performance. By addressing these questions, our research aims to result in insights that are practically relevant to advancements in quantum technology applications in finance.

Research Question 1: How do bond dimension and circuit complexity (measured by the number of CNOT gates) affect the Wasserstein loss in the QGAN?

This question addresses the trade-off between model expressivity and efficiency. We expect that as the bond dimensions and generator circuit complexity increase, there will be a monotonic decrease in the Wasserstein loss, suggesting more accurate quantum simulations. We maintain the same critic architecture for an honest comparison. The circuit complexity is defined as the number of CNOT gates present in the circuit, as this affects the required bond dimension to accurately represent a state. The number of CNOT gates is tied to both the number of qubits

in the circuit and the number of layers. We keep the number of qubits constant, and vary the number of layers in the circuit, thus increasing the number of CNOT gates present in the circuit.
To investigate how bond dimension and circuit complexity affect the Wasserstein loss, we need to first examine what bond dimension is required to achieve our target fidelity in MPS state simulations compared to full-state simulations across various circuit configurations. Bond dimension in an MPS directly influences the model's ability to represent and process entanglement and correlations across the quantum system. Full-state quantum simulation directly models the quantum state using a vector in a Hilbert space, where each qubit added to the system doubles the size of the state vector. We test this by simulating different circuit depths with varying bond dimensions and measuring the fidelity of MPS simulations relative to full-state simulations. We expect to observe a clear trend where deeper circuits (more CNOT gate layers) require higher bond dimensions to achieve fidelity close to one (representing high agreement with full state simulations). This analysis will help us choose the correct bond dimension for various layers in answering the question of how layer depth affects the Wasserstein loss and the influence of the bond dimension on the Wasserstein loss.

Research Question 2: How quickly do QGANs with different circuit configurations reach a defined training plateau, and what are the implications for scenarios with time constraints?

We expect that more complex circuit configurations will require a larger number of epochs to reach a performance plateau. This analysis will help in understanding the trade-offs between circuit complexity and training time. By plotting the epoch against accuracy improvement, we expect to observe that simpler circuits exhibit a steeper initial increase in accuracy, achieving a plateau with fewer training epochs compared to their more complex counterparts.
There are practical implications in scenarios where QGANs must be deployed within strict time constraints, such as financial trading algorithms that need to be updated daily or real-time risk assessment models. The ability to train quickly becomes as important as achieving high long-term accuracy. We think that for applications with close time constraints, simpler models may be more viable despite potentially lower accuracy after longer training. This is because of the limited training time available; simpler models achieve higher accuracy than more complex models that have not yet converged. This could lead end-users to opt for

'suboptimal' models in terms of potential maximum accuracy, but 'optimal' in terms of practical deployment within the given constraints.

Research Question 3: How do composite loss functions, which combine the Wasserstein loss with penalties targeting stylized facts, affect QGAN performance?

This question investigates whether explicitly encouraging stylised facts improves the realism of synthetic financial series beyond distributional alignment alone. Previously, we only trained our QGAN model using the Wasserstein loss. While the Wasserstein loss excels at aligning overall distributions, it treats all discrepancies equally and can overlook the dynamic features that define real financial time series. Stylised facts such as volatility clustering, the leverage effect, and short-range linear autocorrelations are critical for tasks ranging from risk estimation to algorithmic strategy development. If a generative model reproduces the marginal distribution of returns but fails to capture these phenomena, downstream analyses can produce misleading results. By introducing composite loss functions that penalise deviations in these specific metrics, we can guide the QGAN to generate data that not only "looks right" in aggregate but also behaves like real markets. We do this by varying the weights and forms of these additional loss terms. This will also add to the roadmap, as specific applications of models may require certain market behaviours to be more prominent in the synthetic data.

Research Question 4: How does the number of qubits, which sets the maximum time resolution of the generated series, affect the QGAN's ability to reproduce stylized facts?

As the number of qubits is directly related to the number of time steps in the synthetic data, varying the number of qubits influences the resolution of the data. This resolution controls the granularity of the output: more qubits allow us to simulate finer temporal scales, such as minute-level versus five-minute data, where phenomena like rapid volatility clustering and short-range autocorrelations become pronounced.

Together, these four research questions structure the investigation. Addressing them provides insights into the interplay between circuit design, MPS represen-

tation, and training objectives, contributing to a broader understanding of the potential of quantum-inspired methods for financial modelling.

Version of November 21, 2025– Created November 21, 2025 - 12:28

Chapter 4

Results

This chapter presents the findings from our investigation into the performance of MPS-based Quantum Generative Adversarial Networks (QGANs) applied to financial time series modelling. Each section directly corresponds to one of the research questions outlined in Chapter 3. We systematically analyse the role of bond dimension, circuit complexity, training time, composite loss functions, and the number of qubits in overall accuracy in capturing stylised facts of financial time series. We primarily gauge accuracy by the Wasserstein loss (2.16), which served as the training objective throughout the experiments presented in Sections 4.1, 4.2, and 4.4. Only in Section 4.3 were composite losses introduced as a modification.

We also examine the stylised facts, which we qualitatively gauge with plots that we compare to the observed effects in the S&P 500.

RQ1: How do bond dimension and circuit complexity (measured by the number of CNOT gates) affect the Wasserstein loss in the QGAN?

Before we embark on answering the main research question, we first compare the computational scaling of MPS and full-state simulations. As this determines the practicality of training QGANs on larger circuits. Figure 4.1 reports wall-clock time at fixed hardware and a fixed circuit depth of $L = 14$. We can clearly see the linear growth of the simulation time for MPS simulations with a constant bond dimension. In contrast, the simulation time for the full-state simulation grows exponentially, limiting the capabilities of simulating systems with a high number of qubits due to hitting hardware limits.

To highlight the difference in scaling behaviour, we also present the runtime

on a logarithmic scale. See Figure 4.2.

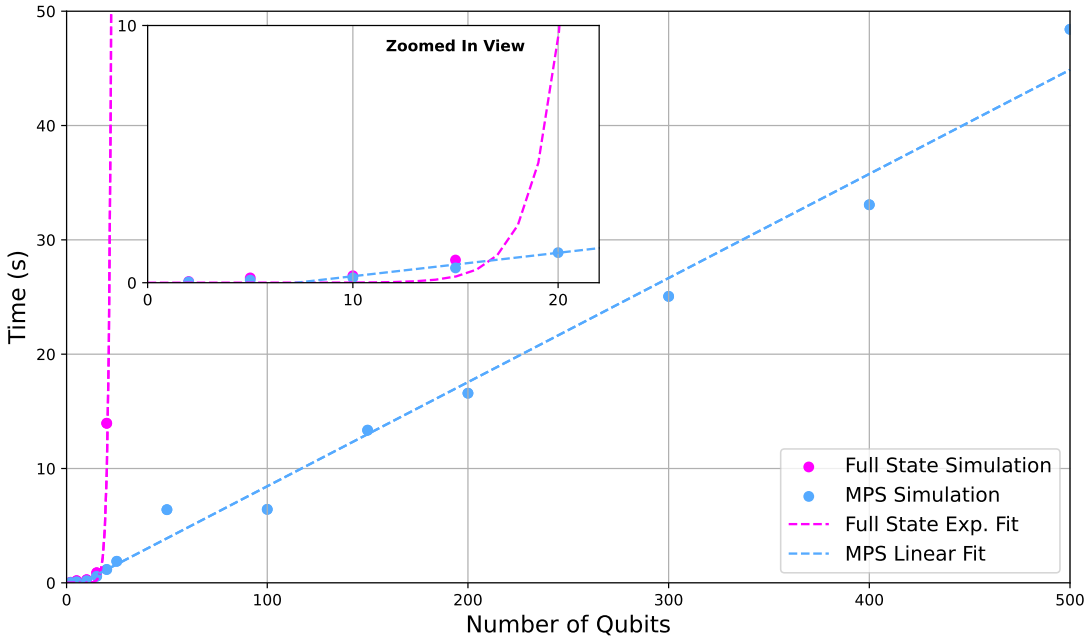


Figure 4.1: *Simulation time versus qubit count for full-state and MPS simulations at fixed depth $L = 14$. Inset: zoom of N up to 20 region showing that for small systems the full state simulation is faster. We can see a clear exponential growth of the time of the Full state simulation in the Zoomed in view. The Linear growth of time of the MPS simulation is evident from the image showing simulation times of up to 500 qubits.*

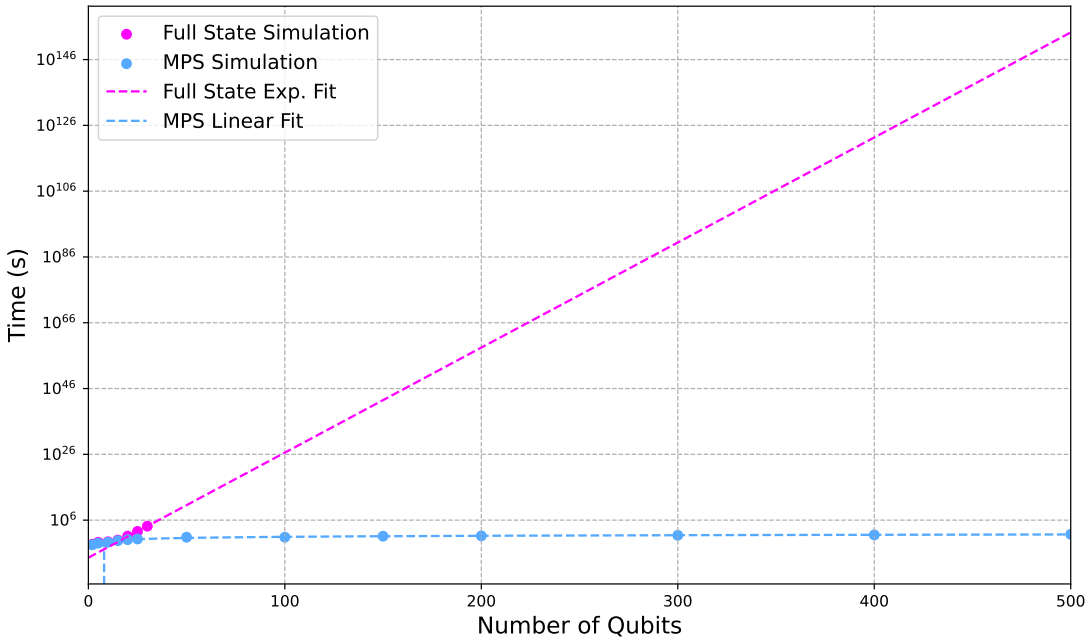


Figure 4.2: *A log-scale plot of the simulation time emphasises the exponential growth of full-state simulations with qubit number, compared to the linear scaling of MPS. For the full state simulation we see a clear linear relationship between the number of qubits and the log simulation time indicating an exponential growth of simulation time vs qubits. For the MPS simulation we see a horizontal line indicating a linear relationship between the simulation time and the number of qubits.*

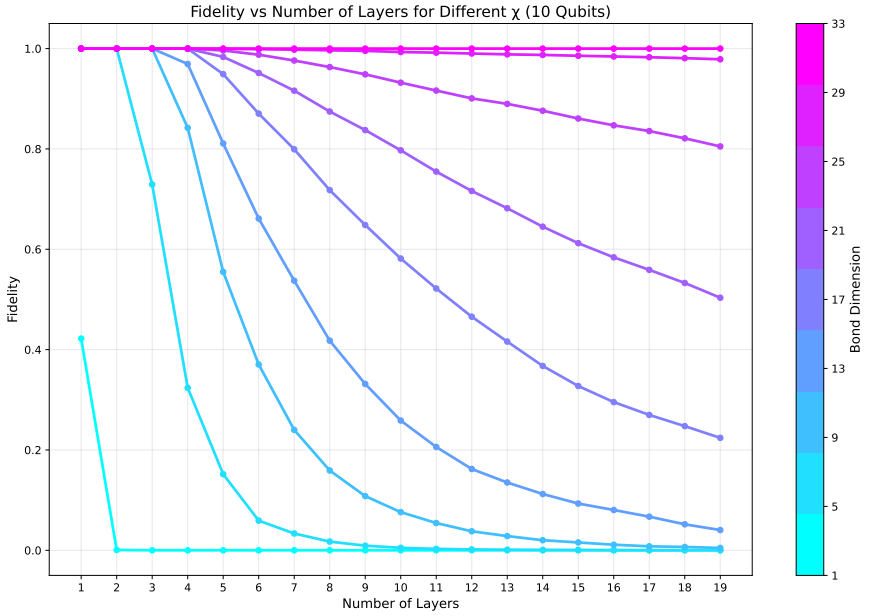
However, reduced computational cost is only valuable if the MPS approximation remains faithful to the full-state simulation. We therefore evaluated the fidelity of quantum states prepared by MPS simulations relative to the quantum states prepared by full-state simulations. χ (bond dimension) ranged from 1 to 32 for the 10 qubit circuit, as χ required for the full approximation of the full-state, which is $\chi_{max} = 2^{N/2} = 2^5 = 32$. For the 20 qubit circuit, we varied χ between 1 and 1024 as $\chi_{max} = 2^{N/2} = 2^{10} = 1024$. The circuit depth (number of layers) varied between 1 and 20.

The plot of the fidelities of our simulations, illustrated in Figure 4.3, offers valuable insights into the interplay between bond dimensions, circuit complexity, and fidelity in MPS simulations compared to the full-state simulation.
Fidelity monotonically decreases as the number of Variational Quantum Circuit (VQC) layers increases when the bond dimension is held constant. This trend reflects the inherent challenge of maintaining quantum state accuracy in deeper circuits. Each additional layer introduces more entangling operations, which, in the context of MPS simulations, require higher bond dimensions to accurately cap-

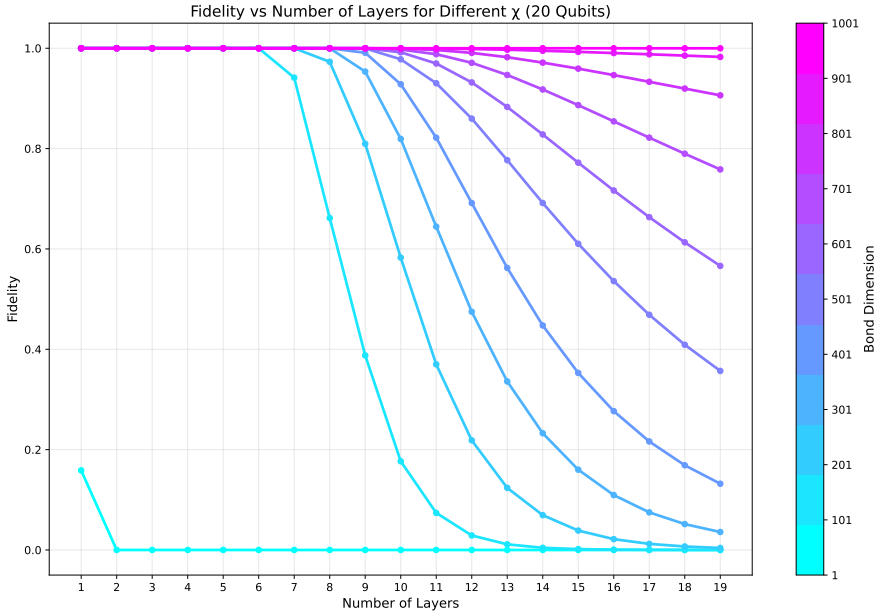
ture the growing entanglement. If the bond dimension is low, it results in a loss of representational accuracy.
Larger systems, such as those with 20 qubits, exhibit more rapid declines in fidelity compared to smaller systems with 10 qubits. This can be attributed to the exponential increase in the dimension of the Hilbert space with the number of qubits, which exponentially complicates the state representation and the simulation of these states. As quantum systems scale up, maintaining high fidelity (close to unity) also becomes increasingly more resource-intensive, as more memory is required for higher bond dimensions.

