Synthetic Financial Time Series Generation with GANs: A Comparative Study and the Proposed P-RSQGAN

by

Yixin Xia (s230026171@mail.bnbu.edu.cn)
Manxi Tao (s230026145@mail.bnbu.edu.cn)

A Final Year Project Thesis (COMP4004; 3 Credits)
submitted in partial fulfillment of the requirements
for the degree of

Bachelor of Science (Honours)
in
Computer Science and Technology
at

Beijing Normal – Hong Kong Baptist University
December, 2025

Abstract

Synthetic financial time series generation has become an important tool for risk analysis, stress testing, and model development in data-constrained financial environments. However, financial returns exhibit complex characteristics, including heavy-tailed distributions, volatility clustering, and regime-dependent dynamics, which pose significant challenges for generative modeling.

This thesis presents a systematic study of Generative Adversarial Network (GAN) based models for financial time series generation. We conduct a unified empirical comparison of four representative baseline models—LSTM-GAN, TimeGAN, WGAN-GP, and QuantGAN—under a consistent experimental protocol. Model performance is evaluated using a multi-view framework that examines distributional fidelity, temporal dependence, and path-wise realism through diagnostics such as empirical density plots, Q–Q plots, autocorrelation analysis, rolling volatility patterns, and fan charts.

Our results show that while baseline models can reproduce global statistical properties of returns to a reasonable extent, they remain fundamentally unconditional. As a consequence, volatility dynamics and tail behaviour are learned implicitly and cannot be controlled at sampling time, limiting their usefulness for scenario-based analysis.

To address this limitation, we propose Projection Regime-Specific QuantGAN (P-RSQGAN), a conditional extension of QuantGAN that enables explicit regime-controlled generation. By introducing regime conditioning in the generator and a projection-based critic within a WGAN-GP framework, P-RSQGAN achieves improved regime separability while preserving distributional fidelity. Empirical results demonstrate that P-RSQGAN provides effective control over volatility regimes and exhibits strong regime consistency, without sacrificing overall realism.

Overall, this work highlights both the strengths and limitations of existing GAN-based generators for financial time series, and demonstrates that incorporating explicit regime awareness is a promising direction for controllable and interpretable synthetic data generation in finance.

Keywords: synthetic financial data; time series generation; GANs; WGAN-GP; QuantGAN; regime conditioning; volatility clustering; P-RSQGAN

Contents

Abstract **I**

1 Introduction **1**

2 Data Generation with GANs **2**

2.1 Synthetic Data Generation 2

2.2 Generative Adversarial Networks 3

2.3 Time-Series and Finance-Oriented GAN Variants 5

2.4 Evaluating GAN Outputs 6

3 Data and Experimental Setup **7**

3.1 Datasets 8

3.2 Preprocessing and window construction 8

3.3 Evaluation protocol 8

4 Methodology **9**

4.1 Baselines 9

4.1.1 LSTM-GAN 9

4.1.2 TimeGAN 10

4.1.3 WGAN-GP 11

4.1.4 QuantGAN 13

4.2 Proposed Model: P-RSQGAN 15

4.2.1 Regime definition and label construction 16

4.2.2 Model architecture 16

5 Evaluation and Comparision **18**

5.1 Evaluation Tools 18

5.2 Baseline Comparison 21

5.3 Compare P-RSQGAN with QuantGAN 26

6 Conclusion **28**

References **31**

1. Introduction

Synthetic financial time series generation has become increasingly important in quantitative finance, with applications ranging from risk management and stress testing to scenario analysis and algorithmic trading [1][2]. In practice, access to high-quality financial data is often constrained by privacy concerns, regulatory restrictions, and the limited availability of extreme market events. These limitations have motivated the development of data-driven generative models that aim to produce artificial return series while preserving essential statistical and temporal properties of real markets [3][4].

Financial time series exhibit several distinctive characteristics that complicate generative modelling. Return distributions are typically heavy-tailed and non-Gaussian, linear autocorrelation in returns is weak, and volatility displays strong clustering over time [2][5][6]. In addition, market dynamics are often non-stationary and may undergo abrupt regime shifts [7]. As a result, realistic data generation requires more than marginal distribution matching; temporal dependence and volatility dynamics must also be reproduced in a coherent manner.

Recent advances in deep generative models, particularly Generative Adversarial Networks (GANs), offer flexible, non-parametric approaches to learning complex data distributions [8]. GAN-based models have been extended to sequential settings through recurrent architectures and convolutional temporal models [9][10]. In the financial domain, specialised frameworks such as QuantGAN demonstrate improved capacity to capture long-range dependence in return series through temporal convolutional structures [3]. Despite these advances, training instability and mode collapse remain persistent challenges in GAN-based models, and performance is highly sensitive to architectural and optimisation choices [11][12][13][14].

In recent years, GAN-based research for synthetic financial time series has expanded beyond early unconditional generators toward richer sequence models for long-range dependence, risk- and application-aligned training objectives, and more systematic diagnostics. Attention-augmented and Transformer-style GANs have been proposed to better reproduce stylised facts and long-horizon dependence, and were empirically tested on market data such as the S&P 500 [15]. In parallel, Tail-GAN introduces a tail-focused formulation for scenario generation that targets risk-sensitive behaviour, such as extreme market events, rather than only matching the centre of the distribution [16]. Recent work also emphasises the importance of evaluation and benchmarking: comparative studies review how different deep generative families behave under Value-at-Risk oriented settings [17], while controlled experiments show that the ability to reproduce multiple stylised facts varies substantially across GAN architectures and training choices [18].

Evaluation further complicates the assessment of generative models for financial time series. Unlike image generation, there is no single dominant metric that captures realism. Model quality is instead assessed through a combination of diagnostics, including distributional analyses, temporal dependence measures, and path-wise simulations [2][1]. A robust evaluation framework must therefore adopt a multi-dimensional perspective, with different diagnostics capturing complementary aspects of realism.

This thesis addresses these challenges through two research questions. First, how well do representative GAN-based models reproduce the distributional and temporal properties of financial return series under a unified evaluation framework? Second, can explicit regime conditioning enable controllable generation while preserving statistical fidelity and training stability?

The contributions of this work are threefold. We provide a systematic comparison of representative baseline models, including LSTM-GAN, TimeGAN, WGAN-GP, and QuantGAN, using a unified experimental protocol and a multi-view evaluation framework. We identify common strengths and limitations shared by unconditional generators. Finally, we propose P-RSQGAN, a regime-aware extension that enables explicit control over volatility regimes while maintaining stable training and realistic dynamics.

The remainder of this thesis is organised as follows. Chapter 2 introduces background concepts and technical preliminaries. Chapter 3 reviews related work. Chapter 4 describes the datasets and experimental setup. Chapter 5 presents the baseline models and the proposed method. Chapter 6 introduces the evaluation framework, followed by empirical results in Chapter 7. Chapter 8 concludes and discusses directions for future research.

2. Data Generation with GANs

This section introduces GAN-based data generation as the core paradigm used in our thesis. We first present the standard GAN formulation and explain why its adversarial training can be unstable for complex distributions such as financial returns. We then motivate stability-oriented objectives (WGAN and WGAN-GP) that provide smoother training signals. Building on these foundations, we review time-series and finance-oriented variants that incorporate sequence modelling and conditioning mechanisms, which are essential for capturing temporal dependence and regime-driven non-stationarity in financial markets. Finally, we discuss why evaluating synthetic financial time series requires a multi-view protocol rather than a single universal score.

2.1 Synthetic Data Generation

Early synthetic generation in finance is mostly model-driven. Monte Carlo simulation samples from an assumed price or return process and produces many paths for downstream tasks such as stress testing [1]. This approach is clear and controllable, but realism depends strongly on whether the assumed process matches real markets.

Econometric time-series models provide a more data-grounded option. ARIMA models describe linear dependence in the mean and are mainly used for forecasting rather than realistic return generation [7]. For volatility, ARCH/GARCH models explicitly model time-varying conditional variance and can reproduce volatility clustering [5, 6]. However, these models remain parametric. They may struggle when the data show strong non-linearity, regime shifts, or complex path-dependent behaviour.

Deep likelihood-based generative models such as VAEs provide a flexible alternative [19]. By compressing the input data into a low dimensional representation in the encoder, pushing it into a latent space, and then reconstructing the input data from this compressed representation in the decoder, it may be trained to produce synthetic data. The architecture of VAE is shown in Figure 1. By maximizing the probability of the newly generated time series in relation to the input data, it seeks to provide an output that best captures the input. When generating the output, the objective is to maximize the possibility that the newly formed time series will closely match the input data. However, because of the noise the decoder adds, VAE often produces less varied samples [20]. In practice, for financial returns, simple likelihood choices can make samples look too smooth, which may reduce tail events unless the model is carefully designed.

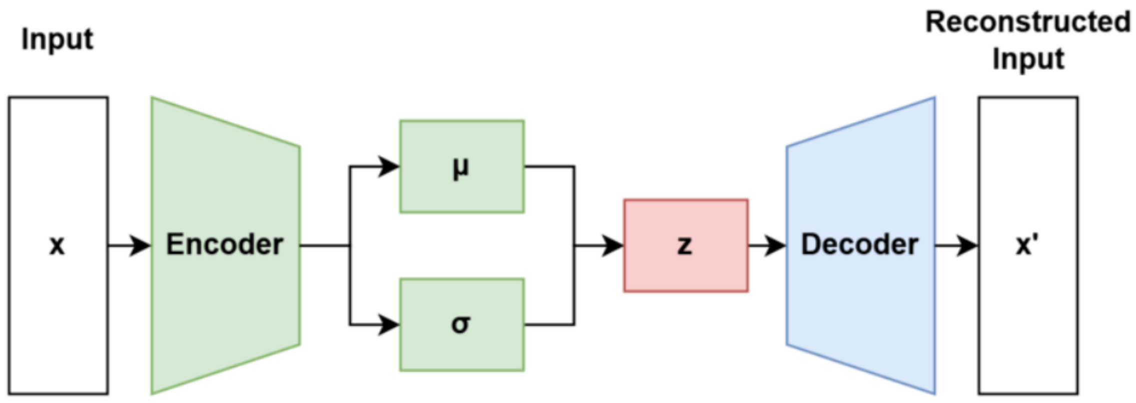


Figure 1: Architecture of VAE

2.2 Generative Adversarial Networks

Generative Adversarial Networks (GANs) consist of two neural networks that are trained together in a competitive setting shown as Figure 2: a generator G and a discriminator D [8]. The generator takes a random latent vector $z \sim p(z)$ (often a standard Gaussian or uniform distribution) and maps it to a synthetic sample $\tilde{x} = G(z)$. The discriminator receives either a real sample $x \sim P_r$ from the training data or a generated sample $\tilde{x} \sim P_g$, and outputs a scalar score $D(\cdot)$ that represents how likely the input is real. During training, the discriminator is updated to assign high scores to real data and low scores to generated data, while the generator is updated to produce samples that increase the discriminator's score. This creates an adversarial game in which the generator gradually learns to match the real data distribution. The architecture of GANs is shown in Figure 2. The original minimax objective is:

$$\min_{G} \max_{D} \; \mathbb{E}_{x \sim P_r}\big[\log D(x)\big] + \mathbb{E}_{z \sim p(z)}\big[\log\big(1 - D(G(z))\big)\big]. \tag{1}$$

Although powerful, vanilla GAN training can be unstable. When real and generated distributions lie on low-dimensional manifolds that do not align well, the discriminator can become near-perfect and generator gradients can vanish, leading to non-convergence. The same mismatch can also create incentives for mode collapse, where the generator outputs only a small subset of possible samples [11].

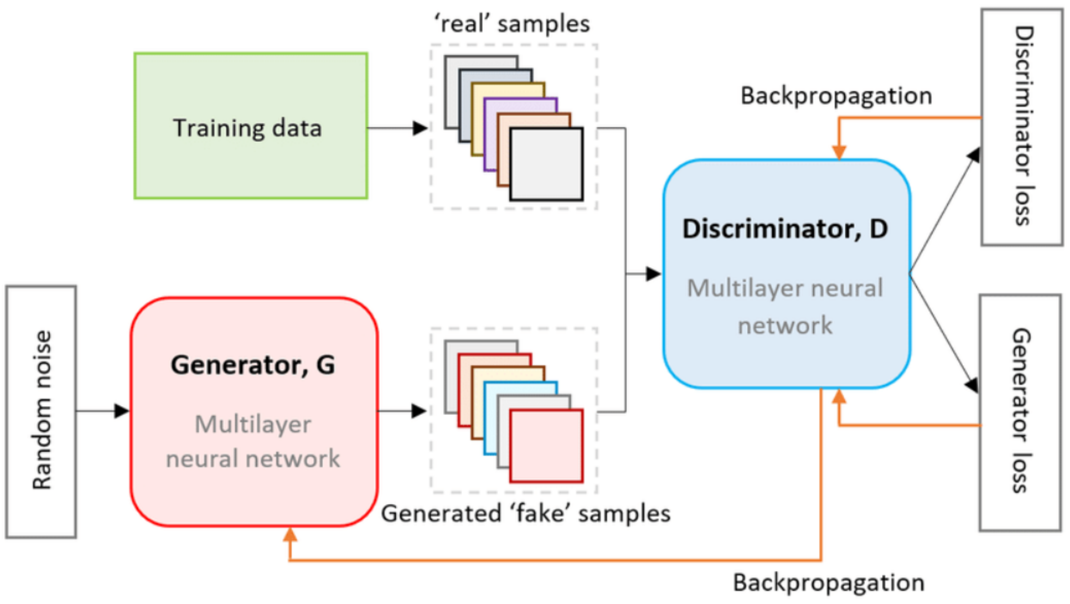


Figure 2: Architecture of GAN. Image and caption credit:[21]

To improve stability, WGAN replaces the discriminator with a critic and uses the Wasserstein-1 distance as the training signal [12]. The objective becomes:

$$\min_{G} \max_{D \in \mathcal{D}_1} \ \mathbb{E}_{x \sim P_r}\big[D(x)\big] - \mathbb{E}_{z \sim p(z)}\big[D(G(z))\big], \tag{2}$$

where $\mathcal{D}_1$ denotes the set of 1-Lipschitz functions. This change often provides smoother gradients and more stable behaviour during training. The original WGAN enforced the Lipschitz constraint by weight clipping, but this can hurt the critic's capacity and can still lead to optimisation issues [12].

WGAN-GP addresses this by adding a gradient penalty that encourages the critic's gradient norm to stay close to 1 on points interpolated between real and generated samples [**?**]:

$$\min_{G} \max_{D} \ \mathbb{E}_{x \sim P_r}\big[D(x)\big] - \mathbb{E}_{z \sim p(z)}\big[D(G(z))\big] - \lambda \, \mathbb{E}_{\hat{x} \sim P_{\hat{x}}}\Big[\big(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1\big)^2\Big]. \tag{3}$$

This modification is widely used because it improves training robustness and reduces sensitivity to weight clipping choices.

2.3 Time-Series and Finance-Oriented GAN Variants

Applying GANs to time series requires more than changing the loss. The generator must learn temporal structure, not only marginal distributions. One line of work introduces sequence-aware architectures and auxiliary objectives to preserve time dependence. A representative step toward sequence-aware learning is **TimeGAN**, which combines adversarial training with explicit mechanisms that preserve temporal dynamics [9]. TimeGAN introduces an embedding space, where an embedder (E) maps data into latent trajectories $h_{1:T} = E(x_{1:T})$, and a recovery network (R) reconstructs $\hat{x}_{1:T} = R(h_{1:T})$. This supports a reconstruction loss:

$$\mathcal{L}_{\text{rec}} = \mathbb{E}\big[\|x_{1:T} - \hat{x}_{1:T}\|_2\big]. \tag{4}$$

To better preserve transitions over time, TimeGAN adds a supervisor (S) that learns one-step dynamics in latent space, commonly expressed as a supervised loss of the form:

$$\mathcal{L}_{\text{sup}} = \mathbb{E}\left[\sum_{t=1}^{T-1} \|h_{t+1} - S(h_t)\|_2\right]. \tag{5}$$

Together with adversarial losses, these additional terms reduce the chance that the generator only matches marginal distributions while ignoring sequential structure. In other words, TimeGAN makes “temporal realism” part of the objective rather than a side effect.

Another line of work adopts architectures that are effective for long-range dependencies. **QuantGAN** uses **Temporal Convolutional Networks (TCNs)** with causal and dilated convolutions to achieve a large receptive field efficiently [3]. Instead of relying on recurrent updates, TCNs can capture long-range patterns using stacked dilated convolutions, which is useful for financial returns where volatility dependence may persist across many lags. In QuantGAN, both the generator and discriminator are typically implemented as TCNs, and training is often framed using a critic-based objective [12]. The architecture of TCN is shown in Figure 3. A Wasserstein-style sequence objective can be written as:

$$\min_{G} \max_{D \in \mathcal{D}_1} \; \mathbb{E}_{x_{1:T} \sim P_r}\big[D(x_{1:T})\big] - \mathbb{E}_{z \sim p(z)}\big[D(G(z))\big], \tag{6}$$

where $\mathcal{D}_1$ enforces the 1-Lipschitz constraint. In practice, this is frequently combined with the gradient penalty of WGAN-GP for stability [13], which is especially helpful when training deep sequence models.

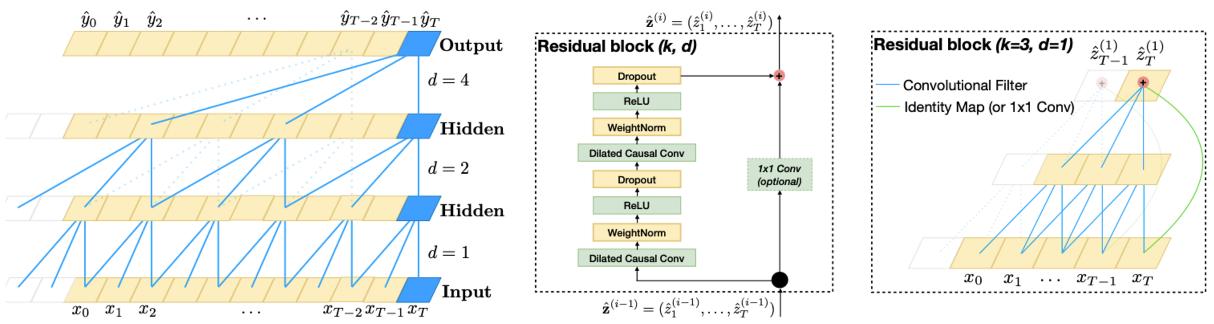


Figure 3: Architecture of TCN. Image and caption credit:[10]. (a) A dilated causal convolution with dilation factors d = 1, 2, 4 and filter size k = 3. The receptive field is capable of cover all values from the input sequence. (b) TCN residual block. An 1x1 convolution is added when residual input and output have different dimensions. (c) An example of residual connection in a TCN. The blue lines are filters in the residual function, and the green lines are identity mappings.

Beyond architecture, conditioning becomes important when financial data are non-stationary. Conditional GANs incorporate side information c (for example, labels, states, or regimes) into both generator and discriminator, so the model learns $p(x \mid c)$ rather than a single unconditional distribution [22]. In finance, a natural conditioning variable is the market regime, such as low-volatility versus crisis periods.

Within this framework, **Regime-Specific Quant GAN(RSQGAN)** extends the QuantGAN-style backbone by explicitly using market regimes as the conditioning signal [4]. It first identifies regimes (via segmentation) to produce labels c, then trains a conditional generator so that sampling can target a selected regime. In implementation terms, conditioning is typically injected through a regime embedding e_c that is concatenated or added to intermediate features, which architecture are show in Fig 4, for example:

$$h' = \mathrm{Concat}(h, e_c) \quad \text{or} \quad h' = h + We_c, \tag{7}$$

where h denotes a feature representation in the TCN stack.

RSQGAN also introduces a sampling-side control mechanism (z-clipping) to manage the fidelity–variety trade-off during generation [4]. A simple way to express this idea is to clip the latent noise:

$$z' = \mathrm{clip}(z, -\alpha, \alpha), \tag{8}$$

where smaller α restricts noise magnitude and can produce more conservative, less diverse samples, while larger α allows broader variation. Conceptually, this makes the generator not only regime-aware but also more user-controllable in how aggressively it explores the distribution.

2.4 Evaluating GAN Outputs

Evaluation is a major difference between image generation and financial time series generation. In images, metrics such as the Fréchet Inception Distance (FID) are widely used because

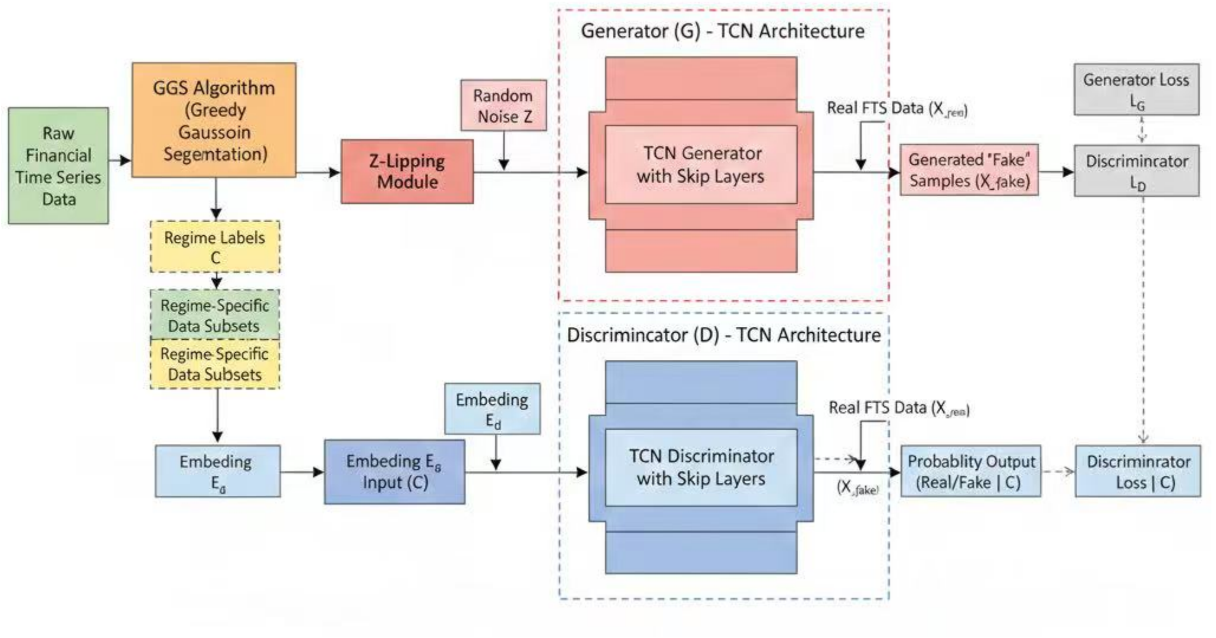

Raw Financial Time Series Data
GGS Algorithm (Greedy Gaussoin Segemtation)
Z-Lipping Module
Random Noise Z
Generator (G) - TCN Architecture
TCN Generator with Skip Layers
Real FTS Data (X_{real})
Generated "Fake" Samples (X_{fake})
Generator Loss L_G
Discriminrator L_D
Regime Labels C
Regime-Specific Data Subsets
Regime-Specific Data Subsets
Discrimincator (D) - TCN Architecture
Embeding E_d
Embeding E_d
Embeding E_6 Input (C)
TCN Discriminator with Skip Layers
Real FTS Data (X_{real})
(X_{fake})
Probablity Output (Real/Fake | C)
Discriminrator Loss | C)

Figure 4: Overview of RSQGAN training and conditional architecture

pretrained feature extractors provide stable representations for comparing real and generated distributions [23].