Now we will look at the practical effects of how the bond dimension and the number of VQC layers jointly influence the Wasserstein loss during QGAN training. As the bond dimension, χ, increases, the MPS representation becomes more expressive and can approximate a larger portion of the full state space. This directly improves fidelity since $F(D)$ is a non-decreasing function of the bond dimension D: for $D_1 < D_2$ we have $F(D_1) \leq F(D_2)$. Thus, increasing χ cannot worsen the accuracy of the simulation and typically leads to a better approximation of the true quantum state.

To prove this, we investigated how variations in circuit complexity influence the Wasserstein loss within our QGAN. The outcomes are visualised in a plot that encapsulates the relationship between circuit complexity and the performance of the quantum model. We trained the QGAN for various layers and bond dimensions and plotted the resulting Wasserstein loss in Figure 4.4. We also plotted the 2D representations of this 3D plot in Figure 4.5 We can identify the following trends: The plot shows a general rough decrease in Wasserstein loss with increases in the number of VQC layers. Each additional layer increases the number of trainable parameters and the depth of entangling operations, potentially allowing the model to capture more intricate temporal correlations in the financial data. However, this increased expressivity comes at the cost of a more complex optimization landscape and higher bond dimension requirements.
We can also see in Figure 4.5a, which shows the 2D plot of Wasserstein loss vs layer depth, that the Wasserstein loss generally decreases with increasing layer depth. The main outlier is the $\chi = 1$ line. The models with $\chi = 1$ are essentially in the product state; thus, the results of the training are merely a "lucky guess". We also see slightly higher values for $L = 18$ in some models. This is due to incomplete training. In Figure 4.5b, which shows the 2D plot of Wasserstein loss versus bond dimension, the Wasserstein loss decreases as the bond dimension increases. The trend stagnates with higher bond dimensions; at some point, the maximum fidelity is reached, and no further benefits can be observed from increasing the bond dimension. Again, we observe discrepancies at $\chi = 16$, where the simulations for $L = 18$ and $L = 5$ became corrupted. Similarly, the run with $\chi = 24$ at $L = 5$



(a) *10 Qubits*



(b) *20 Qubits*

Figure 4.3: *Fidelity of MPS simulation vs full-state simulation for various VQC layers and bond dimensions. The plots demonstrate the fidelity trends across different qubit counts (10 and 20 qubits) with the maximum bond dimensions for that particular qubit amount.*

was corrupted. These issues explain the spikes visible at those points. Again the training may not be complete for the more complex models.

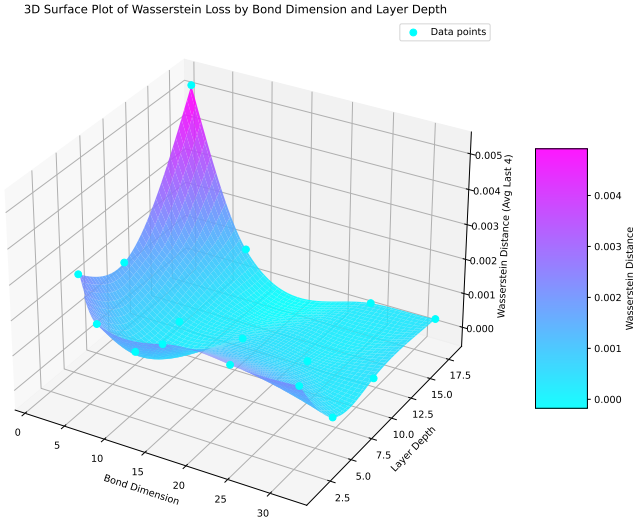


Figure 4.4: *3D visualization of the Wasserstein loss as a function of both bond dimension and the number of VQC layers. To better illustrate the behaviour of the loss function as a function of the bond dimension or layer depth see Figure 4.5.*

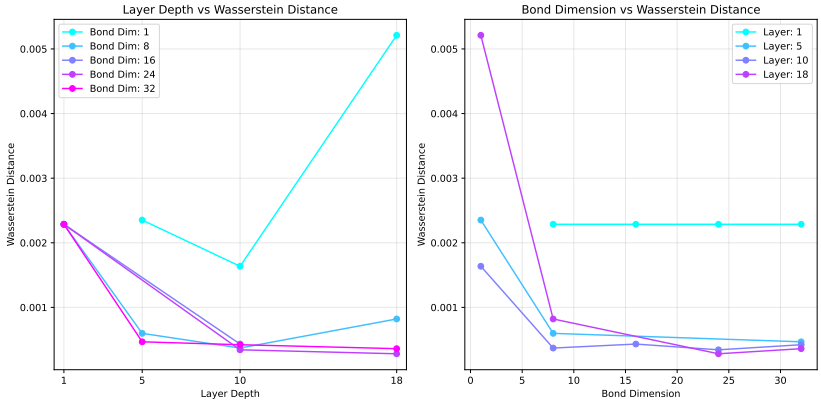


Figure 4.5: *2D plots of the Wasserstein loss as a function of both bond dimension (right) and the number of VQC layers (left). We see that when increasing the amount of layers generally resulted in a lower cost function. For bond dimension we see a average trend of lower cost function values for higher bond dimensions. Some discrepancies at bond dimension 16 are due to miss training. The Layer 1 line does not improve for a higher bond dimension as the minimum cost function value has already been reached for bond dimension 8.*

The final loss values averaged over the last 200 epochs are visible in Table 7.4.

Table 4.1: *Final loss values (average of last 200 epochs) for different configurations*

Layers	χ=1	χ=8	χ=16	χ=24	χ=32
1	—	0.002286	0.002286	0.002286	0.002286
5	0.002352	0.000598	—	—	0.000467
10	0.001636	0.000371	0.000432	0.000343	0.000422
18	0.005211	0.000821	—	0.000282	0.000361

We also show the impact of bond dimension and layer depth in tables 7.5 and 7.6 respectivly.

Table 4.2: *Impact of bond dimension on loss (averaged across all layers)*

χ	Avg Loss	Std Dev	Min	Max	Count
1	0.003067	0.001545	0.001636	0.005211	3
8	0.001019	0.000749	0.000371	0.002286	4
16	0.001359	0.000927	0.000432	0.002286	2
24	0.000971	0.000931	0.000282	0.002286	3
32	0.000884	0.000810	0.000361	0.002286	4

Table 4.3: *Impact of number of layers on loss (averaged across all bond dimensions)*

Layers	Avg Loss	Std Dev	Min	Max	Count
1	0.002286	0.000000	0.002286	0.002286	4
5	0.001139	0.000859	0.000467	0.002352	3
10	0.000641	0.000499	0.000343	0.001636	5
18	0.001669	0.002056	0.000282	0.005211	4

The 5 best performing models all had $L > 10$ and $\chi > 24$ except for one which had $\chi = 8$. The worst performing models had either $L = 1$ or $\chi = 1$. See Section 7.4 for more details. To inspect the real potentially added value of bond dimension and layer depth we also did the previous analysis secluding all $\chi = 1$ simulations as these were just product states. We also did not include $L = 1$ as these were all operating under fidelity 1 and thus were less useful for inspecting bond dimension influence. For more information see Section 7.5.

Stylized Facts

We now examine how circuit depth, L, and bond dimension, χ, affect the reproduction of stylised facts. Figures 4.6 and 4.7 show the absolute-autocorrelation (volatility clustering) and leverage effect. The panels are arranged with rows corresponding to different layer depths $L \in \{1, 5, 10, 18\}$ and columns corresponding to bond dimensions $\chi \in \{1, 8, 16, 32\}$.

Models with $L = 1$ underperform across both stylised facts, consistent with Fig. 4.3a, which showed that fidelity saturates quickly and does not improve with larger χ. Across all depths, simulations with $\chi = 1$ exhibit weaker convergence as they effectively represent only product states. Nevertheless, the $L = 5, \chi = 1$ configuration shows a comparatively stronger leverage effect than other $\chi = 1$ and even $\chi > 1$ settings.

Table 4.4: *Unified loss comparison (average of last 200 epochs).*

	χ=1			χ=8		
Layers	ACF_Abs	ACF_NonAbs	Leverage	ACF_Abs	ACF_NonAbs	Leverage
1	—	—	—	1.486697	0.004275	0.042910
5	0.346551	0.008216	0.019317	0.254102	0.002470	0.046341
10	0.326421	0.002731	0.031527	0.215986	0.017101	0.033307
18	3.169999	0.325667	0.464376	0.270639	0.004764	0.042177

	χ=16			χ=24		
Layers	ACF_Abs	ACF_NonAbs	Leverage	ACF_Abs	ACF_NonAbs	Leverage
1	1.486697	0.004275	0.042910	1.486697	0.004275	0.042910
5	—	—	—	—	—	—
10	0.298751	0.007301	0.043093	0.258265	0.003061	0.038203
18	—	—	—	0.293117	0.004313	0.037966

Version of November 21, 2025– Created November 21, 2025 - 12:28

	χ=32		
Layers	ACF_Abs	ACF_NonAbs	Leverage
1	1.486697	0.004275	0.042910
5	0.404540	0.002893	0.035602
10	0.306334	0.002795	0.031907
18	0.331789	0.003281	0.032223

Table 4.4 shows the resulting stylized facts loss values. We can observe a clear improvement of stylized facts for increasing bond dimension. For increasing the number of layers the trend is less clear but overall higher layer count does improve stylized facts. When we compare the loss values for $L = 10, \chi = 32$ with $L = 18, \chi = 32$ one would get the idea that $L = 10$ produces a better result. But when we visually inspect the stylized facts we would argue that $L = 18$ produces a result more comparable with the observed stylized facts seen in the S&P 500. Overall, performance improves with increasing L and χ, with the deepest circuits ($L = 18$) and highest bond dimension ($\chi = 32$) yielding the best reproduction of both volatility clustering and the leverage effect (barring the good result from $L = 5, \chi = 1$. For comparison, the stylised facts of the S&P 500 are plotted in Fig. 4.8 next to one of the best results from the simulation with $L = 18, \chi = 32$. (see Sec. 2.1) The distributional fit of the model aligns well with the empirical data. The volatility clustering (ACF for absolute log returns) of the model seems to decay for longer lags. The magnitude, however, is still lacking when compared to the observed volatility clustering in the SP 500 data. Standard ACF is adequate. The leverage effect seemed to be the most difficult aspect to capture for all the models across the board. The $L = 18, \chi = 32$ model captured the trend of the negative leverage effects for low lags, but it was again lacking in magnitude. For higher lags, the model struggled to capture the leverage effects.

Version of November 21, 2025– Created November 21, 2025 - 12:28

Figure 4.6: *Plot of absolute log returns of the generated data using QGAN for various MPS configurations.*

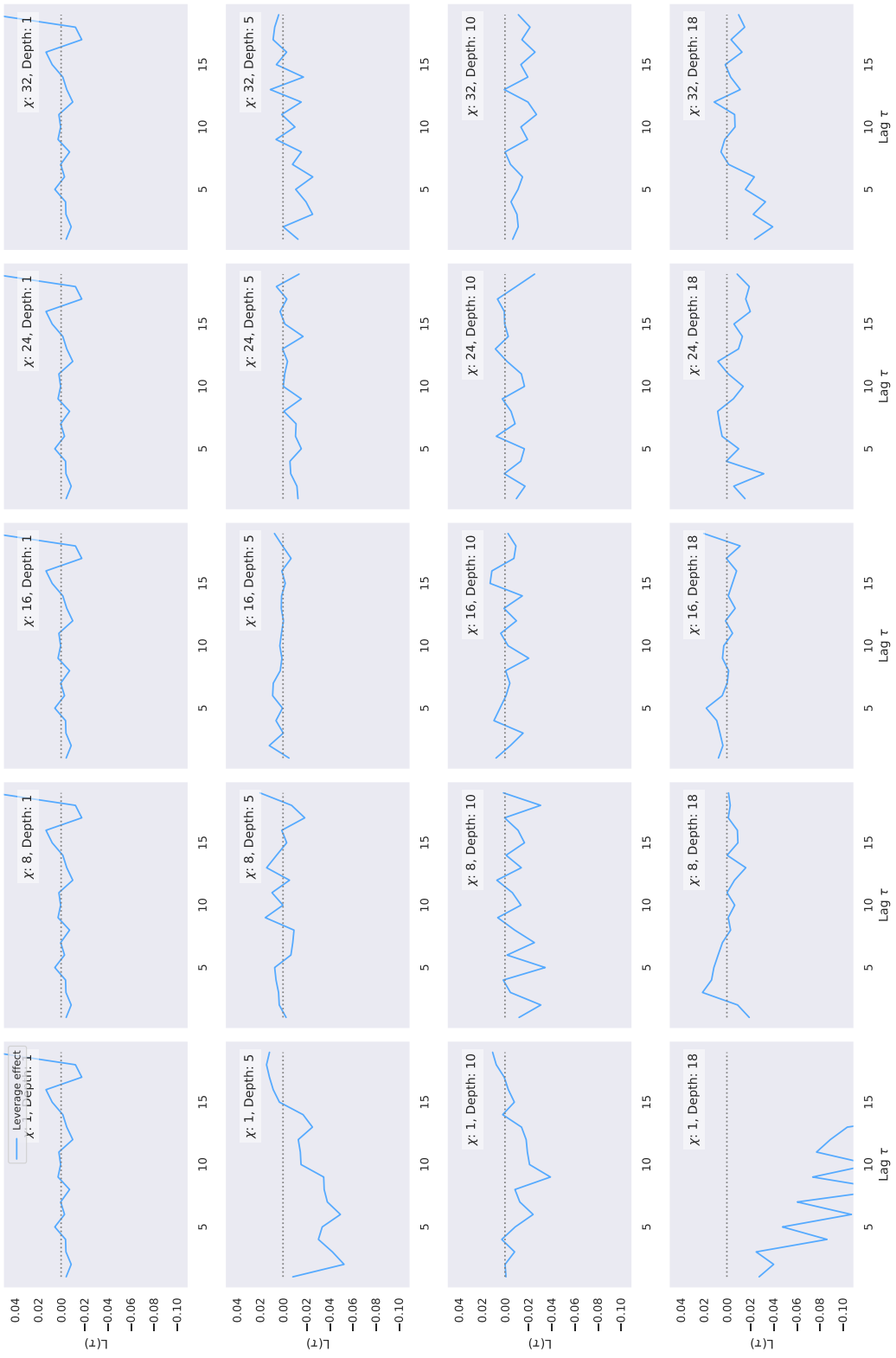


Figure 4.7: *Plot of the leverage effect of the generated data using QGAN for various MPS configurations.*