For financial time series, there is no similarly standard, widely accepted metric. A key reason is that good synthetic financial data must match several properties at once: distributional shape (including tails), temporal dependence, volatility clustering, and behaviour under regimes [2]. Many studies therefore rely on multi-view evaluation: distribution plots (PDF, Q–Q), dependence diagnostics (ACF, volatility clustering), and path-based summaries (e.g., fan charts). RSQGAN also highlights an important caution: if one focuses too narrowly on a small set of stylised-fact losses, the generator may exploit the metric and collapse to limited patterns, which undermines diversity [4]. This motivates evaluation protocols that jointly consider fidelity and variety, especially for crisis regimes.

3. Data and Experimental Setup

This section describes the data sources, preprocessing procedures, and experimental protocol used throughout the empirical analysis. We first introduce the financial datasets employed in this study and discuss their key characteristics. We then detail the construction of return sequences and the normalization strategy applied prior to model training. Finally, we outline the unified evaluation protocol adopted to ensure a fair and consistent comparison across all generative models considered.

An overview of the datasets and key experimental settings is provided in Table 1.

3.1 Datasets

This study considers two representative financial datasets: the NASDAQ Composite Index (NDX) and spot gold prices (AU9999). These assets exhibit distinct market characteristics. Equity index returns typically display higher volatility and stronger sensitivity to market risk sentiment, whereas gold often acts as a defensive asset with different volatility dynamics. Using both datasets allows us to assess model robustness across heterogeneous asset classes.

The NDX data are obtained from Yahoo Finance, which provides publicly accessible historical price series for major financial indices. Data acquisition is performed programmatically using the `pandas_datareader` library in Python, together with the `fix_yahoo_finance` patch, which enables stable access to Yahoo Finance data. Daily closing prices are downloaded via the `get_data_yahoo()` interface by specifying the index ticker and the time span of interest. This procedure ensures reproducibility and consistent integration into the experimental pipeline.

Gold price data are sourced from the RESSET Financial Research Database (`www.resset.com`), maintained by Beijing RESSET Data Technology Co., Ltd. Both datasets consist of daily closing prices and span multiple market cycles, including periods of heightened volatility. This temporal coverage enables the evaluation to reflect model behaviour under both stable and turbulent market conditions.

3.2 Preprocessing and window construction

Let S_t denote the asset price at time t. Log-returns are computed as

$$r_t = \log S_t - \log S_{t-1}. \tag{9}$$

The return series is segmented into overlapping sliding windows of fixed length T, yielding samples $x_i = r_{i:i+T-1}$. All models are trained on identical windowed datasets to ensure comparability.

To stabilise training, returns are normalised prior to model input. Normalisation statistics are computed using the training set and applied consistently to validation and test data. Generated samples are transformed back to the original scale using the corresponding inverse transformation for evaluation and visualisation.

3.3 Evaluation protocol

All experiments follow a unified evaluation protocol to ensure fair comparison across models. In particular, models are trained and evaluated on identical windowed datasets, with consistent sequence length, time-ordered training–testing splits, and sample generation sizes. Where stochastic components are involved, random seeds are fixed whenever applicable.

All reported results are obtained using held-out test data that are not accessed during train-

Table 1: Datasets and experimental settings

Item	**NDX**	**Gold**
Data frequency	Daily (closing price)	Daily (closing price)
Time span	2014/01/02 – 2024/12/30	2015/09/30 – 2025/09/30
Approx. number of observations	$\sim$2750	$\sim$2500
Return definition	Log-returns	
Window length T	128	
Normalisation	Z-score normalisation	
Train/test split	Time-ordered split	
Generated samples per model	Reported per experiment	

ing, thereby preventing information leakage. This protocol ensures that observed performance differences can be attributed to model architecture and training objectives, rather than to discrepancies in data handling or experimental configuration.

4. Methodology

This chapter present the four baseline GAN models and their objectives, followed by our proposed P-RSQGAN, which adds regime conditioning and a projection-based critic to enable controllable generation. For each model, we specify the network structure, loss functions, and the sampling procedure that produces synthetic return sequences for evaluation.

4.1 Baselines

This subsection introduces four baseline models with varying architectural and training characteristics. For models with standard and well-established training procedures, such as LSTM-GAN and TimeGAN, we focus on architectural descriptions and objective functions without providing explicit algorithm listings. In contrast, for models whose training involves non-trivial schedules or stability-oriented components, including WGAN-GP and QuantGAN, we provide explicit algorithms to clarify the optimisation process.

4.1.1 LSTM-GAN

LSTM-GAN serves as a recurrent GAN baseline for sequential data generation. Both the generator and the discriminator are implemented using Long Short-Term Memory (LSTM) networks, which enable the model to capture temporal dependencies through recurrent state updates.

The generator maps a sequence of latent noise vectors to a synthetic return sequence via an LSTM followed by a linear projection. The discriminator processes an input return sequence using an LSTM and outputs a scalar score indicating whether the sequence is real or generated.

LSTM-GAN is trained using the standard GAN objective with a binary cross-entropy loss. Despite its simplicity, LSTM-GAN is commonly used as a baseline for sequential generation.

Generator (G). The generator takes a sequence of latent noise vectors $z_{1:T} \in \mathbb{R}^{T \times d_z}$ sampled from a standard normal distribution and maps it to a synthetic return sequence $\tilde{x}_{1:T} = G(z_{1:T}) \in \mathbb{R}^{T \times 1}$. Concretely, the noise sequence is fed into an LSTM, followed by a linear projection at each time step:

$$\tilde{x}_{1:T} = f\big(W\, \mathrm{LSTM}(z_{1:T}) + b\big), \tag{10}$$

where $f(\cdot)$ denotes a bounded activation function (e.g., tanh) applied to match the normalized data range.

Discriminator (D). The discriminator receives a full return sequence $x_{1:T} \in \mathbb{R}^{T \times 1}$ (real or generated). The sequence is encoded by an LSTM, and the final hidden state h_T is used as a sequence-level representation. A linear layer followed by a sigmoid activation maps h_T to a scalar probability:

$$D(x_{1:T}) = \sigma\big(w^\top h_T + b\big). \tag{11}$$

This design encourages the discriminator to evaluate realism at the sequence level rather than on individual time steps.

Training objective. LSTM-GAN is trained using the classic GAN objective with binary cross-entropy loss. The discriminator loss is

$$\mathcal{L}_D = -\mathbb{E}_{x \sim P_r}[\log D(x)] - \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))], \tag{12}$$

and the generator loss is

$$\mathcal{L}_G = -\mathbb{E}_{z \sim p(z)}[\log D(G(z))]. \tag{13}$$

Training procedure. The discriminator is updated multiple times per iteration to maintain discrimination capability, followed by a generator update. While conceptually simple, this baseline is prone to training instability and limited long-range dependency modelling, making it a useful reference point for more stable and expressive architectures.

4.1.2 TimeGAN

TimeGAN extends the GAN framework for time-series generation by introducing an explicit latent embedding space and supervised temporal constraints. The model consists of five components: an embedder, a recovery network, a generator, a supervisor, and a discriminator.

The embedder and recovery networks learn a latent representation that preserves temporal structure, while the supervisor enforces step-wise temporal consistency in the latent space. Adversarial training is conducted jointly with reconstruction and supervised losses to align generated sequences with real data dynamics.

This architecture enables improved temporal coherence compared to purely adversarial sequence models, at the cost of increased model complexity and training sensitivity.

Model components. TimeGAN consists of five neural networks: an embedder E, a recovery network R, a generator G, a supervisor S, and a discriminator D. The embedder maps real sequences to latent trajectories, $h_{1:T} = E(x_{1:T})$, while the recovery network reconstructs the data, $\hat{x}_{1:T} = R(h_{1:T})$. The generator produces synthetic latent trajectories $\tilde{h}_{1:T} = G(z_{1:T})$, and the supervisor models one-step temporal dynamics in the latent space.

Reconstruction and supervised objectives. To ensure that the latent space preserves information from the original data, a reconstruction loss is defined as:

$$\mathcal{L}_{\text{rec}} = \mathbb{E}\left[\left\|x_{1:T} - R(E(x_{1:T}))\right\|_2\right]. \tag{14}$$

To explicitly enforce temporal consistency, TimeGAN introduces a supervised loss on latent transitions:

$$\mathcal{L}_{\text{sup}} = \mathbb{E}\left[\sum_{t=1}^{T-1} \left\|h_{t+1} - S(h_t)\right\|_2\right]. \tag{15}$$

Adversarial objective. Adversarial training is performed in the latent space. The discriminator D aims to distinguish real latent trajectories $h_{1:T}$ from generated ones $\tilde{h}_{1:T}$, while the generator seeks to fool the discriminator by aligning the two distributions. The adversarial losses follow the standard GAN formulation.

Training strategy. Training proceeds in stages. First, the embedder and recovery networks are optimized using the reconstruction loss. Second, the supervisor is trained to minimize the supervised loss. Finally, all components are jointly trained using a combination of reconstruction, supervised, and adversarial objectives. This design improves temporal coherence relative to vanilla GANs but increases model complexity and sensitivity to training hyperparameters.

4.1.3 WGAN-GP

Wasserstein GAN with gradient penalty (WGAN-GP) follows the Wasserstein GAN framework to improve training stability compared with the vanilla GAN objective [12], and we adopt the gradient penalty regularisation proposed in WGAN-GP [13] to enforce the 1-Lipschitz constraint.

WGAN-GP consists of a generator (G) and a critic (D) (the critic outputs real-valued scores rather than probabilities). Both networks are implemented with single-layer LSTMs.

Generator (G). The generator takes a noise sequence $z_{1:T} \in \mathbb{R}^{T \times d_z}$ with $d_z = 100$ and maps it to a synthetic return sequence $\tilde{x}_{1:T} = G(z_{1:T}) \in \mathbb{R}^{T \times 1}$. Concretely, feeding the noise into an LSTM with hidden size 128, apply a linear layer at each time step, and use a $\tanh(\cdot)$ activation to match the $[-1, 1]$ scaling range:

$$\tilde{x}_{1:T} = \tanh\big(W \, \mathrm{LSTM}(z_{1:T}) + b\big). \tag{16}$$

Critic (D). The critic receives a full return sequence $x_{1:T}$ (real or generated). Encode it with an LSTM and using the final hidden state h_T as a sequence representation. A linear layer then maps h_T to a scalar score:

$$D(x_{1:T}) = w^\top h_T + b. \tag{17}$$

This design encourages the critic to judge realism at the sequence level, rather than evaluating points independently.