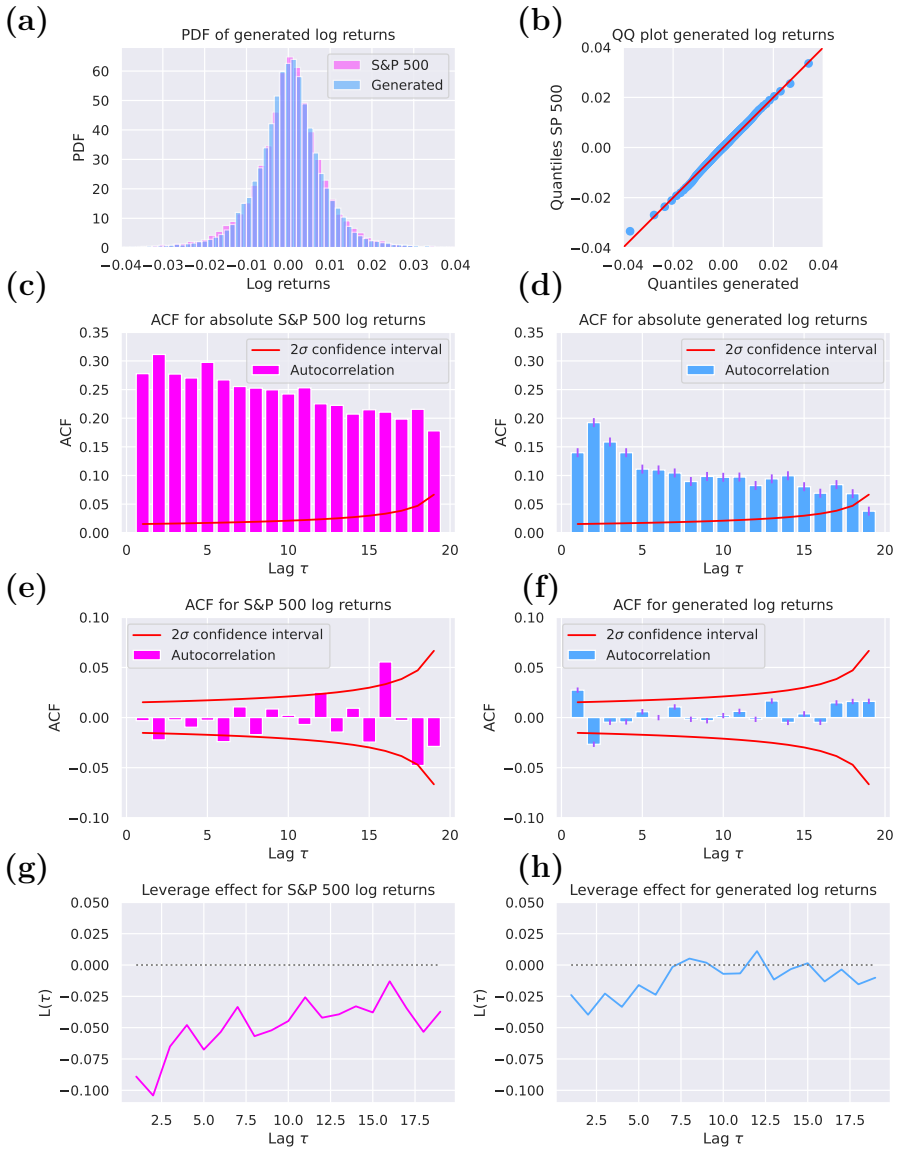


Figure 4.8: *Metrics of the stylized facts for a synthetic time series of window size* 20 *generated by a QGAN with 18 layers and bond dimension 32. (right side), compared to the metrics of the S&P 500 index (left side). In* ***(a)****, we plot the probability density functions and in* ***(b)*** *the quantile-quantile plot of both the S&P 500 index and the generated time series. In* ***(c)****-****(h)****, we plot the metrics absolute autocorrelation, linear autocorrelation and the leverage effect, as an indication of the stylized facts as described in Section 2.1. The Subfigures* ***(c)****,* ***(e)*** *and* ***(g)*** *show the metrics of the S&P 500 index and the Subfigures* ***(d)****,* ***(f)*** *and* ***(h)*** *the metrics of the generated time series, respectively.*

RQ2: How quickly do QGANs with different circuit configurations reach a defined training plateau, and what are the implications for scenarios with time constraints?

We qualitatively judge the training plateau of the model through plotting the loss values per epoch in a log plot. In Figure 4.9, we plot the loss of the model as a function of the training epochs for a simple model with $L = 1$, a more complex model with $L = 5$, and a complex model with $L = 18$, while keeping the bond dimension constant $\chi = 32$ and sufficient for approximating the full-state with fidelity unity for all models.

We anticipated that simpler quantum circuit configurations would be quicker to train and would reach the performance plateau sooner than more complex configurations. This expectation is based on the hypothesis that increased complexity, in the form of more layers and higher bond dimensions, introduces a greater number of parameters and more entanglement that needs to be optimised, thereby extending the training duration. Plotting loss against training epochs (Figure 4.9) reveals differences in convergence behaviour. The $L = 1$ models reaches a stable loss plateau around epoch 400, declining from $\ 0.004$ to $\ 0.0023$ within this period. In contrast, $L = 5, \chi = 32$ and $L = 18, \chi = 32$ models show continued loss reduction throughout the entire 8000-epoch training period, with no clear plateau visible. At epoch 8000, both deeper models exhibit ongoing decline, with $L = 5$ reaching $\ 0.0005$ and $L = 18$ reaching $\ 0.0003$, but the trajectories suggest substantially more training is needed for convergence.

This prevents us from directly comparing convergence times for $L > 1$ models, as we can only say that they require > 8000 epochs.

To get a better idea of this effect, we plotted the loss function as a function of the epochs trained for various circuit configurations in Figure 4.10. The image is a table with the rows indicating various layer depths $L \in \{1, 5, 10, 18\}$. In the columns, we differentiate between the bond dimensions used in the simulations $\chi \in \{1, 8, 16, 32\}$. First of all, we can see some behaviour that we expected. Recall Figure 4.3a. We can, again, see that for $L = 1$ increasing the bond dimension above $\chi = 8$ does not increase the fidelity, as we have already reached a fidelity of 1 for $\chi = 8$. We would then expect the accuracy of the QGAN not to improve by increasing the bond dimension past $\chi = 1$ for $L = 1$. Indeed, in Figure 4.10 we see that for $L = 1$ the loss does not significantly decrease when increasing χ. Furthermore, we also see that for $L > 5$, $\chi = 1$ is not nearly sufficient, and the accuracy obtained after convergence is far worse than that of the higher bond

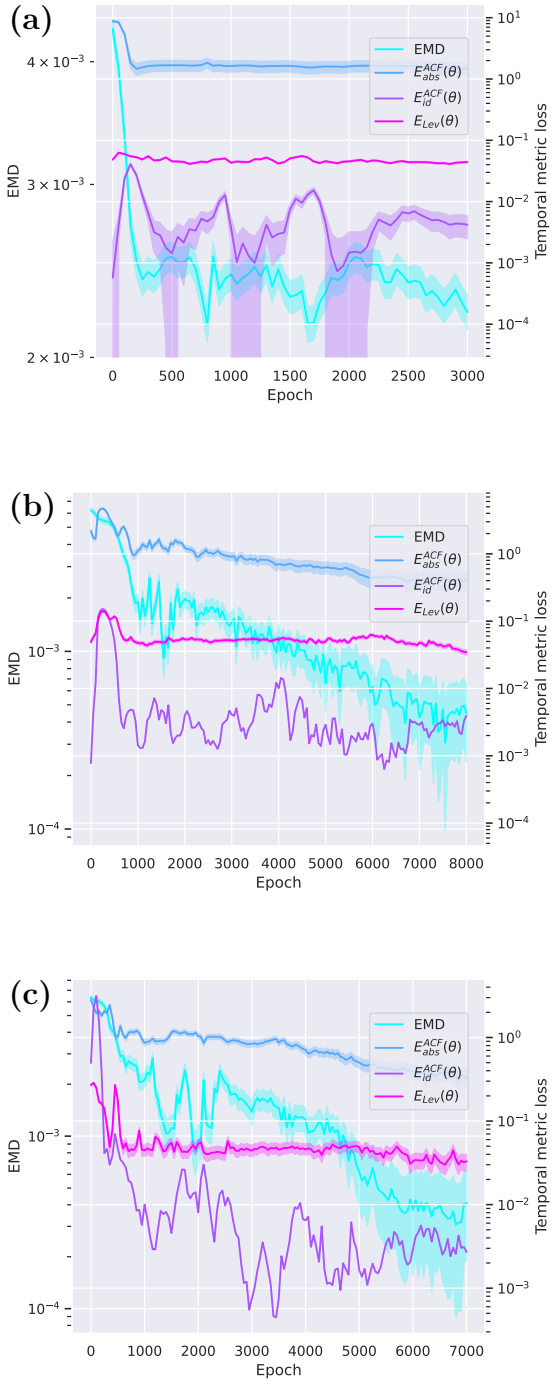


Figure 4.9: *Model loss per epoch. Loss as a function of epochs for MPS simulations with* ***(a)*** $L = 1$, $\chi = 32$; ***(b)*** $L = 5$, $\chi = 32$; ***(c)*** $L = 18$, $\chi = 32$. *Note the x-axis:* 3000 *epochs for* $L = 1$; 8000 *epochs for* $L = 5$ *and* $L = 18$. *All images are log-scaled.*

Version of November 21, 2025– Created November 21, 2025 - 12:28

dimension runs for the same amount of layers. In fact, we see the same result as we saw in Figure 4.5. Increasing the bond dimension tends to increase accuracy until the point at which further increases in bond dimension do not enhance fidelity in our results in Figure 4.3. We see the same trend with the increasing number of layers; the accuracy also tends to improve.

As the number of trainable parameters and the number of CNOT gates increase, we also observe that the time to convergence increases when $L > 1$ compared to $L = 1$. For the higher layer amounts $L = 5, 10, 18$ not such a strong trend can be observed for an epoch count up to 8000. This, together with the images from Figure 4.9 may suggest that longer training is necessary to investigate the behaviour of more in depth models and their convergence. Our pre defined training length may not have been long enough to saturate learning from the model.

An intriguing phenomenon observed during the training of the MPS-based QGAN simulations is an initial spike in the (visibly) autocorrelation function (ACF) loss and leverage effect within approximately the first 1000 epochs. This spike consistently occurs across different circuit configurations. See 6.2 for further discussion.

Version of November 21, 2025– Created November 21, 2025 - 12:28

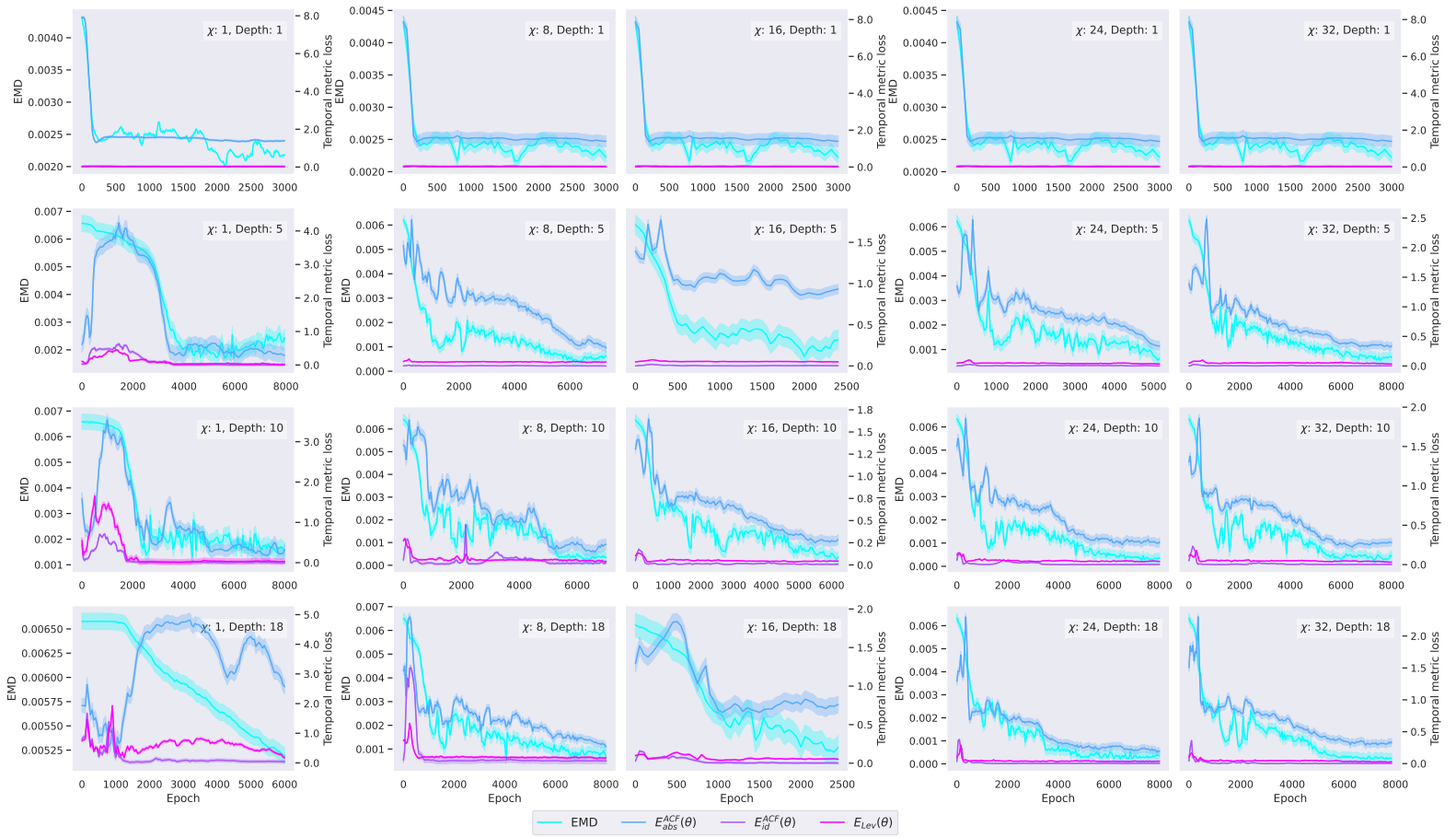


Figure 4.10: *Plot of training efficiency versus circuit complexity, demonstrating the time to reach an accuracy plateau for various QGAN configurations. This plot elucidates the practical implications of circuit design choices in time-sensitive applications.*

52

We further compare these MPS results with the slightly modified full-state results obtained by Ref. [60] in Figure 4.11. To make the comparison fair we changed the entangling gate in the full state simulation from the original used by Ref. [60] which was a CZ gate to a CNOT gate.

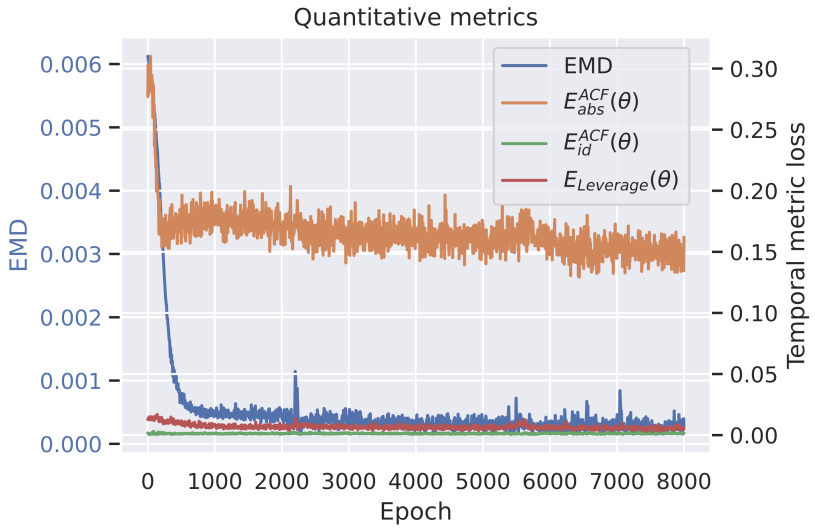


Figure 4.11: *Ref [60] full-state results with the adjusted CZ to CNOT gate change. Showing the model performance metrics during training*

Ref. [60] their simulations converge more rapidly than the MPS-based simulations, taking fewer epochs to minimise the Wasserstein loss than the MPS-based simulations. Even though the bond dimension used in the MPS simulation approximates the full state with high fidelity. In table 4.5, we quantitatively compare the MPS-based simulation of $L = 18, \chi = 32$ with the full state simulation from [60] The full-state approach achieves lower losses for the leverage effect

Table 4.5: *Comparison of lowest loss values between MPS and Full state simulations (CNOT).*

Loss Metric	**MPS**	**Full state**
L_{Wass}	0.000291	0.000327
$L_{\text{ACF, non-abs}}$	0.00044	0.001935
$L_{\text{ACF, abs}}$	0.30919	0.03738
L_{Leverage}	0.02190	0.00662

(0.00662 vs 0.02190) and absolute autocorrelation/volatility clustering (0.03738

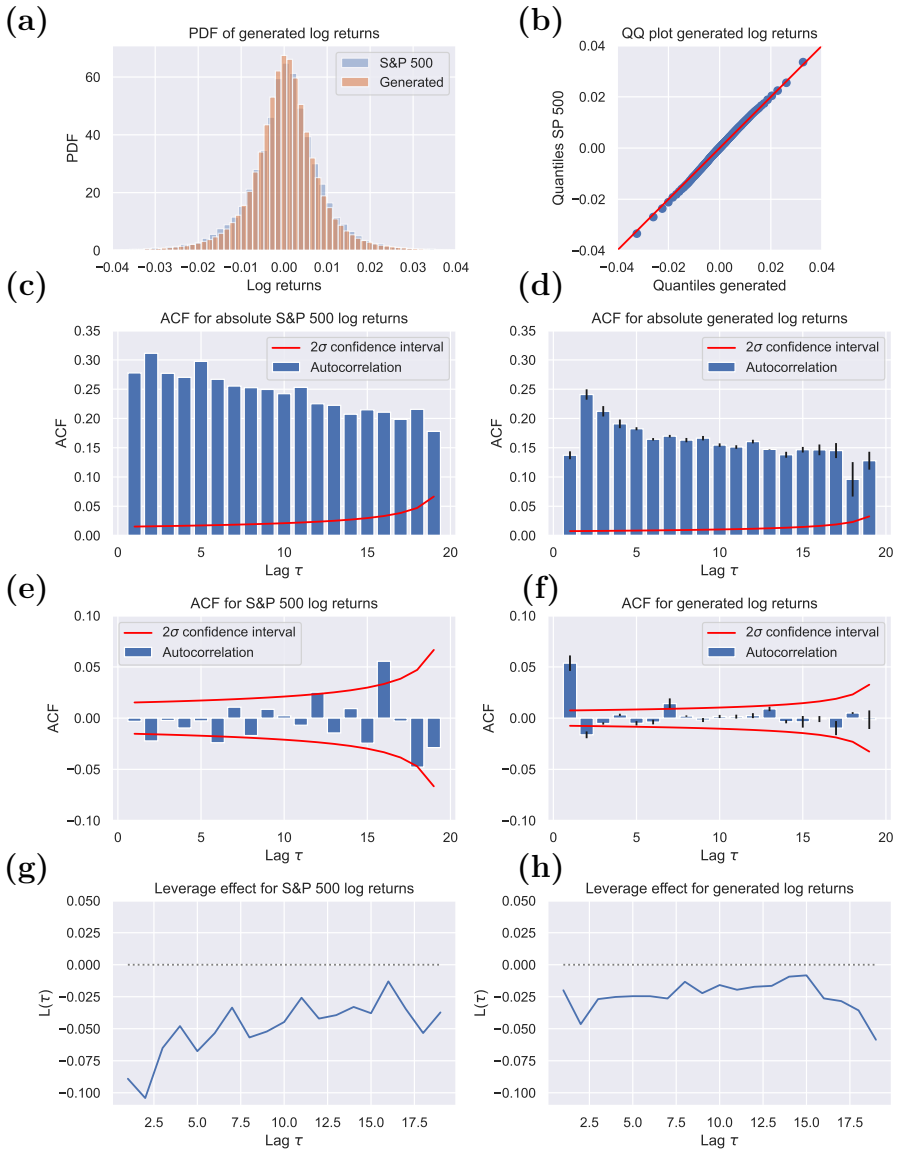


Figure 4.12: *Stylized facts for the S&P500 and the full-state simulation, compared to the metrics of the S&P 500 index (left side). In* **(a)**, *we plot the probability density functions and in* **(b)** *the quantile-quantile plot of both the S&P 500 index and the generated time series. In* **(c)**-**(h)**, *we plot the metrics absolute autocorrelation, linear autocorrelation and the leverage effect, as an indication of the stylized facts as described in Section 2.1. The Subfigures* **(c)**, **(e)** *and* **(g)** *show the metrics of the S&P 500 index and the Subfigures* **(d)**, **(f)** *and* **(h)** *the metrics of the generated time series, respectively.*

vs 0.30919). However, the MPS simulation achieves slightly better Wasserstein distance (0.000291 vs 0.000327) and substantially lower loss on non-absolute autocorrelation (0.00044 vs 0.001935). One reason for this could be the way MPS introduces a different optimisation landscape, one with additional complexity due to the tensor network structure. For further discussion, see 6.1.

RQ3: How do composite loss functions, which combine the Wasserstein loss with penalties targeting stylized facts, affect QGAN performance?

In addressing Research Question 3, we investigated how composite loss functions influence the QGAN's ability to reproduce stylised facts in financial markets. While the Wasserstein loss ensures distributional alignment between real and generated data, it does not directly enforce temporal properties such as volatility clustering or the leverage effect. To address this, we introduced composite loss functions, as discussed in Section 2.1, that combine the Wasserstein loss with additional penalty terms targeting the stylised facts. The general form of the generator loss is

$$L = \lambda_w \cdot L_{\text{Wass}} + \lambda_v \cdot L_{\text{volatility}} + \lambda_l \cdot L_{\text{leverage}}, \tag{4.1}$$

where $\lambda_w, \lambda_v, \lambda_l$ controls the relative contribution of each term. Our aim was to test whether adding to the Wasserstein loss with stylised-fact penalties could improve the realism of generated financial time series beyond distributional similarity. Experiments were conducted using $L = 5$ layers and a bond dimension of $\chi = 32$.

Equal weighting of Wasserstein and volatility clustering losses

First, we set $\lambda_w = \lambda_v = 0.5$, giving equal importance to the Wasserstein distance and volatility clustering. The resulting training is shown in Fig. 4.13. Overall performance was weaker than the Wasserstein-only baseline. In particular, the generator struggled to minimise the Wasserstein distance effectively, suggesting that the additional volatility term dominated the optimisation. This outcome indicates that an overly strong weighting of stylised-fact penalties can worsen overall training stability.

Version of November 21, 2025– Created November 21, 2025 - 12:28

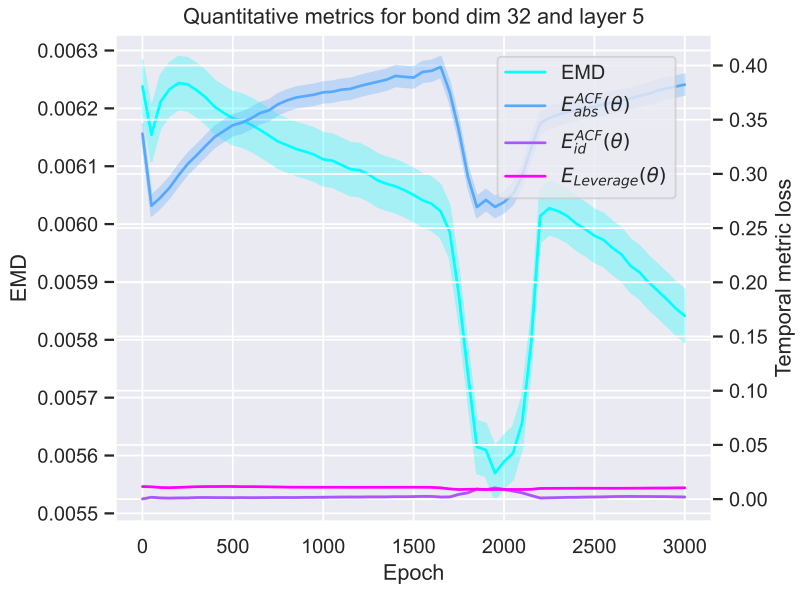


Figure 4.13: *Composite loss:* $L = \frac{1}{2}L_{Wass} + \frac{1}{2}L_{volatility}$.

Reduced weighting of volatility clustering

To help reduce the negative effect observed in the equal-weighted case, we lowered the volatility clustering weight to $\lambda_v = 0.1$ while keeping $\lambda_w = 0.9$. As shown in Fig. 4.14, this adjustment led to slightly more stable training and marginally improved results compared to the 0.5/0.5 setting. The training model was able to achieve relatively low EMD values, but when we look at the generated stylised facts in Figure 4.15 and compare them to the baseline Wasserstein only training from Figure 4.16, we see that despite the addition of the ACF in the loss function, the model performed below the Wasserstein only baseline.

Version of November 21, 2025– Created November 21, 2025 - 12:28

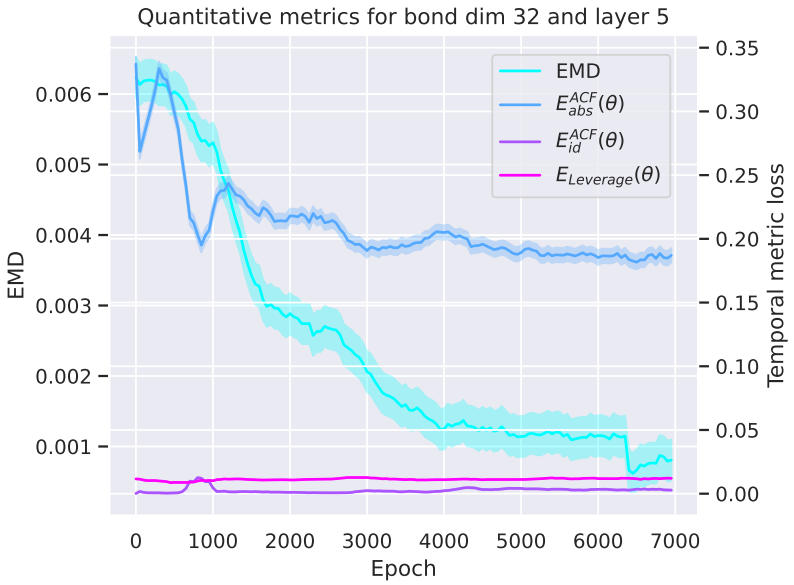


Figure 4.14: *Composite loss:* $L = 0.9L_{Wass} + 0.1L_{volatility}$.

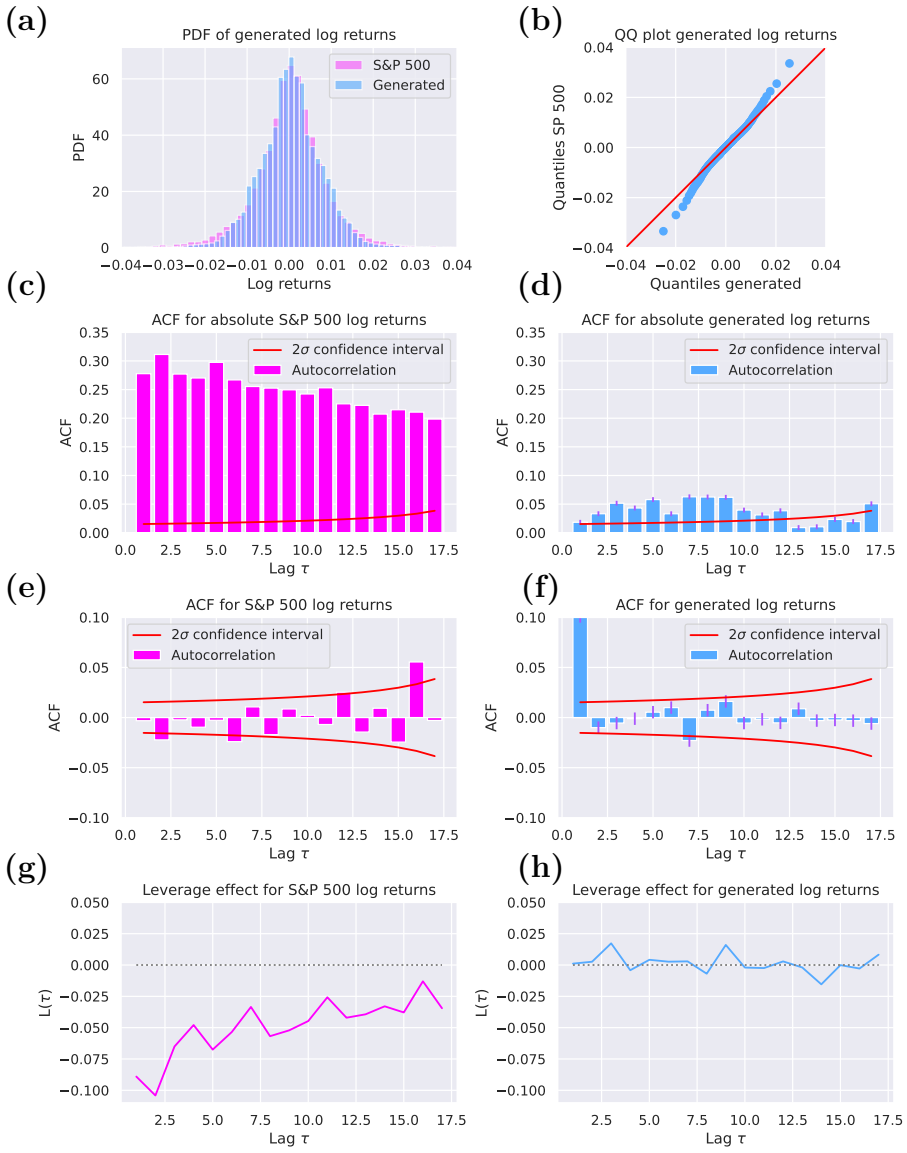


Figure 4.15: *Observed stylized facts of S&P500 and the model with Composite loss:* $L = 0.9L_{Wass} + 0.1L_{volatility}$*, compared to the metrics of the S&P 500 index (left side). In* ***(a)****, we plot the probability density functions and in* ***(b)*** *the quantile-quantile plot of both the S&P 500 index and the generated time series. In* ***(c)****-****(h)****, we plot the metrics absolute autocorrelation, linear autocorrelation and the leverage effect, as an indication of the stylized facts as described in Section 2.1. The Subfigures* ***(c)****,* ***(e)*** *and* ***(g)*** *show the metrics of the S&P 500 index and the Subfigures* ***(d)****,* ***(f)*** *and* ***(h)*** *the metrics of the generated time series, respectively.*

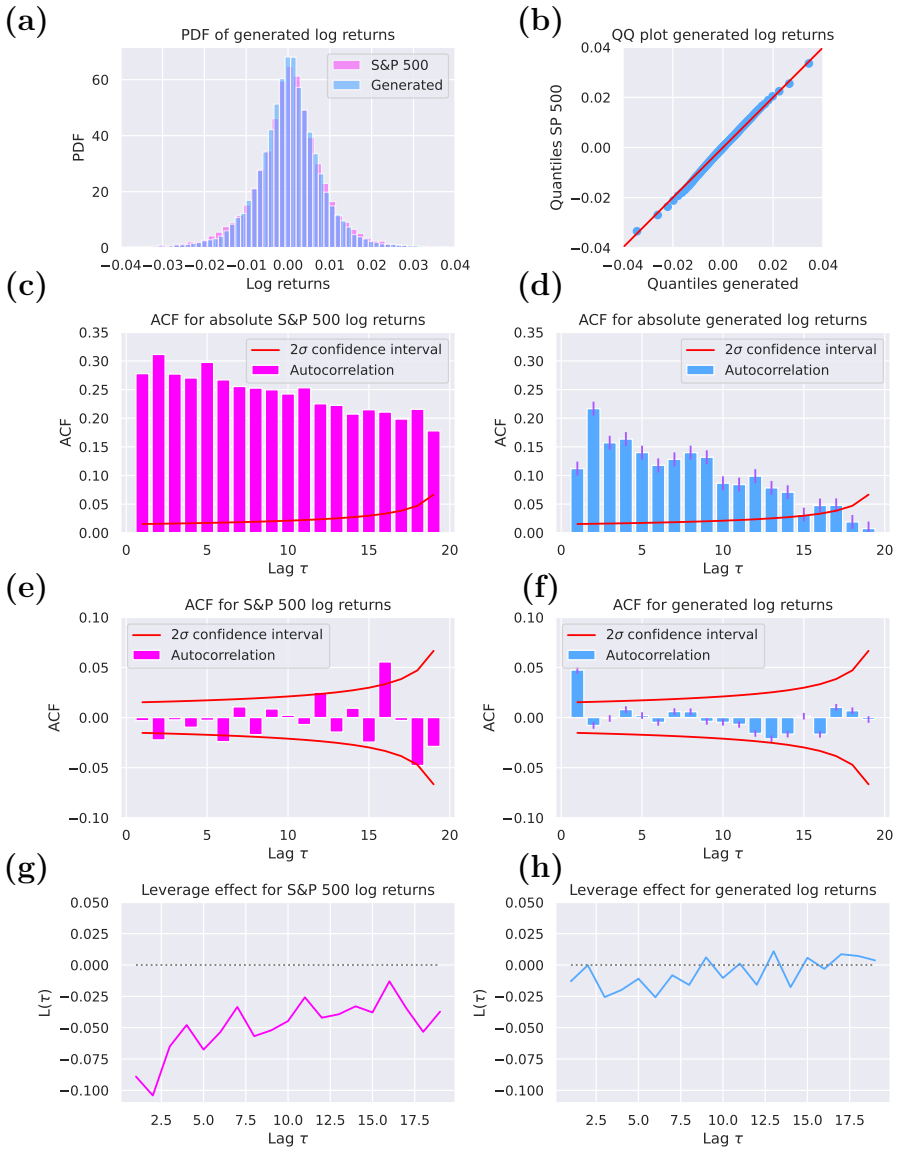


Figure 4.16: *Observed stylized facts of S&P500 and the model with $L = 5$, $\chi = 32$ trained on only the Wasserstein loss, compared to the metrics of the S&P 500 index (left side). In* ***(a)****, we plot the probability density functions and in* ***(b)*** *the quantile-quantile plot of both the S&P 500 index and the generated time series. In* ***(c)****-****(h)****, we plot the metrics absolute autocorrelation, linear autocorrelation and the leverage effect, as an indication of the stylized facts as described in Section 2.1. The Subfigures* ***(c)****,* ***(e)*** *and* ***(g)*** *show the metrics of the S&P 500 index and the Subfigures* ***(d)****,* ***(f)*** *and* ***(h)*** *the metrics of the generated time series, respectively.*

Applying composite loss to both generator and critic

An additional source of poor performance could be the asymmetric training structure introduced by applying the additional loss component solely to the generator. Typically, stable WGAN training arises from similar loss functions for both the generator and the discriminator. Incorporating specialised terms into only the generator's loss could disrupt the equilibrium, resulting in an underperforming generator.