Gradient Penalty. The original WGAN objective is derived from the Wasserstein-1 distance and requires the critic to be 1-Lipschitz [12]. WGAN-GP enforces this constraint through a gradient penalty [13]. In the implementation, the critic loss is:

$$\mathcal{L}_D = \mathbb{E}_{\tilde{x} \sim P_g}\big[D(\tilde{x})\big] - \mathbb{E}_{x \sim P_r}\big[D(x)\big] + \lambda \, \mathbb{E}_{\hat{x} \sim P_{\hat{x}}}\Big[\big(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1\big)^2\Big], \tag{18}$$

where $\lambda = 10$. The interpolated samples are constructed as:

$$\hat{x} = \alpha x + (1 - \alpha)\tilde{x}, \tag{19}$$

with α sampled per batch and broadcast to the sequence shape. The penalty term pushes the gradient norm toward 1 on the interpolation path between real and generated sequences, which empirically improves stability relative to weight clipping [13].

The generator is trained to increase critic scores on generated samples:

$$\mathcal{L}_G = -\mathbb{E}_{\tilde{x} \sim P_g}\big[D(\tilde{x})\big]. \tag{20}$$

Algorithm 1 WGAN-GP Training

```
Input: Real sequence dataset $X$; epochs $E$; batch size $B$;
       critic steps $n_{\text{critic}}$; gradient penalty weight $\lambda$;
       generator $G_\theta$; critic $D_\phi$; Adam optimisers for $\theta$ and $\phi$
Output: Trained generator $G_\theta$
 1: for epoch $= 1$ to $E$ do
 2:     for minibatch $x \sim X$ ($|x| = B$) do
 3:         for $t = 1$ to $n_{\text{critic}}$ do
 4:             Sample $z \sim \mathcal{N}(0, I)$
 5:             $\tilde{x} \leftarrow G_\theta(z)$
 6:             Sample $\alpha \sim U(0, 1)$
 7:             $\hat{x} \leftarrow \alpha x + (1 - \alpha)\tilde{x}$
 8:             Compute $L_D$ using: $E[D_\phi(\tilde{x})] - E[D_\phi(x)] + \lambda E[(\|\nabla_{\hat{x}} D_\phi(\hat{x})\|_2 - 1)^2]$
 9:             $\phi \leftarrow$ Adam$(\phi, \nabla_\phi L_D)$
10:         end for
11:         Sample $z \sim \mathcal{N}(0, I)$
12:         $\tilde{x} \leftarrow G_\theta(z)$
13:         Compute $L_G = -E[D_\phi(\tilde{x})]$
14:         $\theta \leftarrow$ Adam$(\theta, \nabla_\theta L_G)$
15:     end for
16: end for
```

4.1.4 QuantGAN

QuantGAN is a GAN framework designed for financial time series generation that replaces recurrent networks with Temporal Convolutional Networks (TCNs). The key idea is that causal, dilated 1D convolutions can model long-range temporal dependence efficiently, while keeping training stable under a Wasserstein objective [3]. In the implementation, following the Quant-GAN design at the architecture level (TCN generator and TCN critic), and train it using the **WGAN-GP** objective to improve stability and reduce failure modes such as non-convergence and mode collapse [12][13].

Temporal Convolutional Network (TCN). A TCN processes a sequence using stacked causal convolution blocks, so the output at time t depends only on $\{x_1, \ldots, x_t\}$. To increase the receptive field without using very deep networks, QuantGAN adopts dilated convolutions, where the dilation grows exponentially with depth. In code, the dilation at level ℓ is

$$d_\ell = 2^\ell, \quad \ell = 0, 1, \ldots, L - 1. \tag{21}$$

Each TemporalBlock contains two dilated 1D convolutions with weight normalisation, followed by non-linearities and dropout, and a residual connection. Denoting a TemporalBlock by $\mathrm{TB}_\ell(\cdot)$, the stacked TCN can be written compactly as:

$$h^{(0)} = x, \qquad h^{(\ell+1)} = \mathrm{TB}_\ell\big(h^{(\ell)}\big), \qquad \ell = 0, \dots, L-1. \tag{22}$$

In implementation, causal padding is applied before the convolution and then removed to keep the output length unchanged and preserve causality. This backbone is used in both the generator and the critic.

Generator (G). The generator takes a multi-channel noise sequence $z_{1:T} \in \mathbb{R}^{d_z \times T}$ and maps it to a single-channel synthetic return sequence $\tilde{x}_{1:T} \in \mathbb{R}^{1\times T}$. In the implementation, z is sampled from a standard normal distribution, and the generator is:

$$\tilde{x}_{1:T} = G(z_{1:T}) = \mathrm{Conv}_{1\times1}\Big(\mathrm{TCN}(z_{1:T})\Big). \tag{23}$$

Here, $\mathrm{TCN}(\cdot)$ is the stacked TemporalConvNet, and $\mathrm{Conv}_{1\times1}$ denotes a point-wise 1D convolution that projects the final TCN channel dimension to 1 output channel.

Critic (D). QuantGAN uses a critic that mirrors the generator's temporal modelling capacity. The critic receives a sequence $x_{1:T} \in \mathbb{R}^{1\times T}$ (real or generated), encodes it with the same TCN backbone, and maps the last time step feature to a scalar:

$$D(x_{1:T}) = w^\top h_T + b, \qquad h = \mathrm{TCN}(x_{1:T}), \;\; h_T = h[:, T]. \tag{24}$$

This design makes the critic judge realism at the sequence level, rather than treating the sequence as independent points.

Training objective (TCN-WGAN-GP). The QuantGAN training using the Wasserstein objective with gradient penalty. Let $x \sim P_r$ be real sequences and $\tilde{x} \sim P_g$ be generated sequences where $\tilde{x} = G(z)$, $z \sim \mathcal{N}(0, I)$. Sampling interpolations:

$$\hat{x} = \alpha x + (1-\alpha)\tilde{x}, \quad \alpha \sim \mathcal{U}(0,1). \tag{25}$$

The critic loss is:

$$\mathcal{L}_D = \mathbb{E}_{\tilde{x}\sim P_g}\big[D(\tilde{x})\big] - \mathbb{E}_{x\sim P_r}\big[D(x)\big] + \lambda\, \mathbb{E}_{\hat{x}\sim P_{\hat{x}}}\Big[\big(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1\big)^2\Big], \tag{26}$$

and the generator loss is:

$$\mathcal{L}_G = -\mathbb{E}_{\tilde{x}\sim P_g}\big[D(\tilde{x})\big]. \tag{27}$$

In the training loop, the critic is updated more frequently than the generator ($n_{\text{critic}} > 1$), which is standard practice in WGAN training to keep the critic near optimal and provide a useful gradient signal for G [13].

Algorithm 2 QuantGAN Training

```
Input:  Real sequence dataset $X$; epochs $E$; batch size $B$;
        sequence length $T$; latent channels $d_z$;
        critic steps $n_{\text{critic}}$; gradient penalty weight $\lambda$;
        generator $G_\theta$ (TCN + $1 \times 1$ Conv); critic $D_\phi$ (TCN + Linear);
        Adam optimisers for $\theta$ and $\phi$
Output: Trained generator $G_\theta$
 1: for epoch $= 1$ to $E$ do
 2:     for minibatch $x \sim X$ ($|x| = B$) do
 3:         for $t = 1$ to $n_{\text{critic}}$ do
 4:             Sample $z \sim \mathcal{N}(0, I)$ with shape $(B, d_z, T)$
 5:             $\tilde{x} \leftarrow G_\theta(z)$
 6:             Sample $\alpha \sim U(0, 1)$ with shape $(B, 1, 1)$
 7:             $\hat{x} \leftarrow \alpha x + (1 - \alpha)\tilde{x}$
 8:             $L_{\text{GP}} \leftarrow E[(\|\nabla_{\hat{x}} D_\phi(\hat{x})\|_2 - 1)^2]$
 9:             $L_D \leftarrow E[D_\phi(\tilde{x})] - E[D_\phi(x)] + \lambda L_{\text{GP}}$
10:             $\phi \leftarrow$ Adam$(\phi, \nabla_\phi L_D)$
11:         end for
12:         Sample $z \sim \mathcal{N}(0, I)$
13:         $\tilde{x} \leftarrow G_\theta(z)$
14:         $L_G \leftarrow -E[D_\phi(\tilde{x})]$
15:         $\theta \leftarrow$ Adam$(\theta, \nabla_\theta L_G)$
16:     end for
17: end for
```

4.2 Proposed Model: P-RSQGAN

Based on previous QuantGAN-style generators, we propose **Projection-Regime-Specific QuantGAN (P-RSQGAN)** as a controlled, regime-aware extension of our GAN framework. We design and implement P-RSQGAN to close a practical diagnostic gap we observe in unconditional generation: a single global model can match the centre of the return distribution while under-representing volatility-dependent behaviour, particularly volatility clustering and tail events that occur more frequently during high-volatility periods. To make generation both conditional and controllable, we inject regime information into the model at two levels. We condition the generator on the target regime label so that sampling can be directed to a desired volatility state. We then strengthen this conditioning by using a projection-based conditional critic (the "P" mechanism), which forces the critic score to depend explicitly on regime embeddings rather than treating the label as an auxiliary input that can be ignored. Finally, we train the model with a WGAN-GP objective to improve optimisation stability and reduce mode collapse and oscillatory training dynamics

4.2.1 Regime definition and label construction

Training samples are constructed as sliding windows $x_i = r_{i:i+T-1} \in \mathbb{R}^T$. For each window, we assign a regime label based on **realised volatility** within that window:

$$\sigma_i = \text{Std}(x_i). \tag{28}$$

With two regimes, we use a median split:

$$y_i = \mathbb{I}(\sigma_i > \text{median}(\{\sigma_j\})) \in \{0, 1\}. \tag{29}$$

This produces a simple and reproducible partition into **low-volatility** and **high-volatility** windows. The regime label y_i is then used as the conditioning variable throughout training and sampling.

4.2.2 Model architecture

P-RSQGAN consists of a conditional generator G and a conditional critic D. Both use a **Temporal Convolutional Network (TCN)** stack with causal, dilated 1D convolutions to capture long temporal context without recurrence [3].

Generator (G): regime injected by embedding concatenation. The generator takes a noise sequence $z \in \mathbb{R}^{d_z \times T}$ and a regime label $y \in \{0, \ldots, J-1\}$. We embed the regime label into a vector $e_G(y) \in \mathbb{R}^{d_e}$, repeat it over time, and concatenate it with the noise channels:

$$E_G(y) = \text{repeat}(e_G(y), T) \in \mathbb{R}^{d_e \times T}, \qquad \bar{z} = [z; E_G(y)] \in \mathbb{R}^{(d_z+d_e)\times T}. \tag{30}$$

The concatenated input is passed through a TCN and projected to one output channel:

$$\tilde{x}_{1:T} = G(z, y) = \text{Conv}_{1\times 1}\big(\text{TCN}(\bar{z})\big) \in \mathbb{R}^{1\times T}. \tag{31}$$

This design makes the regime condition explicit at the generator input, enabling controlled sampling for a specified regime.