To test this hypothesis, we introduced the composite loss to both the generator and critic simultaneously, as shown in Figure 4.17. Although slightly improved, this approach still fails to outperform the standard Wasserstein loss method.

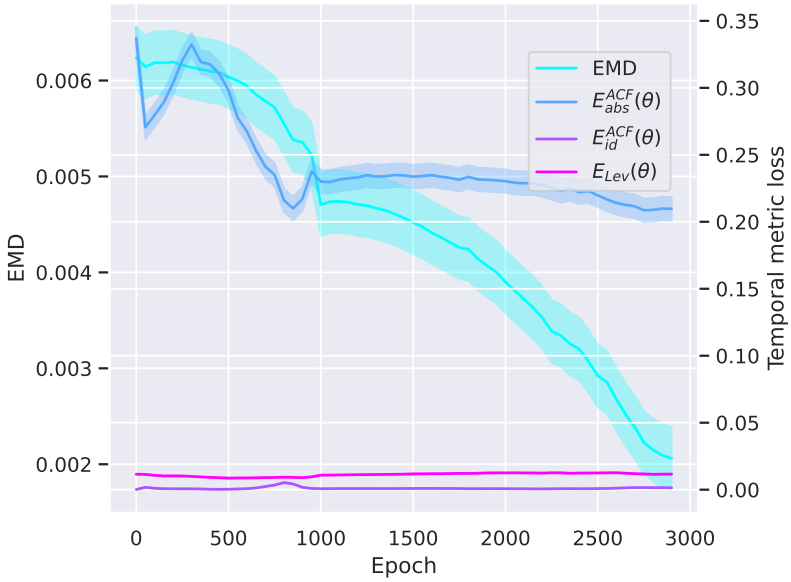
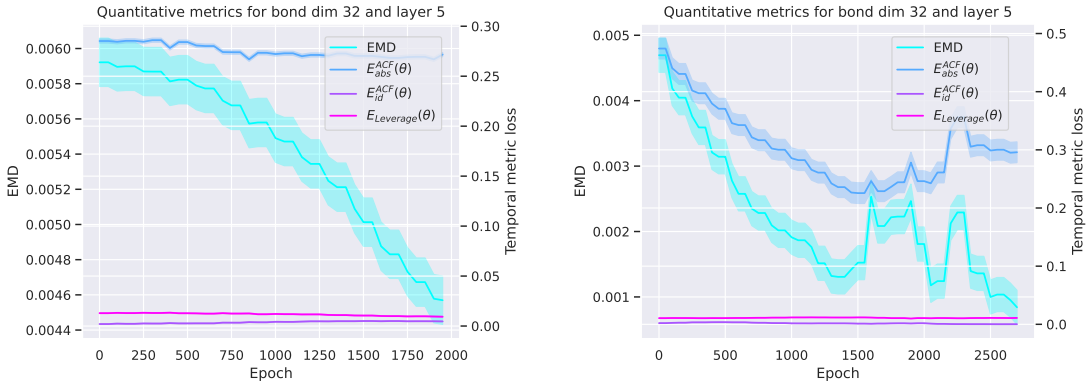


Figure 4.17: *Composite loss function applied to both generator and critic:* $L = 0.9L_{Wass} + 0.1L_{volatility}$.

Periodic isolation of the generator

To explore alternative strategies, we periodically isolated the generator from the GAN training loop to exclusively train it on either volatility clustering or leverage effects. Three schedules were tested: (i) 25 epochs every 50 GAN epochs, (ii) 1 epoch every 2 GAN epochs, and (iii) 500 epochs every 1000 GAN epochs (Figs. 4.18, 4.19, and 4.20). The models were very unstable during training and very frequently had exploding gradients resulting in unusable results. We could not complete all simulations in the time frame for this thesis. For this reason the

Version of November 21, 2025– Created November 21, 2025 - 12:28



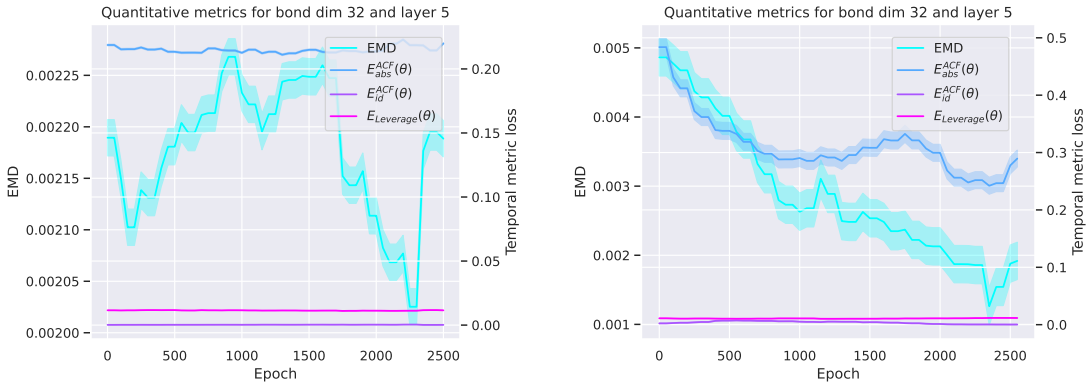
(a) *Generator trained on only volatility clustering during removal from GAN training loop.*

(b) *Generator trained on only leverage effect during removal from GAN training loop.*

Figure 4.18: *Generator removed from GAN training loop for 25 epochs every 50 GAN epochs.*

models were run for a smaller amount of epochs and this was compared against the base line Wasserstein only simulation.

It is interesting to note that in Figure 4.19 one can see that the leverage effect more closely aligns with the earth moving distance as they both decline during training, while the volatility clustering is very volatile. This is probably due to the chance of the initialised parameters of the model, though the opposite seems to be happening in Figure 4.18 Also, these attempts (Figures 4.18, 4.19, and 4.20) showed diminished returns compared to training with Wasserstein loss alone.



(a) *Generator trained on only volatility clustering during removal from GAN training loop.*

(b) *Generator trained on only leverage effect during removal from GAN training loop.*

Figure 4.19: *Generator removed from GAN training loop for 1 epoch every 2 GAN epochs.*

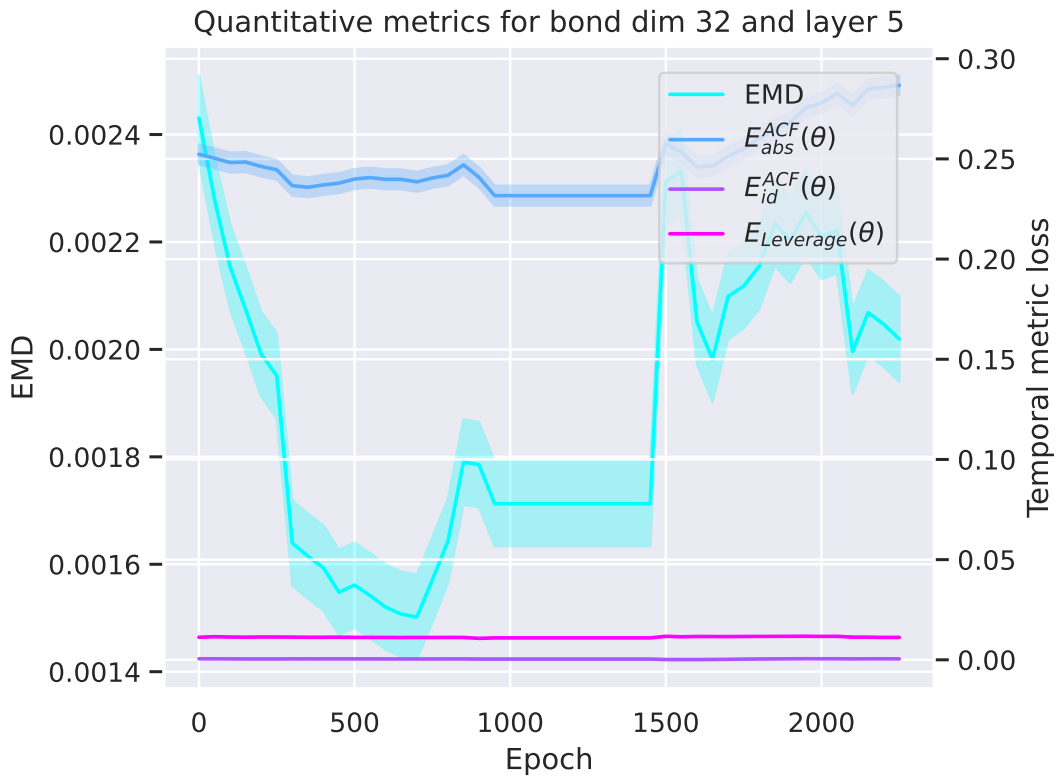


Figure 4.20: *Generator removed from GAN training loop for 500 epochs every 1000 GAN epochs. Generator trained on only volatility clustering during removal from GAN training loop.*

The best performing model, Generator trained on only leverage effect for 25 epochs during removal from 50 GAN training epochs Fig 4.18b, was trained until epochs 6300 for a fair comparison to the Wasserstein only baseline. See the results in Figure 4.21. As we can see the model again trained sub par compared to the Wasserstein only baseline (Figure 4.16). All metrics we worse than the baseline Wasserstein only model. Also the leverage effect was worse. What is interesting to see is that the leverage effect in Figure 4.21 seems to be a relatively smooth line when compared to all the Wasserstein only models (Figure 4.8).

Hence, for practical applications targeting specific stylised financial properties, one is better off selecting suitable configurations from an array of pre-trained models, as demonstrated in Figures 4.6 and 4.7.

Version of November 21, 2025– Created November 21, 2025 - 12:28

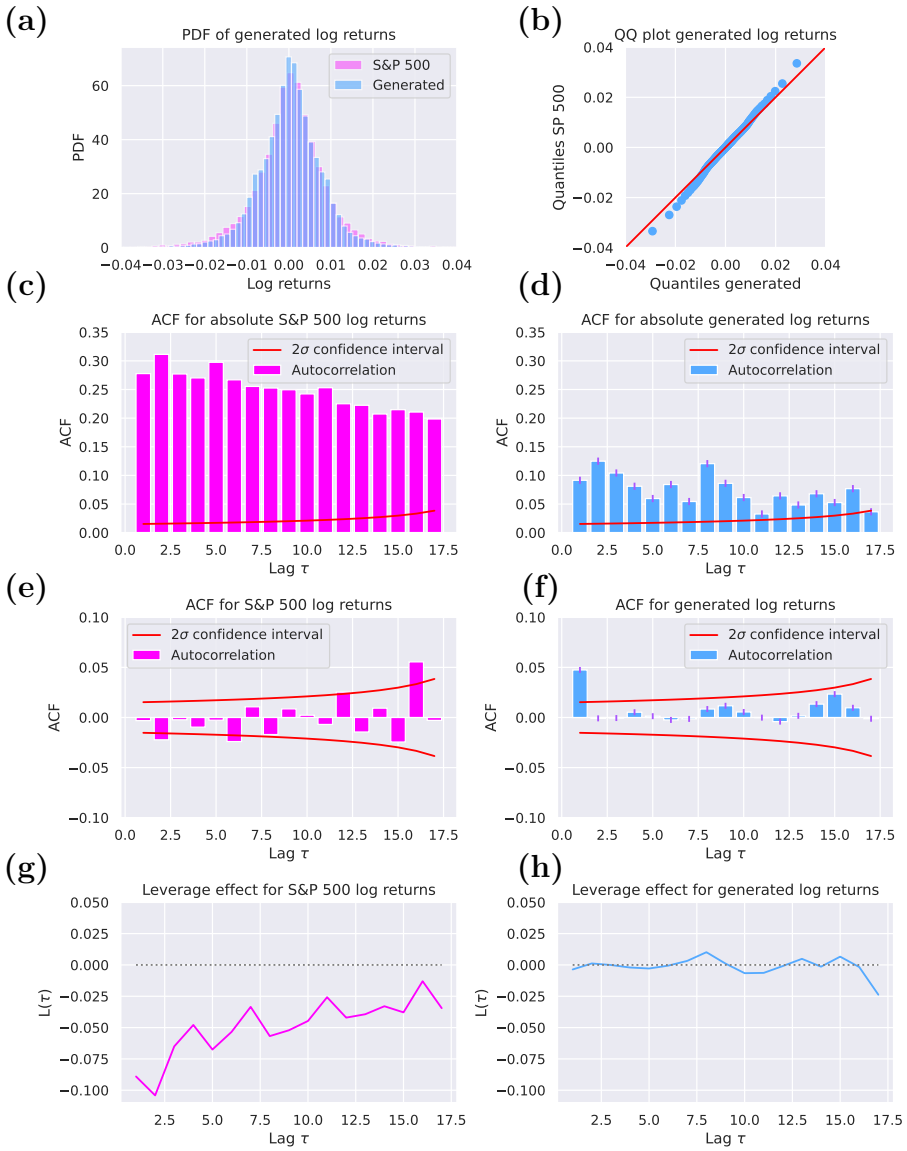


Figure 4.21: *Observed stylized facts of S&P500 and the model with $L = 5$, $\chi = 32$ where the generator was removed from GAN training loop for 25 epochs every 50 GAN epochs training in that period only on leverage effect, compared to the metrics of the S&P 500 index (left side). In* ***(a)****, we plot the probability density functions and in* ***(b)*** *the quantile-quantile plot of both the S&P 500 index and the generated time series. In* ***(c)****-****(h)****, we plot the metrics absolute autocorrelation, linear autocorrelation and the leverage effect, as an indication of the stylized facts as described in Section 2.1. The Subfigures* ***(c)****,* ***(e)*** *and* ***(g)*** *show the metrics of the S&P 500 index and the Subfigures* ***(d)****,* ***(f)*** *and* ***(h)*** *the metrics of the generated time series, respectively.*

RQ4: How does the number of qubits, which sets the maximum time resolution of the generated series, affect the QGAN's ability to reproduce stylized facts?

We examine how varying the number of qubits, N, impacts a QGAN's ability to replicate stylised facts. Since our setup encodes time steps as the result of measuring different observables of a quantum state, increasing N effectively increases the number of representable time steps $2N$. This was an intractable experiment involving full-state simulation, as the run time explodes for $N = 20$ qubits. Due to time constraints, we report preliminary results for up to 1000 epochs; see Figure 4.23. These already demonstrate that MPS-based training remains tractable at $N = 20$ qubits, where full-state simulations are infeasible.