Critic (D): projection conditioning. The key modification relative to a standard conditional discriminator is the projection critic. After encoding the input sequence with a TCN, we take the last-time-step feature vector $h(x) \in \mathbb{R}^C$ and compute:

$$D(x, y) = w^\top h(x) + b + \langle h(x), e_D(y) \rangle, \tag{32}$$

where $e_D(y) \in \mathbb{R}^C$ is a regime embedding. This projection term injects the regime condition directly into the critic score and typically yields stronger conditional control than simply concatenating labels to inputs, because the critic is explicitly encouraged to align features with the regime embedding [24].

Summary of key changes vs. RSQGAN. Compared with the original RSQGAN conditioning framework, P-RSQGAN makes two targeted changes:

1. **Regime injection:** inject regimes into the generator via embedding concatenation at the channel level (Eqs. 30–31), enabling direct regime-controlled sampling.
2. **"P" mechanism:** replace a plain conditional critic with a projection-based critic (Eq. 32), strengthening the conditional signal in the adversarial game.
3. **Objective upgrade (training stability):** replace the conditional vanilla GAN minimax objective used in RSQGAN with a WGAN-GP objective to improve gradient quality and stabilise training.

Training objective Let $x \sim P_r(\cdot \mid y)$ be a real window with label y, and $\tilde{x} = G(z, y)$ with $z \sim \mathcal{N}(0, I)$. We train using WGAN-GP conditioned on y [13]. The critic loss is:

$$\mathcal{L}_D = -\mathbb{E}_{(x,y)}\big[D(x, y)\big] + \mathbb{E}_{(z,y)}\big[D(G(z, y), y)\big] + \lambda\, \mathbb{E}_{(\hat{x},y)}\Big[\big(\|\nabla_{\hat{x}} D(\hat{x}, y)\|_2 - 1\big)^2\Big], \tag{33}$$

where the interpolation sample is:

$$\hat{x} = \alpha x + (1 - \alpha)\tilde{x}, \qquad \alpha \sim \mathcal{U}(0, 1). \tag{34}$$

The generator is trained to increase the critic score:

$$\mathcal{L}_G = -\mathbb{E}_{(z,y)}\big[D(G(z, y), y)\big]. \tag{35}$$

In our implementation, the critic is updated more frequently than the generator ($n_{\text{critic}} > 1$), which is standard for stable Wasserstein training.

Algorithm 3 P-RSQGAN Training

```
Input: Windowed real returns $X = \{(x_i, y_i)\}$ with regimes $y_i \in \{0, \ldots, J-1\}$;
       Generator $G_\theta$ with regime embedding $e_G(\cdot)$; Projection critic $D_\phi$ with regime embedding $e_D(\cdot)$;
       Critic steps $n_{\text{critic}}$, gradient penalty weight $\lambda$, epochs $E$; Adam optimisers for $\theta$ and $\phi$
Output: Trained generator $G_\theta$
1:  for epoch $= 1$ to $E$ do
2:      for minibatch $(x, y) \sim X$ do
3:          Sample $z \sim \mathcal{N}(0, I)$
4:          $\tilde{x} \leftarrow G_\theta(z, y)$
5:          Sample $\alpha \sim U(0, 1)$
6:          $\hat{x} \leftarrow \alpha x + (1-\alpha)\tilde{x}$
7:          $L_D \leftarrow -E[D_\phi(x, y)] + E[D_\phi(\tilde{x}, y)] + \lambda E[(\|\nabla_{\hat{x}} D_\phi(\hat{x}, y)\|_2 - 1)^2]$
8:          $\phi \leftarrow \text{Adam}(\phi, \nabla_\phi L_D)$
9:          if step mod $n_{\text{critic}} == 0$ then
10:             Sample $z \sim \mathcal{N}(0, I)$
11:             $\tilde{x} \leftarrow G_\theta(z, y)$
12:             $L_G \leftarrow -E[D_\phi(\tilde{x}, y)]$
13:             $\theta \leftarrow \text{Adam}(\theta, \nabla_\theta L_G)$
14:         end if
15:     end for
16: end for
```

5. Evaluation and Comparision

Evaluating synthetic financial time series is inherently multi-dimensional. A generator may match the marginal return distribution while failing to reproduce temporal dependence, or it may produce realistic short windows but lead to implausible long-horizon price scenarios. Therefore, we organise all diagnostics used in this thesis into three complementary groups: (i) *distribution fidelity*, (ii) *temporal dependence*, and (iii) *path-wise realism*. For the proposed conditional model, we additionally introduce *regime controllability and regime consistency* diagnostics, which quantify whether the requested regime label is respected at sampling time. Table 2 summarises the diagnostics introduced above and the fixed settings used in the implementation.

5.1 Evaluation Tools

Let $\{S_t\}_{t=0}^{T}$ denote the observed price (or index) series. We work with log-returns

$$r_t = \log S_t - \log S_{t-1}. \tag{36}$$

After training each model, we generate M synthetic return sequences $\{\tilde{\mathbf{r}}_{1:H}^{(m)}\}_{m=1}^{M}$ over a horizon H. Generated sequences are inverse-transformed back to the original scale using the same normalisation statistics as in training. Unless stated otherwise, all diagnostics are computed on held-out test windows to avoid information leakage.

Density visualisation We visualise the empirical return distribution using either a histogram or kernel density estimation (KDE). This provides an intuitive check of whether the generator matches the centre of the distribution while also allocating sufficient mass to the tails.

Q–Q plots To inspect tail behaviour, we use Q–Q plots that compare empirical quantiles of generated returns against a reference distribution (typically Gaussian) or directly against real-data quantiles. Deviations from the diagonal line indicate non-Gaussianity and tail mismatch.

Temporal dependence tools

ACF of absolute returns Volatility clustering is captured by applying the ACF to a magnitude proxy such as $|r_t|$ (or r_t^2):

$$\gamma_{|r|}(k) = \rho(|r_t|, |r_{t+k}|). \tag{37}$$

A slower decay in $\gamma_{|r|}(k)$ indicates persistent volatility dependence. This diagnostic is used alongside rolling volatility to assess whether the model reproduces clustered high-variance periods.

Rolling realised volatility We compute rolling realised volatility using a fixed window size w:

$$\sigma_t(w) = \sqrt{\frac{1}{w-1} \sum_{i=t-w+1}^{t} (r_i - \bar{r}_t)^2}, \tag{38}$$

where $\bar{r}_t$ is the mean return within the rolling window. Plotting $\sigma_t(w)$ over time provides a direct visual check of volatility clustering and regime-like transitions.

Path-wise realism tools

Fan chart We use a fan chart to summarise the distributional behaviour of model-generated price trajectories in a scenario-based manner. Specifically, we sample M synthetic return sequences from the trained generator, inverse-transform them back to the original scale, and convert each sequence into a price (or index) path by cumulatively compounding returns from a common initial value S_0. At every time step t, we compute pointwise empirical quantiles across the M simulated prices to form nested prediction intervals. For example, the 90% interval is defined by the 5th and 95th percentiles, while the median path corresponds to the 50th percentile.

We visualise these intervals as layered shaded bands, where wider and lighter bands indicate lower-probability outer regions and darker bands indicate higher-probability central regions.

Unlike a conditional forecasting setup that treats a historical segment as deterministic, our GANs generate complete sequences from noise without conditioning on an observed past. Therefore, the fan chart is plotted over the full horizon to reflect unconditional scenario dispersion. Overlaying the realised price path on the fan chart provides an intuitive calibration-style check of whether the observed trajectory is statistically plausible under the generated distribution and whether the model captures the accumulation of uncertainty, tail risk, and path variability over time.

Regime diagnostics

Regime controllability verification For an unconditional model, regime behaviour can only be assessed *post hoc*: we generate samples, compute $\sigma(\cdot)$ and then classify each sample as low- or high-volatility based on the real-data threshold. This checks whether both regimes appear in the generated distribution, but it does not provide control.

For a conditional model, we can request a target regime label y at sampling time and then verify whether the realised volatility distribution shifts as intended. We report the mean and standard deviation of $\sigma(\cdot)$ for samples generated under each requested regime. Clear separation indicates that the conditioning signal produces measurable changes in volatility state.

Regime consistency score To quantify how consistently a conditional model respects the requested regime, we compute a regime consistency score by comparing the *target* label y used during generation with a *predicted* label $\hat{y}$ inferred from the generated sample using the fixed evaluation rule. From these pairs $(y, \hat{y})$, we compute a confusion matrix and standard classification metrics (accuracy, precision, recall, and F_1). This produces a single interpretable summary of conditional adherence, and it also reveals asymmetric leakage patterns.

Table 2: Definitions and implementation settings

Item	**Definition**
Log-return	$r_t = \log S_t - \log S_{t-1}$
Descriptive stats	`describe()`: count, mean, std, min, 25%, 50%, 75%, max
Return ACF	$\gamma(k) = \rho(r_t, r_{t+k})$
Rolling volatility	$\sigma_t(w)$ as rolling std (Eq. 23)
Price reconstruction	$S_t = S_0 \exp(\sum_{i=1}^{t} \tilde{r}_i)$
Fan chart	Percentile bands of $\{S_t^{(m)}\}_{m=1}^{M}$ for each t
Mean adjustment (path sim)	$\tilde{r} \leftarrow \tilde{r} - \overline{\tilde{r}} + \mu_{\text{target}}$

5.2 Baseline Comparison

Before presenting model-specific comparisons, we briefly clarify the evaluation principles used throughout this section. For financial time series generation, model quality cannot be assessed by a single diagnostic. A well-performing generator is expected to jointly reproduce distributional properties (such as heavy tails and asymmetry), temporal dependence structures (including weak return autocorrelation and volatility clustering), and realistic path-wise dynamics under forward simulation.

Accordingly, the following analysis adopts a multi-dimensional evaluation perspective. We examine training dynamics to assess stability, distributional fidelity through empirical densities and Q–Q plots, temporal dependence via autocorrelation and rolling volatility patterns, and scenario generation ability using fan charts. All figures should therefore be interpreted collectively, rather than in isolation, as different diagnostics capture complementary aspects of realism.

We begin by comparing four baseline generative models—LSTM-GAN, TimeGAN, WGAN-GP, and QuantGAN—across a set of diagnostic dimensions commonly used in financial time-series evaluation. The objective of this section is not to rank models by a single score, but to identify systematic strengths and limitations shared by unconditional generators, thereby motivating the need for regime-aware extensions introduced later.

We first examine training dynamics to assess optimisation stability across models.Figure 5 and Figure 6 illustrate representative training loss trajectories for LSTM-GAN and QuantGAN. LSTM-GAN exhibits pronounced instability during early training, with large oscillations in both generator and discriminator losses before converging to a relatively flat regime. QuantGAN, built upon the Wasserstein objective, shows smoother critic behaviour and more stable long-run dynamics, albeit with slower convergence. These patterns are consistent with prior observations that Wasserstein-based formulations improve optimisation stability in sequential GAN training.

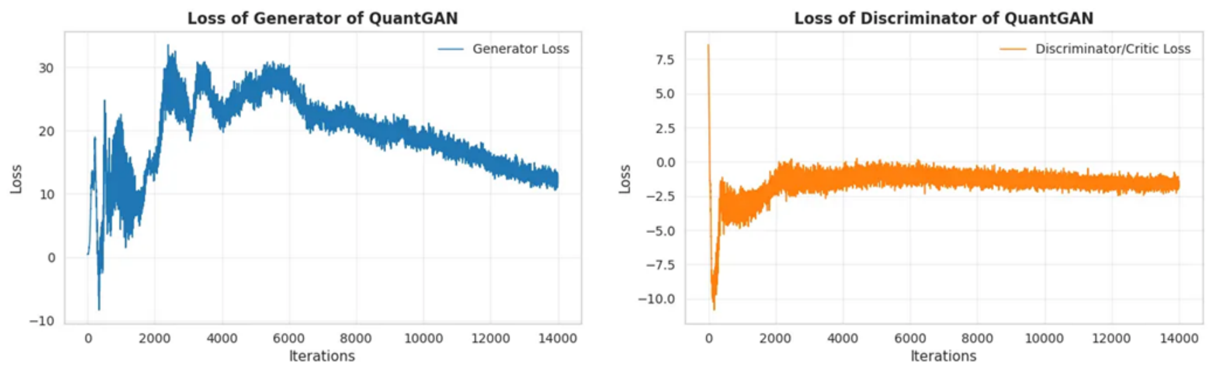


Figure 5: shows QuantGAN training losses versus iterations: generator loss (left) and discriminator/critic loss (right), with iterations on the x-axis and loss values on the y-axis.

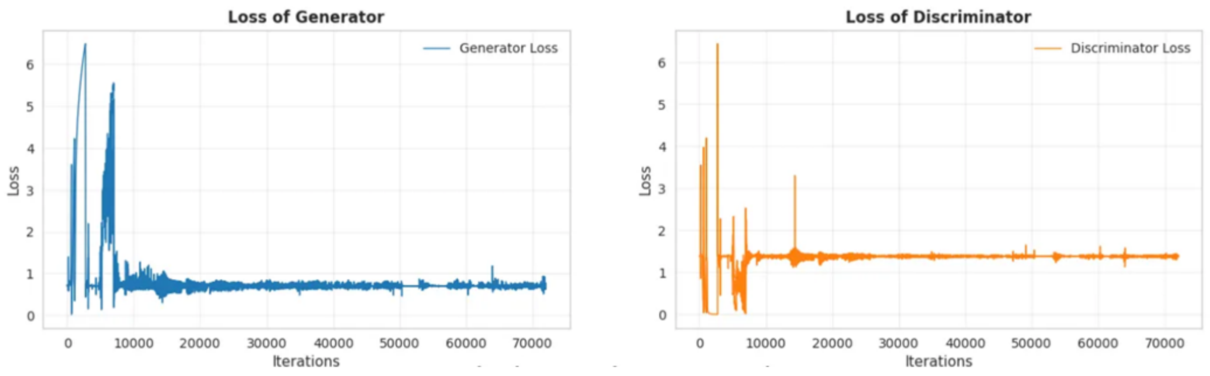


Figure 6: shows generator (left) and discriminator (right) losses over training iterations, with iterations on the x-axis and loss values on the y-axis.

Training dynamics.

Distributional fidelity and tail behaviour. We assess distributional realism using empirical density plots and Quantile–Quantile (Q–Q) diagnostics. Empirical PDFs indicate that all baseline models capture the central mass of the return distribution to a reasonable degree. However, LSTM-GAN and TimeGAN tend to underestimate tail thickness, producing overly smooth distributions. WGAN-GP improves tail coverage but exhibits a more concentrated central peak, while QuantGAN provides the closest overall balance between central density and tail dispersion.

Tail behaviour is further examined using Q–Q plots. Figure 7 provides an overall comparison across baseline models, highlighting systematic differences in tail alignment. QuantGAN and WGAN-GP closely follow empirical quantiles across most of the distribution, including the tails. In contrast, deviations at extreme quantiles are more pronounced for LSTM-GAN and TimeGAN, indicating weaker reproduction of heavy-tailed behaviour.

To further examine the tail behaviour of Wasserstein-based baselines, Figure 8 presents expanded Q–Q diagnostics for QuantGAN and WGAN-GP. Both models closely track empirical quantiles across the central region and exhibit improved alignment in the tails compared to vanilla GAN-based baselines. Minor deviations remain at the most extreme quantiles, indicating that tail risk is learned implicitly rather than explicitly controlled.

Overall, Wasserstein-based models exhibit superior tail fidelity. Nevertheless, all baseline generators remain unconditional and do not provide explicit control over tail risk or regime-specific extremes.

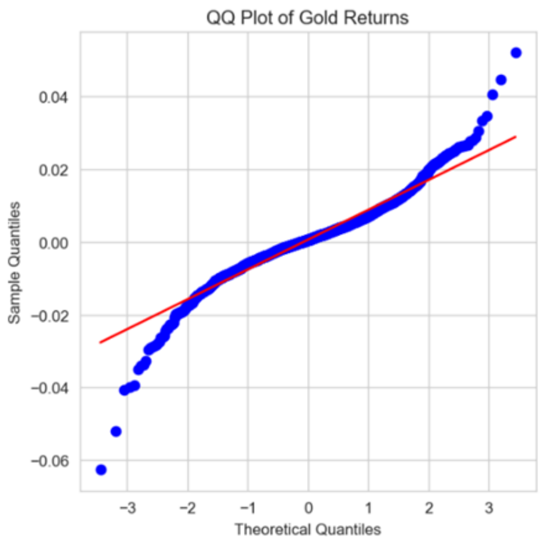
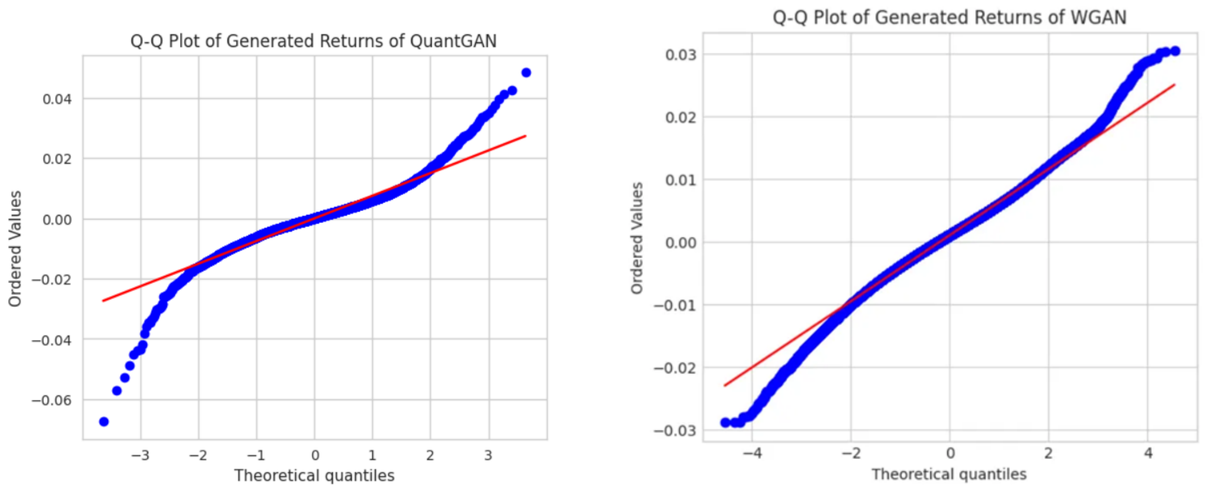


Figure 7: presents a Q–Q plot of real gold log-returns versus a normal reference: theoretical quantiles on the x-axis and empirical sample quantiles on the y-axis. Blue points denote matched quantiles and the red line is the reference line; departures from this line are most visible in the distribution tails.



(a) QuantGAN

(b) WGAN-GP

Figure 8: Quantile–Quantile (Q–Q) plots of generated returns against a normal reference distribution.

Temporal dependence. Linear temporal dependence is evaluated through the autocorrelation function (ACF) of returns. As shown in Figure 9, all baseline models reproduce the near-zero autocorrelation structure characteristic of financial returns, without introducing spurious linear dependence. Consequently, linear ACF does not provide strong discrimination among models.

Higher-order temporal structure is examined through rolling volatility dynamics. Figure 10 shows that while all models generate time-varying volatility, the resulting dynamics remain largely unconditional. Periods of high and low volatility emerge organically, but their occurrence and persistence cannot be controlled or targeted during sampling.

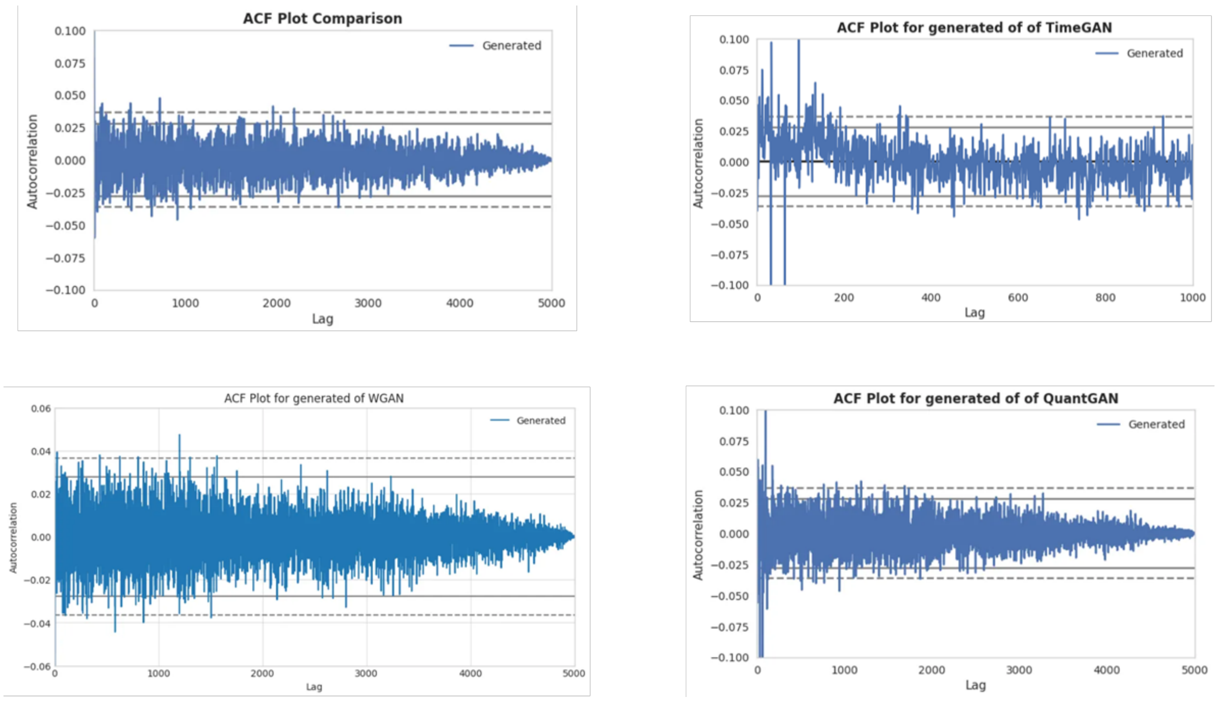


Figure 9: Autocorrelation functions (ACF) of generated return series across baseline models. The horizontal dashed lines indicate approximate 95% confidence bounds under the null hypothesis of zero autocorrelation.