These results collectively advance our understanding of the interplay between quantum circuit design and the performance of QGANs in financial modelling, providing valuable insights for future research and practical applications.

Version of November 21, 2025– Created November 21, 2025 - 12:28

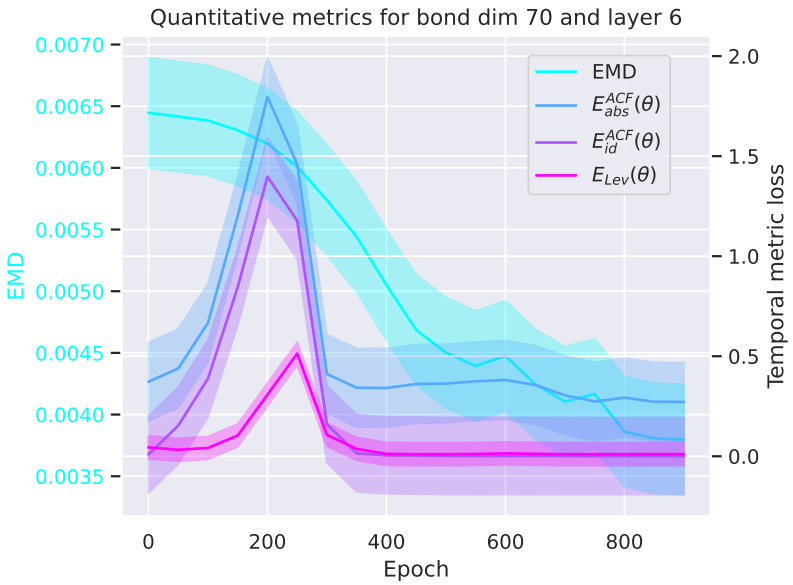


Figure 4.22: *Wasserstein loss vs training epochs with a QGAN featuring 20 qubits,* 6 *layers and* $\chi = 70$.

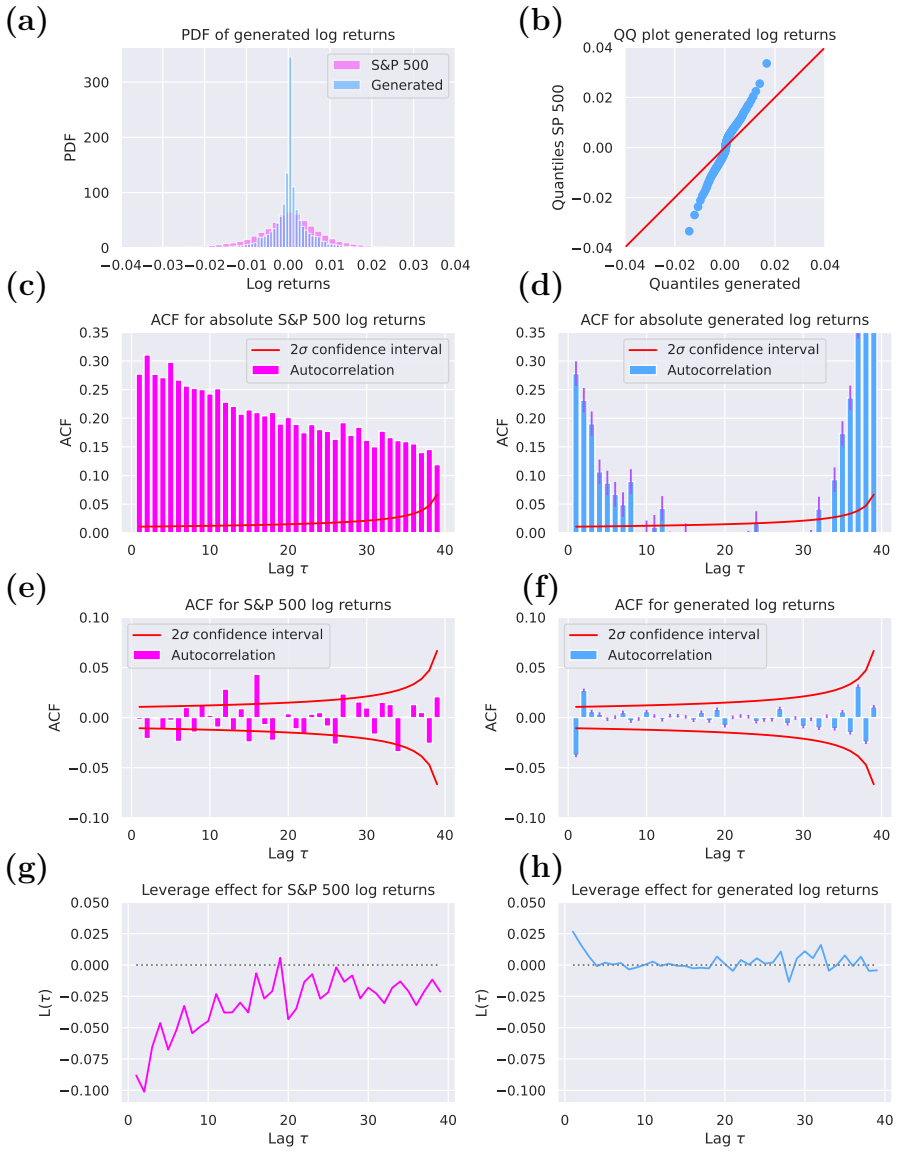


Figure 4.23: *Stylized facts of S&P500 and model featuring 20 qubits, 6 layers and $\chi = 70$, compared to the metrics of the S&P 500 index (left side). In* ***(a)****, we plot the probability density functions and in* ***(b)*** *the quantile-quantile plot of both the S&P 500 index and the generated time series. In* ***(c)****-****(h)****, we plot the metrics absolute autocorrelation, linear autocorrelation and the leverage effect, as an indication of the stylized facts as described in Section 2.1. The Subfigures* ***(c)****,* ***(e)*** *and* ***(g)*** *show the metrics of the S&P 500 index and the Subfigures* ***(d)****,* ***(f)*** *and* ***(h)*** *the metrics of the generated time series, respectively.*

Chapter 5

Conclusion and Outlook

In this thesis, we investigated the use of Matrix Product State (MPS)-based Quantum Generative Adversarial Networks (QGANs) as a novel approach to modelling financial time series. Our motivation stemmed from the observation that many classical models struggle to capture key empirical properties such as heavy tails, volatility clustering, and the leverage effect that characterise real-world financial time series. By harnessing quantum-inspired techniques, we aimed to determine whether QGANs, assisted by MPS representations in order to efficiently simulate circuits with a higher number of layers ($L > 10$) and higher-qubit ($N > 10$), could model richer data distributions while remaining classically tractable.

Our findings show that the choice of bond dimension, circuit depth, and qubit number is central to balancing model expressiveness against computational cost. Each parameter shapes how well the QGAN captures financial phenomena and how efficiently it can be trained.

Bond Dimension and Fidelity

We showed that MPS simulations can closely approximate full-state simulations but only if the bond dimension is chosen appropriately. For shallow circuits or few qubits, a small bond dimension suffices, enabling efficient training. As circuit complexity increases, higher bond dimensions become essential to preserve fidelity. By benchmarking fidelity across architectures, we established practical guidelines for setting bond dimensions based on the target circuit configuration. This insight allowed for optimised resource allocation in subsequent experiments. See Figure 4.3a.

Circuit Complexity and Wasserstein Loss

The relationship between circuit depth and Wasserstein loss depends on sufficient training time. Models with $L = 1$ consistently achieved moderate performance (loss ≈ 0.0023) across all bond dimensions $\chi \geq 8$, reaching their performance plateau within 400 epochs. For intermediate depth ($L = 5, 10$), increasing layer count generally improved performance, with $L = 10$ configurations achieving some of the lowest loss values ($L = 10, \chi = 24$: 0.000343).

Our deepest circuits ($L = 18$) showed mixed results. While $L = 18, \chi = 24$ achieved the best overall performance (0.000282) considering only Wasserstein loss. See Table 5.1. The continued decline in loss curves at 8000 epochs (Figure 4.9) indicates that $L \geq 10$ models had not fully converged within the training period, making direct comparisons of final performance difficult.

Bond Dimension and Wasserstein Loss

The impact of bond dimension on performance depends strongly on circuit depth. For shallow circuits ($L = 1$), increasing χ beyond 8 provided no improvement, consistent with the fidelity analysis showing that $\chi = 8$ is sufficient to exactly represent these states (Figure 4.3a).

For deeper circuits ($L \geq 5$), increasing bond dimension from $\chi = 1$ to $\chi = 8$ produced substantial improvements ($L = 10$: from 0.001636 to 0.000371). However, further increasing from $\chi = 16$ yielded more small and inconsistent gains.

The clearest benefit of higher bond dimension appears at $L = 18$, where $\chi = 24$ (0.000282) and $\chi = 32$ (0.000361) outperformed $\chi = 8$ (0.000821). This suggests that the bond dimension requirement scales with circuit depth, and that $\chi \geq 24$ is necessary to capture the entanglement structure of deeper circuits within the training time available. Extended training may be required to fully exploit higher bond dimensions.

Table 5.1: *Final loss values (average of last 200 epochs) for different configurations*

Layers	χ=1	χ=8	χ=16	χ=24	χ=32
1	—	0.002286	0.002286	0.002286	0.002286
5	0.002352	0.000598	—	—	0.000467
10	0.001636	0.000371	0.000432	0.000343	0.000422
18	0.005211	0.000821	—	0.000282	0.000361

Version of November 21, 2025– Created November 21, 2025 - 12:28

Time-to-Convergence vs. Model Complexity

Convergence behaviour varied with circuit depth. Shallow circuits ($L = 1$) reached a performance plateau within approximately 400 epochs across all tested bond dimensions, demonstrating rapid and stable training. In contrast, deeper models ($L \geq 5$) had continued loss reduction through the entire 8000-epoch training period (Figure 4.9), indicating that they had not reached convergence.

Within the 8000-epoch training window, we could not establish clear convergence times for $L \geq 5$ models, as their loss curves remained in decline. This prevented direct comparison of training efficiency across deeper architectures. Suggesting these models require substantially longer training (15000+).

This has practical implications: for applications needing rapid deployment (daily model retraining), $L = 1$ models provide immediate usability but limited performance. Deeper models ($L \geq 10$) offer better performance but require significantly greater training time.

Composite Loss Functions

Attempts to directly optimize stylized facts through composite loss functions were unsuccessful across all tested configurations. We evaluated several approaches on a baseline ($L = 5$, $\chi = 32$):

- Equal weighting of Wasserstein and volatility clustering losses ($\lambda_w = \lambda_v = 0.5$) badly degraded both Wasserstein loss and stylized fact reproduction.
- Reduced volatility weighting ($\lambda_w = 0.9$, $\lambda_v = 0.1$) marginally improved stability but still underperformed the Wasserstein-only baseline (Figure 4.14).
- Applying composite losses to both generator and critic simultaneously did not resolve the training instability.
- Periodic isolation of the generator for stylized fact training (tested at 3 different schedules) resulted better result but still subpar compared to the Wasserstein only baseline and these models were also highly unstable during training with frequent gradient explosions and inferior final performance.

These results show that Wasserstein distance and stylized fact penalties create conflicting optimization objectives that destabilize GAN training. Importantly, these experiments were limited to a single architecture ($L = 5$, $\chi = 32$) and relatively short training runs (6,300 epochs for the best-performing variant).

In practise, if someone is looking to capture specific stylized facts they should instead select appropriate pre-trained models from a range of architectures. Our results in Section 4.1 demonstrate that different (L, χ) configurations can generate

Version of November 21, 2025– Created November 21, 2025 - 12:28

different temporal properties. For example, the strong leverage effect in $L = 5, \chi = 1$ model suggesting that architecture selection is more effective than loss function tampering for targeting specific stylized facts.

Number of Qubits

Increasing the number of qubits increased the time resolution of the generated series. Using MPS allowed us to successfully simulate 20-qubit circuits efficiently, something intractable with full-state simulation, while still achieving active learning and reasonable training times. This demonstrates the scalability advantage of MPS for quantum-inspired financial modelling. Our results are still preliminary as we only trained for 1000 epochs.

Future Work

This research sparks the following interesting continuations of the research:

Train models for more epochs

As we saw in section 4 more epochs could result in a possible even lower loss function value and a higher stylized fact agreement with real world data.

Entanglement gate swap in MPS

Running the MPS circuit with CZ gates instead of CNOT gates (keeping all other layers identical) would test whether the architecture-induced biases are due to the simulator choice or the entanglement gates.

Cross-asset correlations

Extending the framework to include cross-asset correlations would make the models more relevant for systemic risk analysis and portfolio modelling (see Section 7.2).

Expanded Finance Applications

Finally, there is potential to extend the produced synthetic time series to financial tasks, such as portfolio optimisation or scenario analysis, especially when realistic synthetic data is scarce.

By focussing on the interplay between entanglement, model complexity, and stylised

Version of November 21, 2025– Created November 21, 2025 - 12:28

fact modelling, this work contributes to a promising path forward for quantum-inspired financial modelling. While significant challenges remain, particularly with respect to scalability and hardware limitations, our study underlines the feasibility and relevance of quantum-based techniques for financial modelling and points towards a future where quantum-inspired methods can meaningfully augment or even transform the way financial systems are analysed and simulated.

Chapter 6

Discussion

In the previous chapter, we presented our experimental results on MPS-based QGANs, focussing on the impact of bond dimension, circuit complexity, training time, composite loss functions, and qubit number. While these results demonstrate clear trends, some require more in-depth interpretation in the context of quantum-inspired modelling.

We reflect on the unexpected behaviours observed during training. We discuss two aspects. First, we compare the performance of MPS simulations against full-state simulation, reflecting on computational efficiency and fidelity. Second, we address the unexpected initial spike observed in the stylised fact loss during training and its implications for model stability and convergence.

6.1 Comparison MPS-and Full-state simulations

When comparing the results gathered from the MPS-based simulations and the full-state simulation from Ref [60] in Figure 4.8 and 4.11, we observe clear differences. First of all, regarding the **training convergence**: Across all tested circuit configurations (various bond dimensions and layer depths) and entanglement gate types, the MPS-based QGAN consistently required more training epochs to converge (to reach a loss plateau) compared to the full-state QGAN. This was so even when the MPS bond dimension was set high enough to ensure 100% fidelity of the state. So, despite having equal expressiveness, the MPS model's training was slower to reach the same Wasserstein loss minimum. This suggests that the optimisation landscape observed by the training algorithm differs between the two methods. Several factors could be the reason for this: **Hypothesis 1: Gauge freedom in MPS** MPS represents the state with many intermediate tensors, and there is freedom to redistribute amplitudes among these tensors without changing

the overall quantum state [61]. This can lead to a more complex, degenerate parameter space. Gradient descent (which we used to train the QGAN) might have to navigate many flat directions or unnecessary degrees of freedom in the MPS. This can have the effect of an optimisation landscape with broad plateaus or narrow valleys that slow down convergence. In contrast, the full-state model directly optimises the gate angles affecting the entire state at once, without an intermediate factorised representation, so it may have a simpler landscape for the optimiser.

Hypothesis 2: Tensor contraction and entanglement representation In the MPS, entanglement is introduced gradually via nearest-neighbour CNOTs that increase the bond dimensions between adjacent qubits. We allow the bond dimension to grow up to the full-state limit; however, the state remains represented as a chain of tensors. After each two-qubit gate, the state is factorised (via the SVD) into the MPS form. The gradients must go through these SVD operations and the chain of contracted tensors. This influences how entanglement is divided at each step. If the optimiser has not yet found it useful to increase entanglement between certain qubits, the MPS will continue to represent a low-entangled state, and changing that could involve overcoming a mountain in the loss landscape. So the generator might need to, for a short period, worsen some aspects of the loss to increase entanglement before it can improve overall. This challenge is known in tensor-network-based optimisation [61]. In our case, we use straight gradient descent on all gate parameters, which might not exploit the full power of the MPS structure. In the full-state representation, entanglement is not something that the optimiser needs to explicitly address by singular values. It arises as soon as entangling gates are applied. There is no factorisation after each gate. The state-vector remains a single entity. So, any entangling gate immediately mixes the amplitudes globally. There is no concept of bond dimensions or truncation in the full-state simulation. The only limitation is the circuit's depth. If the circuit is the same, one would think that both the MPS and the full state have the same set of states. But in practice, the MPS's way of representing those states can make some configurations easier or harder to find.

Consider representing a highly entangled state that exhibits correlations between qubit 1 and qubit N. In the full state, as long as the circuit has a path of entangling gates from 1 to N, the optimiser can adjust those gates to create that correlation. In the MPS, similar phenomena occur; however, long-range correlations will only appear if each intermediary bond (1-2, 2-3, ..., N-1-N) has built up the necessary entanglement. If any one of those bonds remains low, qubit 1 and N remain effectively uncorrelated. The tensor network must correctly perform all the intermediate entanglements. This sequential entanglement build-up may lead to a spikier loss surface. There might be local minima that represent states with only

short-range entanglement that partly, but not fully, fit the data. The optimiser can become stuck fitting the local correlations and has difficulty moving to states of long-range entanglement.

The full-state model might be better at the global entangled solutions since it doesn't have to perform tensor factorisation at each step. So, the full-state treats the system as a whole. The MPS treats it as a chain of parts that must work together. This difference in entanglement representation can explain why the MPS architecture exhibits a different loss landscape than the full-state one.

One interesting outcome we see from the two implementations, even though they are both trained on the same loss (just Wasserstein distance), is that they both ended up being good at different stylised facts of the financial time series: Under that, the previous experiments with the full state simulation using the CZ gates, MPS tended to reproduce absolute autocorrelation (volatility clustering) better, while the full state captured the leverage effect more clearly. After correcting the full state implementation and aligning the gates to both use CNOT gates, the results change drastically. The full state now achieves lower losses on both absolute-ACF and leverage, while MPS retains a slightly lower L_{Wass} (see Table 7.1). This indicates that the earlier difference in results was not just a simulator consequence, but was at least partly caused by gate choice.

6.2 Initial spike in stylized facts loss

Initially, the quantum generator is initialised with random rotations. In this early period, the quantum state tends to accurately capture simple local stylized by chance. Maybe because of the low entanglement in the beginning. This further helps our argument in section 6.1.

As the training begins, parameter updates alter the circuit parameters. Due to the tensor-network structure in the MPS, small parameter adjustments in the beginning epochs may disrupt local correlations, causing a spike in the stylised facts loss.

After this spike, the training stabilises. Perhaps entanglement gradually starts as the MPS tensors change, reflected by a more homogeneous structure of singular values. This entanglement allows for a more accurate representation of local correlations. So, the ACF loss starts to decrease as training runs.

This behaviour shows how the unique optimisation problem of MPS-based quantum generative models relies on initial adjustments in entanglement structure for results.

Version of November 21, 2025– Created November 21, 2025 - 12:28

Chapter 7

Appendices

7.1 Classical Modelling

When studying financial time series, there are two methods of approaching this problem. Stochastic processes and agent-based models. Stochastic processes describe financial time series as a series of random variables with temporally dependent parameters. A model like the ARCH model is a statistical model that describes the variance of the current error term as a function of the actual sizes of the previous time periods and their error terms [62]. The variance is related to the squares of the previous innovations. The ARCH model is appropriate when the error variance in a time series follows an autoregressive (AR) model. If an autoregressive moving average (ARMA) model is assumed for the error variance, the model is a generalised autoregressive conditional heteroskedasticity (GARCH) model. [63]. It is difficult, however, to recover all the major stylised facts with such explicit mathematical formulations. Agent-based models describe an agent's interactions with other agents and the environment. The interaction leads to complex collective behaviour and the emergence of the stylised facts. It is difficult, however, to design these agents' behaviour and to calibrate a large number of model parameters based on real data.

We want a model that can reproduce the stylised facts but can be built without any assumptions about the original data. Deep learning, especially deep generative models, could be the solution. Generative adversarial networks (GANs) have especially shown their ability in the generation of data. They have shown promising results when generating images and audio [64, 65] and for financial data [66–68].

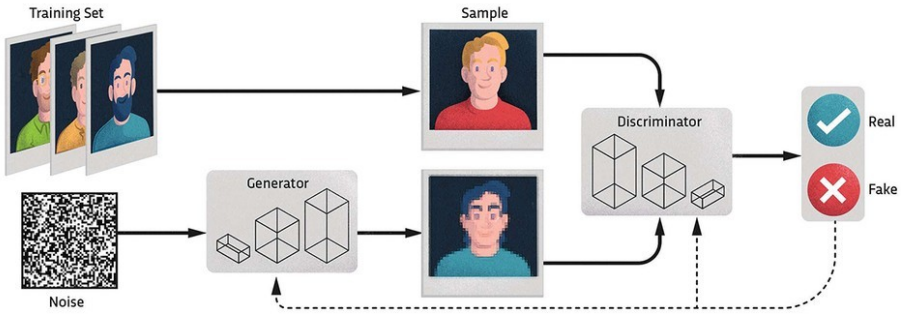


Figure 7.1: *GAN architeecture showing a generator creating an artificial image from noise and the discriminator trying to figure out if it is a real image or not.[**?**]*

7.1.1 Generative Adversarial Networks

Generative Adversarial Networks (GANs) [43] consist of two neural networks: the generator and the discriminator, that engage in a zero-sum game. The generator's role is to create data that is indistinguishable from genuine data, while the discriminator's role is to detect whether the data it receives is real (from the dataset) or fake (created by the generator).

Generator: The generator starts with a random noise signal and transforms this signal into data instances that resemble the real data as closely as possible. This transformation is guided by backpropagation signals derived from the discriminator's assessments, effectively teaching the generator to improve its data generation capabilities.

Discriminator: On the other side, the discriminator evaluates the authenticity of each data instance it receives, classifying it as either real or fake. This model is trained on a dataset of genuine data to establish a baseline of authenticity, which it uses to benchmark the synthetic outputs of the generator. See figure 7.1

It is formulated into the following optimization problem:

$$\text{mit}_G \max_D V(D,G) = \mathbb{E}_{x-H\,\text{man}}(x)[\log D(x)] + \mathbb{E}_{z-\beta_z(z)}[\log(1-D(G(z)))] \tag{7.1}$$

The neural networks in the GAN are directly trained from data and so do not need explicit initial assumptions. The application of GANs to finance is still relatively new [67–69]. A significant issue in training GANs is the problem of vanishing gradients. This occurs when the discriminator becomes too effective compared to the generator, leading to a saturated loss function that provides weak gradients for optimization. Another issue is mode collapse, where the diversity of the generated data diminishes. Imagine an artist who finds that landscapes are more popular

than portraits and begins to only paint landscapes, ignoring other subjects. This leads to a lack of diversity in the artist's work, similar to how a GAN might focus on generating only one type of output, neglecting the breadth of the original data distribution. To address these issues, advancements such as the Wasserstein GAN, further refined by incorporating a gradient penalty, have markedly enhanced GAN stability [47].

Wasserstein GAN

There are a variety of distances one can use to compare two distributions P_r, P_g $\in Prob(X)$. Arjovsky found in his paper on Wasserstein [47] that the Earth-Mover(EM) distance or Wasserstein-1 distance is the best choice. In statistics, the earth mover's distance (EMD) is a measure of the distance between two probability distributions over a region D. If the distributions are interpreted as two different ways of piling up a certain amount of dirt over the region D, the EMD is the minimum cost of turning one pile into the other; where the cost is assumed to be the amount of dirt moved times the distance by which it is moved. Calculating the EMD is in itself an optimization problem. There are infinitely many ways to move the earth around, and we need to find the optimal one. The EMD is widely used in content-based image retrieval to compute distances between the color histograms of two digital images. In this case, the region is the RGB color cube, and each image pixel is a parcel of "dirt". In mathematics, EMD is known as Wasserstein metric. The Wasserstein or Kantorovich-Rubinstein metric or distance is a distance function defined between probability distributions on a given metric space M. The Earth-Mover (EM) distance or Wasserstein-1 is the following:

$$W\left(\mathbb{P}_r, \mathbb{P}_g\right) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|] \tag{7.2}$$

where $\Pi\left(\mathrm{P}_r, \mathbb{P}_g\right)$ is the set of all joint distributions $\gamma(x, y)$ whose marginals are respectively $\mathbb{P}_r$ and $\mathbb{P}_g$. Intuitively, $\gamma(x, y)$ indicates how much "mass" must be transported from x to y in order to transform the distributions $\mathbb{P}_\mathrm{r}$ into the distribution $\mathbb{P}_g$. The EM distance then is the "cost" of the optimal transport plan. [47] A practical approximation of optimizing the EM distance is by using the Kantorovich-Rubinstein duality:

$$W_1\left(\mathbb{P}_r, \mathbb{P}_g\right) = \sup_{\|f\|_L \leq 1} \mathop{\mathbb{E}}_{x\sim\mathbb{P}_r}[f(x)] - \mathop{\mathbb{E}}_{\tilde{x}\sim\mathbb{P}_g}[f(\tilde{x})] \tag{7.3}$$

where $\sup_{\|f\|_L \leq 1}$ is the supremum over all the 1-Lipschitz functions. In practical settings involving neural networks as critics, this distance is approximated by maximizing the expected difference between evaluations on real versus generated

data. Although it is ambitious to assume that these networks can achieve the theoretical maximum, they often provide a useful approximation of the Wasserstein-1 distance. Enhancing the training of GANs involves adjusting the generator to minimize this distance, moving the generated distribution closer to the real one. The challenge lies in enforcing the 1-Lipschitz condition effectively; initial methods like weight clipping proved suboptimal, later giving way to gradient penalty techniques that encourage this condition more naturally and effectively in the loss function formulation. This gradient penalty is an added term to our loss function and enforces the 1-Lipschitz condition. With gradient penalty, our loss function becomes

$$L = \underset{\tilde{x}\sim\mathbb{P}_g}{\mathbb{E}}[D(\tilde{x})] - \underset{x\sim\mathbb{P}_r}{\mathbb{E}}[D(x)] + \lambda \underset{\hat{x}\sim\mathbb{P}_{\hat{x}}}{\mathbb{E}}\left[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2\right] \tag{7.4}$$

where λ is a scaling parameter for our gradient penalty, $\|\|_2$ indicates the Euclidean norm and $\hat{x}$ is a random convex combination of x and $\tilde{x}$ given by $\hat{x} = \epsilon x + (1-\epsilon)\tilde{x}$ with $\epsilon \sim U[0,1]$.

7.2 Community Detection in Correlation Matrices

In financial markets, assets often move together forming communities or clusters, which are not easily told apart through simple correlation measures. Detecting these communities can reveal insights into market structure and systemic risks. Techniques such as those proposed by MacMahon and Garlaschelli involve using methods from network theory to detect communities within correlation matrices of stock returns [70]. We can see this when we calculate the eigenvalues of the correlation matrix of, for instance, all the companies in the S&P500. If we plot the density of the eigenvalues against the eigenvalue value, we get figure 7.2. As random matrix theory would predict, we expect the eigenvalue density of the data to be close to the Marcenko-Pastur prediction, which we can see when we shuffle the data. There are, however, higher eigenvalues than we would theoretically expect. This is a result of the community behaviour of financial data. The largest eigenvalue found at around $\lambda = 170$ represents the "market mode". This "market mode" represents the communal movement of the whole S&P500. The market tends to trend together in, for instance, financial crises like the 2008 financial crisis. The other higher valued eigenvalues represent certain groups within the whole S&P500 index. Groups like energy companies or tech companies also tend to have correlated price movements, which we see represented in the data by these higher than theoretically expected eigenvalues. This is an important property of

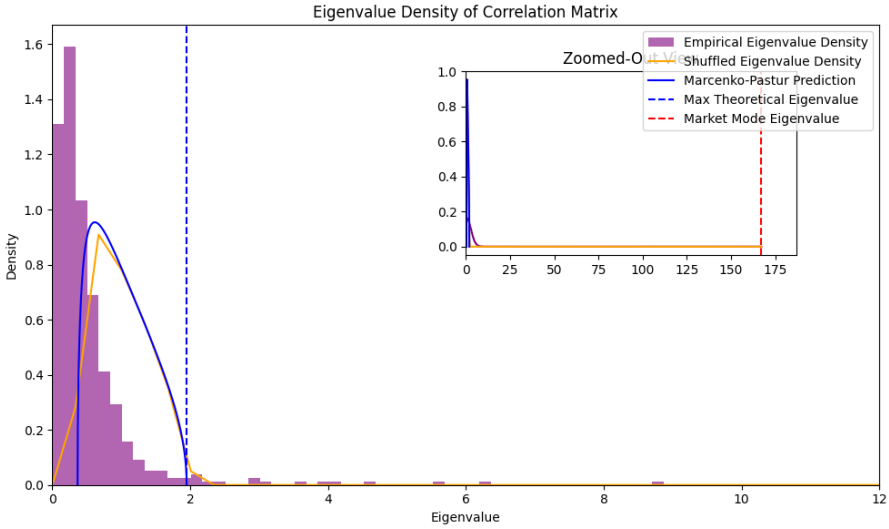


Figure 7.2: *The eigenvalue density of the empirical correlation matrix for $N = 445$ stocks of the S&P500 index (purple) and the Marcenko-Pastur prediction for a random correlation matrix with the same values of N and T (blue). The orange plot is the eigenvalue density obtained by randomly shuffling the empirical increments within each of the observed time series. The inset is the fully zoomed-out version of the plot, showing that the empirical correlation matrix has a maximum eigenvalue of about 170 as well as a handful of other leading eigenvalues above the predicted maximum value.*

financial time series because, for instance, it allows investors to hedge their risk by investing in non-correlated markets.

7.3 Full State CZ gate Results comparison to MPS

Here we compare the MPS results with the full-state results obtained by Ref. [60] in Figure 7.3 with the CZ gate active .

Again we see that Ref. [60] their simulations converge more rapidly than the MPS-based simulations, taking less epochs to minimize the Wasserstein loss than the MPS-based simulations. In table 7.1 we quantitatively compare the MPS-based simulation of $L = 18, \chi = 32$ the full state simulation from [60] We can see that

Table 7.1: *Comparison of lowest loss values between MPS and Full state simulations.*

Loss Metric	**MPS**	**Full state, CZ**
L_{Wass}	0.0002914	0.0002720
$L_{\text{ACF, non-abs}}$	0.0004479	0.0015201
$L_{\text{ACF, abs}}$	0.3091965	0.1532742
L_{Leverage}	0.0219054	0.0046997

the full-state approach is better at capturing the leverage effect but performs worse in absolute autocorrelation. When comparing the full state results from Ref. [60] with the original CZ gate (Figure 7.3 to the modified CNOT gate version (Figure 4.12 we see some major developments. The absolute autocorrelation has drastically improved with the CNOT gate version as compared to the CZ gate version. This result may illustrate that circuit design in gate type might also be a very important factor in the result quality next to the layer amount.