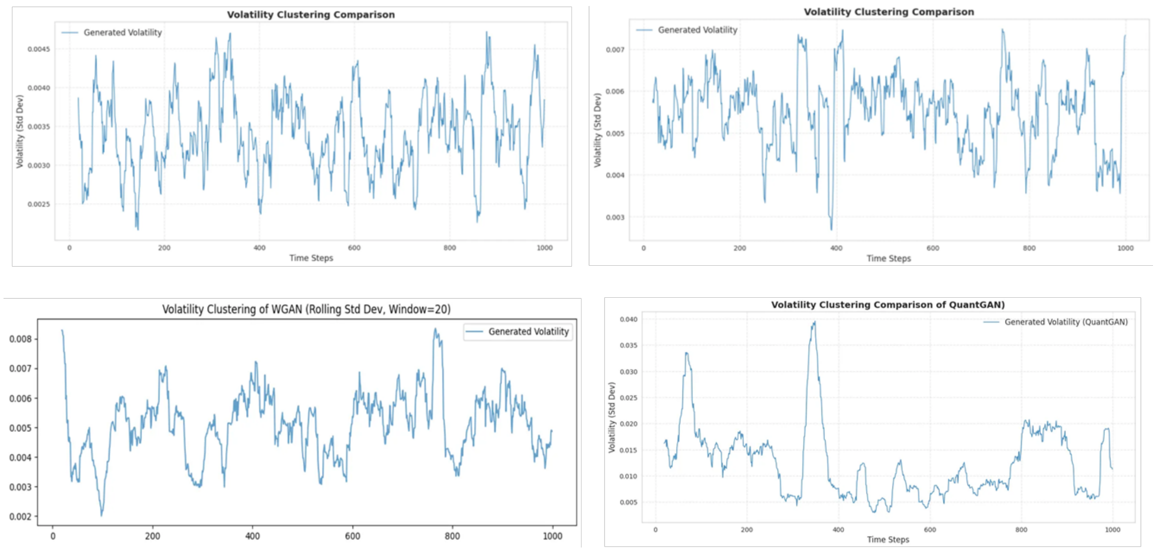
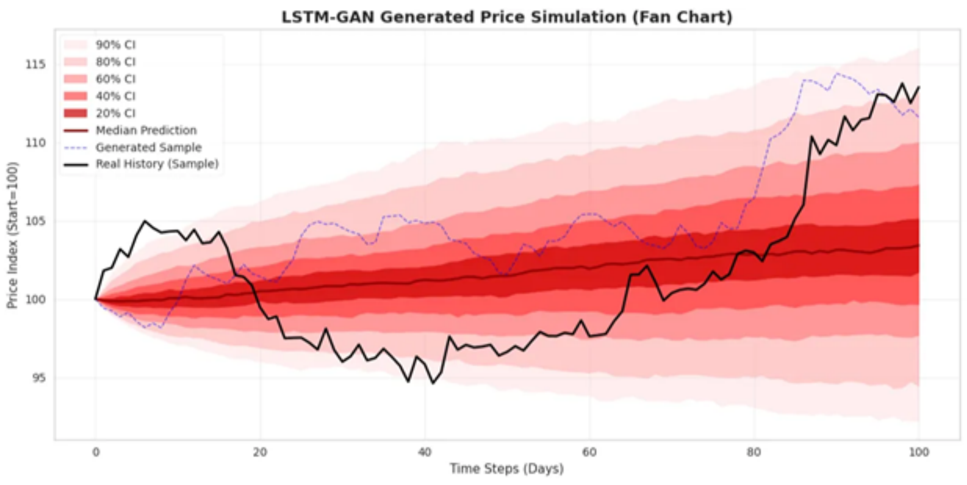
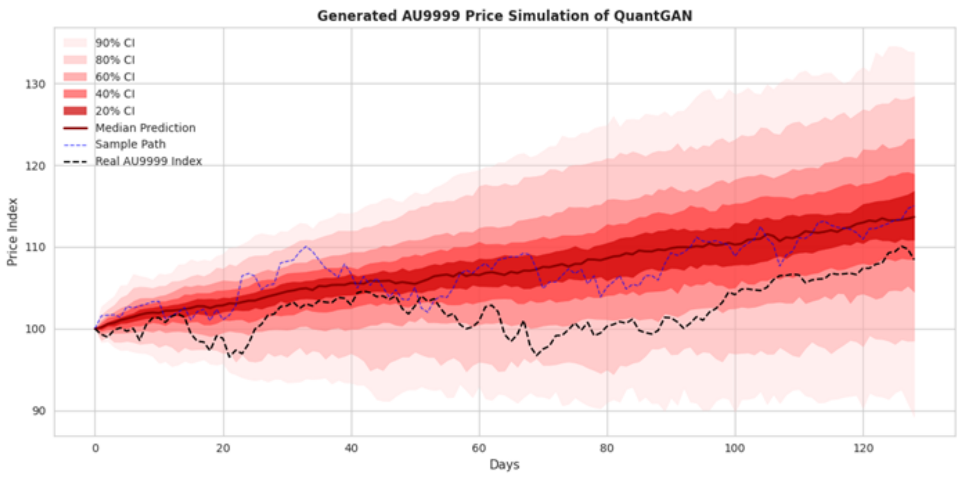


Figure 10: shows rolling realised volatility (moving-window standard deviation) over time for generated paths from different models, with time steps on the x-axis and realised volatility on the y-axis.

Scenario generation. Finally, we evaluate long-horizon scenario generation using fan charts constructed from simulated price paths. Figure 11 compares representative fan charts from LSTM-GAN and QuantGAN. LSTM-GAN produces relatively narrow prediction bands, suggesting limited uncertainty propagation. QuantGAN generates wider and more coherent fan structures, reflecting improved dispersion over long horizons. Nevertheless, both models remain unconditional: the width and shape of the fan charts cannot be explicitly adjusted to reflect different market regimes or risk states.



(a) LSTM-GAN generate NASDAQ price



(b) QuantGAN generate GOLD price

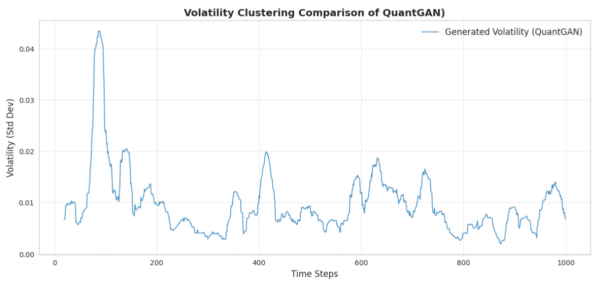
Figure 11: Fan charts constructed from generated price trajectories. Shaded bands denote empirical prediction intervals (labeled as "CI") computed pointwise across simulations: for example, the 90% CI is bounded by the 5th and 95th percentiles at each time step. The dark red line is the median (50th percentile) trajectory, the blue line is one randomly selected simulated path, and the black line is the realized price path.

Summary. Across distributional, temporal, and scenario-based diagnostics, baseline models demonstrate strong capability in matching global statistical properties of financial returns. However, a common limitation emerges: volatility dynamics and tail behaviour are learned implicitly and cannot be controlled at sampling time. This motivates the development of conditional mechanisms that allow explicit regulation of regime-dependent behaviour, which we address next.

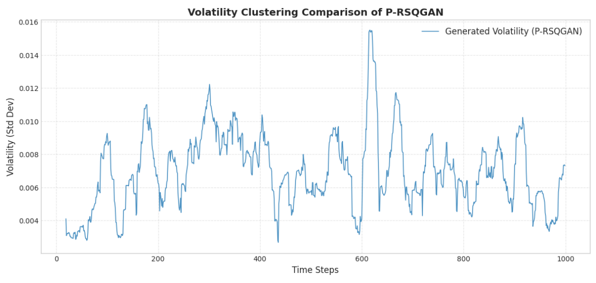
5.3 Compare P-RSQGAN with QuantGAN

After benchmarking four baseline models, we observed a recurring diagnostic gap: unconditional generators can match the global return distribution and basic temporal statistics, but they provide limited control over volatility-dependent behaviour. This becomes a practical limitation when we want to stress-test models under different market conditions, such as low-volatility versus high-volatility periods. To address this, we introduce P-RSQGAN, a conditional extension that enables regime-controlled sampling and strengthens regime enforcement via a projection-based critic.

Generated-time-series diagnostics. Figure 12 compares the rolling volatility dynamics of the two models. As shown in Figure 12a, QuantGAN exhibits a more natural progression of volatility clustering, effectively capturing the organic transitions and continuous flow characteristic of real market data; this indicates its robust capability in mimicking the aggregate statistical behaviour of the asset. Conversely, P-RSQGAN in Figure 12b displays distinct, discrete shifts between volatility levels driven by the conditional inputs. While QuantGAN arguably yields a more realistic visual representation of typical market fluidity, P-RSQGAN demonstrates the intended regime separability, reflecting its structural design to explicitly modulate risk states rather than solely replicating the unconditional data distribution.



(a) QuantGAN: Rolling volatility



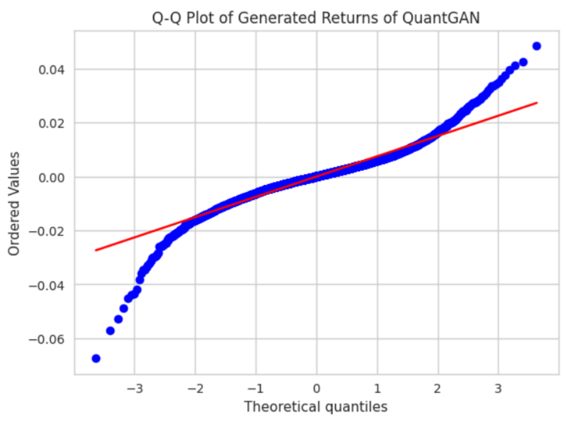
(b) P-RSQGAN: Rolling volatility

Figure 12: Comparison of rolling volatility clustering between QuantGAN and P-RSQGAN.

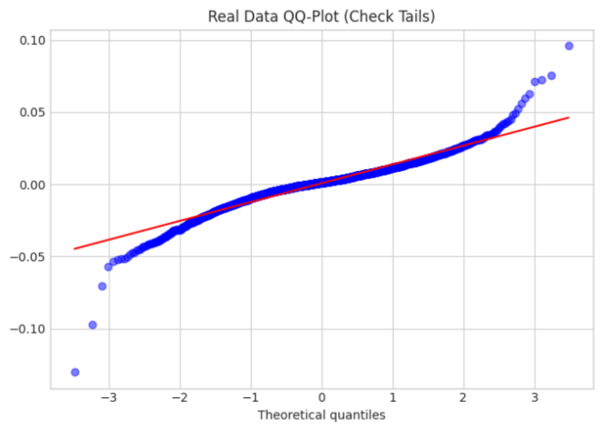
Tail behaviour (Q–Q plots). The Quantile-Quantile (Q-Q) plots in Figure 13 evaluate how well the generated returns match the distribution of real market data. The real data plot (Figure 7) serves as the benchmark, displaying the characteristic ”heavy tails” of financial time series: the data points align with the diagonal line in the centre but deviate significantly at the extreme ends.

As shown in the comparison, both models demonstrate exceptional performance in reproducing this distribution. The Q-Q plot for QuantGAN (Figure 13a) closely mirrors the benchmark, with central quantiles hugging the diagonal and tail quantiles curving away, confirming its ability to capture non-Gaussian features. Similarly, P-RSQGAN (Figure 13b) exhibits an almost identical pattern, effectively reconstructing both the central mass and the heavy tails of the return distribution.

Both models are equally effective at capturing the fundamental statistical properties of the dataset. This indicates that adding the regime-specific condition to P-RSQGAN preserves the model's ability to learn the overall global distribution without compromising statistical fidelity.

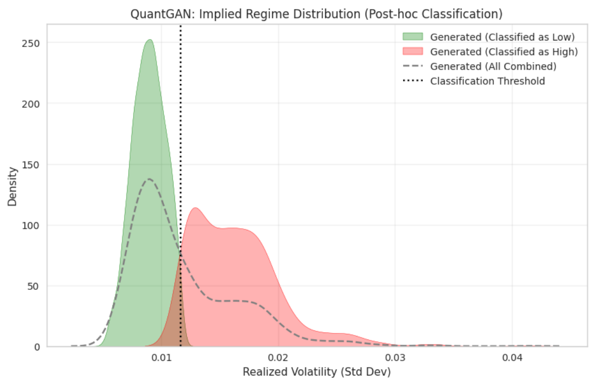


(a) QuantGAN: Q-Q Plot

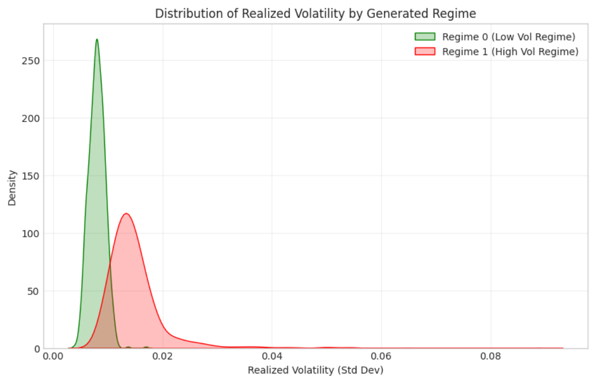


(b) P-RSQGAN: Q-Q Plot

Figure 13: Panel (a) shows QuantGAN-generated returns against a normal reference; panel (b) shows P-RSQGAN-generated returns under the same diagnostic. Deviations from the red reference line at both ends indicate heavy tails.



(a) QuantGAN: implied regime distribution



(b) P-RSQGAN: realised volatility by generated regime

Figure 14: plots density estimates of realised volatility (x-axis) across generated sequences (y-axis: density). Panel (a) shows QuantGAN samples split by a fixed threshold (vertical dashed line at 0.01168), with an additional curve for the unsplit distribution. Panel (b) shows P-RSQGAN samples grouped by the regime label used at generation.

Regime controllability. QuantGAN is unconditional and does not accept a regime label at sampling time. To check whether it still produces different volatility states, we apply a post-hoc volatility classification using the real-data median threshold (0.01168). Among 2000 generated samples, 1225 (61.7%) are classified as low-volatility and 759 (38.3%) as high-volatility. Figure 14a shows two separable groups on the realised-volatility axis, indicating that QuantGAN can generate both calm and turbulent samples. However, the regime split is only observable after generation, so regime coverage cannot be controlled directly.

P-RSQGAN models the conditional distribution $P_g(r_{1:T} \mid y)$ and enables direct regime-controlled generation by specifying $y \in \{0, 1\}$. Regime controllability is supported by realised volatility statistics: Regime 0 has mean volatility 0.007969 (std 0.001448), while Regime 1 has mean volatility 0.014876 (std 0.005903). Figure 14b further shows a clear right-shift and wider spread for Regime 1, consistent with a high-volatility regime.

Regime consistency We quantify regime adherence with a regime-consistency classifier. P-RSQGAN achieves 88.20% overall accuracy with the confusion matrix (rows = target, cols = predicted):

$$\begin{bmatrix} 999 & 1 \\ 235 & 765 \end{bmatrix}. \tag{39}$$

The error pattern is asymmetric:

- **Regime 0 is highly reliable:** recall is effectively 1.00 (999/1000), meaning that when we request low-vol samples, the generator almost always produces outputs that remain low-vol under the evaluation rule.
- **Regime 1 shows partial leakage:** recall is 0.77 (765/1000). About 23.5% of samples requested as high-vol are classified as low-vol. This suggests that the conditional constraint is strong but not perfect, and that the high-vol regime is harder to enforce consistently.

This asymmetry is consistent with common training dynamics in conditional generation: the low-vol regime corresponds to a more concentrated and easier distribution, while the high-vol regime contains more heterogeneous behaviours and more extreme events. Nevertheless, compared to QuantGAN's post-hoc regime separation, P-RSQGAN provides a clear improvement in controllable regime selection and measurable regime adherence.

6. Conclusion

This study investigated the problem of realistic and controllable financial time series generation using GAN-based models. Through a systematic comparison of LSTM-GAN, TimeGAN, WGAN-GP, and QuantGAN, we demonstrated that while modern generative models are capable of reproducing global distributional properties and basic temporal structures, unconditional generation imposes inherent limitations. In particular, volatility dynamics and tail behaviour are learned implicitly and cannot be regulated at sampling time, constraining their usefulness in risk-sensitive applications.

To address this gap, we proposed P-RSQGAN, a regime-aware extension that incorporates explicit volatility conditioning through a projection-based critic. Empirical evaluations

Table 3: Summary of P-RSQGAN Performance Results: Regime Controllability and Consistency

Panel A: Regime Controllability Verification

Statistics of realized volatility for generated samples under different regime labels.

Target Regime	**Mean Volatility**	**Vol. Std. Dev.**	**Condition Stability**
Regime 0 (Low)	0.007969	0.001448	High
Regime 1 (High)	0.014876	0.005903	Moderate

Panel B: Regime Consistency Score

Classification metrics evaluating the model's adherence to conditional inputs.

Metric	**Regime 0 (Low)**	**Regime 1 (High)**	**Overall Performance**
Precision	0.81	**1.00**	**88.20%**
Recall	**1.00**	0.77	
F1-Score	0.89	0.87	

Note: "Precision" for Regime 1 being 1.00 indicates zero false positives, meaning generated high-volatility samples are genuine.

show that the proposed model preserves distributional fidelity while enabling direct regime-controlled sampling and measurable regime consistency. Compared with unconditional baselines, P-RSQGAN offers clearer separation between volatility states and improved interpretability of generated scenarios.

This study has a number of limitations despite the positive outcomes. First, a limited number of summary data and visual diagnostics serve as the main foundation for the evaluation. Although these studies are useful, findings may vary depending on the plots and window settings selected. Second, generalization to other markets, sampling frequencies, and longer horizons may be hampered by our experiments' emphasis on constrained assets and very short fixed-length windows. Lastly, the practical usefulness of the synthetic data is only indirectly evaluated because the current setup does not directly examine downstream utility (e.g., risk forecasting, scenario development for stress testing, or strategy backtesting).

Overall, this work demonstrates that explicit regime awareness constitutes a practical and effective mechanism for enhancing controllability in financial time series generation. By bridging global distribution learning with conditional structure enforcement, regime-aware GANs provide a promising foundation for stress testing, scenario analysis, and other downstream financial applications.

Future work can address these issues in several directions. By employing multi-regime clustering, hidden Markov models, or joint criteria that incorporate trend and liquidity proxies, regime design on modelling can be enhanced beyond a binary volatility split. Stronger conditional enforcement, such as enhanced projection conditioning, supplementary regime classification losses, or re-weighted sampling that highlights high-volatility windows during training, may be investigated to lessen high-regime leakage.

References

[1] P. Glasserman, *Monte Carlo Methods in Financial Engineering.* Springer, 2004.

[2] R. Cont, “Empirical properties of asset returns: stylized facts and statistical issues,” *Quantitative Finance*, vol. 1, no. 2, pp. 223–236, 2001. [Online]. Available: https://doi.org/10.1080/713665670

[3] M. Wiese, R. Knobloch, R. Korn, and P. Kretschmer, “Quant gans: deep generation of financial time series,” *Quantitative Finance*, vol. 20, no. 9, pp. 1419–1440, 2020.

[4] A. Huang, M. Khushi, and B. Suleiman, “Regime-specific quant generative adversarial network: A conditional generative adversarial network for regime-specific deepfakes of financial time series,” *Applied Sciences*, vol. 13, no. 19, 2023. [Online]. Available: https://www.mdpi.com/2076-3417/13/19/10639

[5] R. F. Engle, “Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation,” *Econometrica*, vol. 50, no. 4, pp. 987–1007, 1982. [Online]. Available: http://www.jstor.org/stable/1912773

[6] T. Bollerslev, “Generalized autoregressive conditional heteroskedasticity,” *Journal of Econometrics*, vol. 31, no. 3, pp. 307–327, 1986.

[7] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time Series Analysis: Forecasting and Control*, 5th ed. Wiley, 2015.

[8] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[9] J. Yoon, D. Jarrett, and M. van der Schaar, “Time-series generative adversarial networks,” in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[10] S. Bai, J. Z. Kolter, and V. Koltun, “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling,” *CoRR*, vol. abs/1803.01271, 2018. [Online]. Available: http://arxiv.org/abs/1803.01271

[11] M. Arjovsky and L. Bottou, “Towards principled methods for training generative adversarial networks,” arXiv preprint, 2017.

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

[13] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, “Improved training of wasserstein gans,” in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[14] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, “Gans trained by a two time-scale update rule converge to a local nash equilibrium,” in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[15] W. Fu, A. Hirsa, and J. Osterrieder, “Simulating financial time series using attention,” 2022.

[16] R. Cont, M. Cucuringu, R. Xu, and C. Zhang, “Tail-gan: Learning to simulate tail risk scenarios,” 2022.

[17] J. Ericson, R. Donnelly, C. Chew, T. Jiao, A. Sanz-Ruiz, and A. Trotman, “Deep generative modeling for financial time series generation with application in value-at-risk: A comparative review,” 2024.

[18] S. Kwon and Y. Lee, “Can gans learn the stylized facts of financial time series?” in *Proceedings of the 5th ACM International Conference on AI in Finance (ICAIF ’24)*, 2024, pp. 126–133.

[19] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” 2022. [Online]. Available: https://arxiv.org/abs/1312.6114

[20] X. Yin, Y. Han, Z. Xu, and J. Liu, “Vaecgan: a generating framework for long-term prediction in multivariate time series,” *Cybersecurity*, vol. 4, 12 2021.

[21] C. Little, M. Elliot, R. Allmendinger, and S. Samani, “Generative adversarial networks for synthetic data generation: A comparative study,” 12 2021.

[22] M. Mirza and S. Osindero, “Conditional generative adversarial nets,” arXiv preprint, 2014.

[23] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, G. Klambauer, and S. Hochreiter, “Gans trained by a two time-scale update rule converge to a nash equilibrium,” *CoRR*, vol. abs/1706.08500, 2017. [Online]. Available: http://arxiv.org/abs/1706.08500

[24] T. Miyato and M. Koyama, “cgans with projection discriminator,” *CoRR*, vol. abs/1802.05637, 2018. [Online]. Available: http://arxiv.org/abs/1802.05637