Version of November 21, 2025– Created November 21, 2025 - 12:28

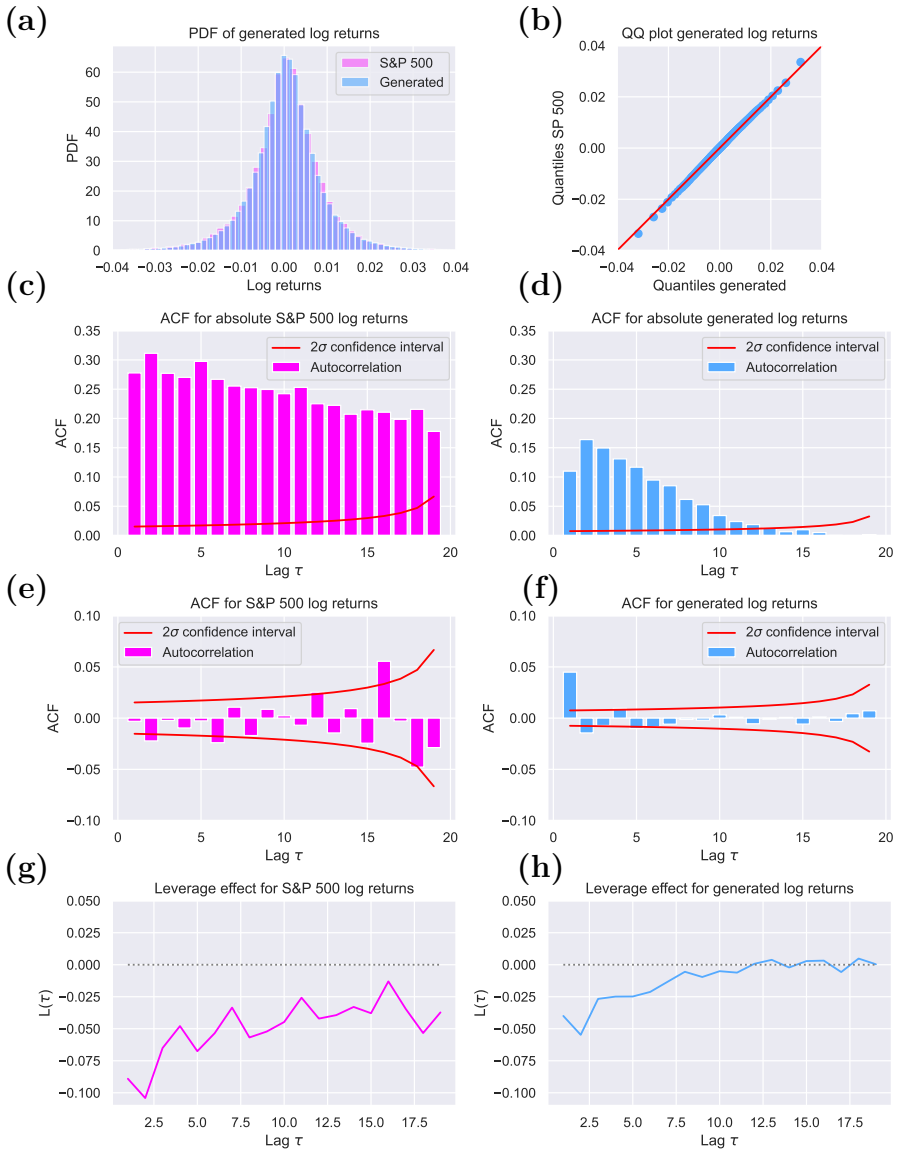


Figure 7.3: *Stylized facts for the S&P500 and the full-state simulation with the CZ gate, compared to the metrics of the S&P 500 index (left side). In* **(a)**, *we plot the probability density functions and in* **(b)** *the quantile-quantile plot of both the S&P 500 index and the generated time series. In* **(c)**-**(h)**, *we plot the metrics absolute autocorrelation, linear autocorrelation and the leverage effect, as an indication of the stylized facts as described in Section 2.1. The Subfigures* **(c)**, **(e)** *and* **(g)** *show the metrics of the S&P 500 index and the Subfigures* **(d)**, **(f)** *and* **(h)** *the metrics of the generated time series, respectively.*

7.4 Best and Worst Performing models

Table 7.2: *Top 5 configurations by final loss*

Rank	χ	Layers	Final	Min	Improvement%
1	24	18	0.000282	0.000207	95.6
2	24	10	0.000343	0.000259	94.7
3	32	18	0.000361	0.000307	93.5
4	8	10	0.000371	0.000247	94.6
5	32	10	0.000422	0.000288	92.8

Table 7.3: *Worst 5 configurations by final loss*

Rank	χ	Layers	Final	Min	Improvement%
1	24	1	0.002286	0.002161	48.5
2	8	1	0.002286	0.002161	48.5
3	32	1	0.002286	0.002161	48.5
4	1	5	0.002352	0.001711	62.7
5	1	18	0.005211	0.005192	21.1

7.5 Filtered model results

Table 7.4: *Final loss values (average of last 4 measurements) for different configurations*

Layers	χ=8	χ=16	χ=24	χ=32
5	0.000598	—	—	0.000467
10	0.000371	0.000432	0.000343	0.000422
18	0.000821	—	0.000282	0.000361

Version of November 21, 2025– Created November 21, 2025 - 12:28

Table 7.5: *Impact of bond dimension on loss (averaged across all layers)*

χ	Avg Loss	Std Dev	Min	Max	Count
8	0.000596	0.000184	0.000371	0.000821	3
16	0.000432	0.000000	0.000432	0.000432	1
24	0.000313	0.000031	0.000282	0.000343	2
32	0.000417	0.000044	0.000361	0.000467	3

Table 7.6: *Impact of number of layers on loss (averaged across all bond dimensions)*

Layers	Avg Loss	Std Dev	Min	Max	Count
5	0.000533	0.000065	0.000467	0.000598	2
10	0.000392	0.000037	0.000343	0.000432	4
18	0.000488	0.000238	0.000282	0.000821	3

Table 7.7: *Top 5 configurations by final loss*

Rank	χ	Layers	Final	Min	Improvement%
1	24	18	0.000282	0.000207	95.6
2	24	10	0.000343	0.000259	94.7
3	32	18	0.000361	0.000307	93.5
4	8	10	0.000371	0.000247	94.6
5	32	10	0.000422	0.000288	92.8

Table 7.8: *Worst 5 configurations by final loss*

Rank	χ	Layers	Final	Min	Improvement%
1	32	10	0.000422	0.000288	92.8
2	16	10	0.000432	0.000337	93.2
3	32	5	0.000467	0.000307	92.8
4	8	5	0.000598	0.000403	89.9
5	8	18	0.000821	0.000590	89.0

Acknowledgements

I would like to express my sincere gratitude to all those who supported me during the course of this thesis. First and foremost, I am very thankful to my PhD supervisor, David Dechant, whose daily guidance and willingness to put his own work aside and give me time to figure out my questions made an enormous difference. He helped me stay on track and move forward.

I am also very grateful to Jordi Tura for the valuable advice and knowledge he shared during our biweekly meetings. These discussions were really helpful in giving direction to the thesis and helped me understand the subject a lot better. My co-supervisor Vedran Dunjko, whose insightful comments greatly improved the quality of the thesis and challenged me to think more critically. (And for the quote which I will remember for the rest of my professional career in finance.)

A special thanks goes to Patrick Emonts, for his enormous help with the coding aspects of this project and for repeatedly pulling me out of deep technical problems. His practical expertise and readiness to step in were a real lifesaver.

Finally, I want to acknowledge the broader research environment at Leiden University and all the colleagues of the AQA group who were always ready to help. Also, the ALICE computing cluster, which provided the resources that made this work possible.

Bibliography

[1] Michael Lewis. *Liar's poker*. WW Norton & Company, 2010.

[2] Christina Stead. *House of All Nations*. Peter Davies, London, 1938.

[3] John Y. Campbell, Andrew W. Lo, and A. Craig MacKinlay. *The Econometrics of Financial Markets*. Princeton University Press, 1997.

[4] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[5] Ruey S Tsay. Analysis of financial time series. *John Eiley and Sons*, 2005.

[6] Omer Berat Sezer, Mehmet Ugur Gudelek, and Ahmet Murat Ozbayoglu. Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. *Applied soft computing*, 90:106181, 2020.

[7] Martin Sewell. Characterization of financial time series. *Rn*, 11(01):01, 2011.

[8] Young Shin Kim, Svetlozar T Rachev, Michele Leonardo Bianchi, Ivan Mitov, and Frank J Fabozzi. Time series analysis for financial market meltdowns. *Journal of Banking & Finance*, 35(8):1879–1891, 2011.

[9] Oana Peia and Kasper Roszbach. Finance and growth: time series evidence on causality. *Journal of Financial Stability*, 19:105–118, 2015.

[10] George Udny Yule. On a method of investigating periodicities in disturbed series with special reference to wolferâs sunspot numbers. *Statistical Papers of George Udny Yule*, pages 389–420, 1971.

[11] Robert F Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica: Journal of the econometric society*, pages 987–1007, 1982.

[12] Tim Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of econometrics*, 31(3):307–327, 1986.

[13] Robert Engle. Dynamic conditional correlation: A simple class of multivariate generalized autoregressive conditional heteroskedasticity models. *Journal of business & economic statistics*, 20(3):339–350, 2002.

[14] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[15] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[16] F. E. H. Tay and L. Cao. Application of support vector machines in financial time series forecasting. *Omega*, 29(4):309–317, 2001.

[17] Thomas H. Davenport and Jeanne G. Harris. *Competing on Analytics: The New Science of Winning*. Harvard Business School Press, 2007.

[18] J. Zhang, Y. Zheng, C. Zhang, and W. Qi. Stock price prediction of s&p 500 via combination of cnn and lstm. In *Proceedings of the International Conference on Neural Information Processing (ICONIP)*. Springer, 2017.

[19] Matthew F. Dixon, Diego Klabjan, and J. H. Bang. Financial time series forecasting with deep learning: A systematic review. *International Journal of Forecasting*, 36(2):610–640, 2020.

[20] Wei Zhang, Yuyi Xie, and Yumin Wang. A survey on deep learning in financial markets. *arXiv preprint arXiv:1806.03805*, 2018.

[21] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6-7):467–488, 1982.

[22] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2(79):79, 2018.

[23] F. Arute, K. Arya, R. Babbush, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[24] Simon J. Devitt, William J. Munro, and Kae Nemoto. Quantum error correction for beginners. *Reports on Progress in Physics*, 76(7):076001, 2013.

[25] Emmanuel E Haven. A discussion on embedding the black–scholes option pricing model in a quantum physics setting. *Physica A: Statistical Mechanics and its Applications*, 304(3-4):507–524, 2002.

[26] Belal E Baaquie. *Quantum finance: Path integrals and Hamiltonians for options and interest rates*. Cambridge University Press, 2007.

[27] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of political economy*, 81(3):637–654, 1973.

[28] Robert C Merton. Theory of rational option pricing. *The Bell Journal of economics and management science*, pages 141–183, 1973.

[29] Ashley Montanaro. Quantum speedup of monte carlo methods. *Proceedings of the Royal Society A*, 471(2181):20150301, 2015.

[30] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. Quantum support vector machine for credit risk analysis. *Physical Review A*, 98(2):022321, 2018.

[31] Stefan Woerner and Daniel J. Egger. Quantum risk analysis. In *Proceedings of the ACM International Conference on Computing Frontiers*, pages 47–56. ACM, 2019.

[32] Samuel L. Braunstein and Peter van Loock. Quantum information with continuous variables. *Rev. Mod. Phys.*, 77:513–577, 2005.

[33] Rama Cont. Empirical properties of asset returns: stylized facts and statistical issues. *Quantitative finance*, 1(2):223, 2001.

[34] Zhaohua Ding, Clive W.J. Granger, and Robert F. Engle. A long memory property of stock market returns and a new model. *Journal of Empirical Finance*, 1(1):83–106, 1993.

[35] Zhaohua Ding and Clive W.J. Granger. Granger causality in long-memory processes. *Journal of Time Series Analysis*, 17(6):107–124, 1996.

[36] T Qiu, B Zheng, F Ren, and S Trimper. Return-volatility correlation in financial dynamics. *Physical Review EâStatistical, Nonlinear, and Soft Matter Physics*, 73(6):065103, 2006.

[37] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[38] Paul Adrien Maurice Dirac. A new notation for quantum mechanics. In *Mathematical proceedings of the Cambridge philosophical society*, volume 35, pages 416–418. Cambridge University Press, 1939.

[39] Max Born. Zur Quantenmechanik der stoÃvorgÃ€nge. *Zeitschrift fÃŒr Physik*, 37(12):863–867, 1926.

[40] Carl Pomerance. A tale of two sieves. *Notices of the American Mathematical Society*, 43(12):1473–1485, December 1996.

[41] Peter W. Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 124–134. IEEE Computer Society, 1994.

[42] Brian Coyle, Maxwell Henderson, Justin Chan Jin Le, Niraj Kumar, Marco Paini, and Elham Kashefi. Quantum versus classical generative modelling in finance. *Quantum Science and Technology*, 6(2):024013, 2021.

[43] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[44] //www.pilot44.com/insights/the-evolution-of-generative-ai.

[45] Seth Lloyd and Christian Weedbrook. Quantum generative adversarial learning. *Physical Review Letters*, 121(4):040502, 2018.

[46] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, volume 27. NeurIPS, 2014.

[47] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[48] M. De Angelis and A. Gray. Why the 1-wasserstein distance is the area between the two marginal cdfs, 2021. Page 1.

[49] Peiqi Wang, Dongsheng Wang, Yu Ji, Xinfeng Xie, Haoxuan Song, XuXin Liu, Yongqiang Lyu, and Yuan Xie. Qgan: Quantized generative adversarial networks. *arXiv preprint arXiv:1901.08263*, 2019.

[50] A. P'erez-Salinas, A. Cervera-Lierta, E. Gil-Fuster, and J. I. Latorre. Data re-uploading for a universal quantum classifier. *Quantum*, 4:226, 2020.

[51] Alice Barthe, Michele Grossi, Sofia Vallecorsa, Jordi Tura, and Vedran Dunjko. Parameterized quantum circuits as universal generative models for continuous multivariate distributions. *arXiv preprint arXiv:2402.09848*, 2024.

[52] Jinfu Chen Jordi Tura Patrick Emonts, Kshiti Sneh Rai. *Lecture notes Tensor Networks*. Leiden University, 2024.

[53] Ulrich Schollwöck. The density-matrix renormalization group in the age of matrix product states. *Annals of Physics*, 326(1):96–192, 2011.

[54] Dmitri Burago, Yuri Burago, and Sergei Ivanov. *A Course in Metric Geometry*, volume 33 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 2001.

[55] Mikhail Gromov. *Metric Structures for Riemannian and NonâRiemannian Spaces*. Modern BirkhÃ€user Classics. BirkhÃ€user Verlag, Boston; Basel; Berlin, reprint of the 2001 edition edition, 2007.

[56] Christopher A. Fuchs and Jeroen van de Graaf. Cryptographic distinguishability measures for quantum mechanical states. *IEEE Transactions on Information Theory*, 45(4):1216–1227, 1999. arXiv:quant-ph/9712042.

[57] Johnnie Gray. quimb: a python library for quantum information and many-body calculations. *Journal of Open Source Software*, 3(29):819, 2018.

[58] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[59] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed framework for emerging {AI} applications. In *13th USENIX symposium on operating systems design and implementation (OSDI 18)*, pages 561–577, 2018.

[60] Eliot Schwander. Quantum generative modelling for financial time series. Msc thesis.

[61] H.-M. Rieser, F. Köster, and A. P. Raulf. Tensor networks for quantum machine learning. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 479(2275):20230218, 2023.

[62] Tim Bollerslev, Ray Y Chou, and Kenneth F Kroner. Arch modeling in finance: A review of the theory and empirical evidence. *Journal of econometrics*, 52(1-2):5–59, 1992.

[63] Christian Francq and Jean-Michel Zakoian. *GARCH models: structure, statistical inference and financial applications*. John Wiley & Sons, 2019.

[64] Alec Radford. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[65] Chris Donahue, Julian McAuley, and Miller Puckette. Synthesizing audio with generative adversarial networks. *arXiv preprint arXiv:1802.04208*, 1, 2018.

[66] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

[67] Xingyu Zhou, Zhisong Pan, Guyu Hu, Siqi Tang, and Cheng Zhao. Stock market prediction on high-frequency data using generative adversarial nets. *Mathematical Problems in Engineering*, 2018(1):4907423, 2018.

[68] Adriano Koshiyama, Nick Firoozye, and Philip Treleaven. Generative adversarial networks for financial trading strategies fine-tuning and combination. *Quantitative Finance*, 21(5):797–813, 2021.

[69] Ugo Fiore, Alfredo De Santis, Francesca Perla, Paolo Zanetti, and Francesco Palmieri. Using generative adversarial networks for improving classification effectiveness in credit card fraud detection. *Information Sciences*, 479:448–455, 2019.

[70] Mel MacMahon and Diego Garlaschelli. Community detection for correlation matrices. *Physical Review X*, 5(2), April 2015.