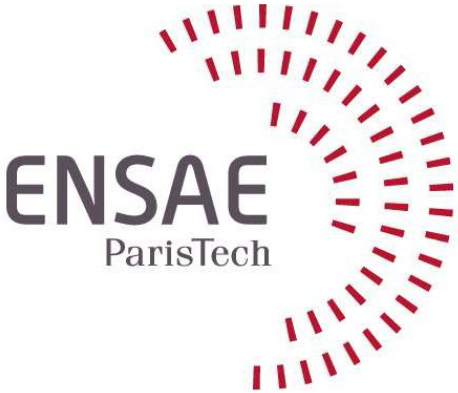


Quant GANs: Deep Generation of Financial Time Series

Machine Learning in Finance

MARZOUG Ayoub, AKIL Zakaria & HABIB TAHA

April 2025

Challenges in Financial Time Series

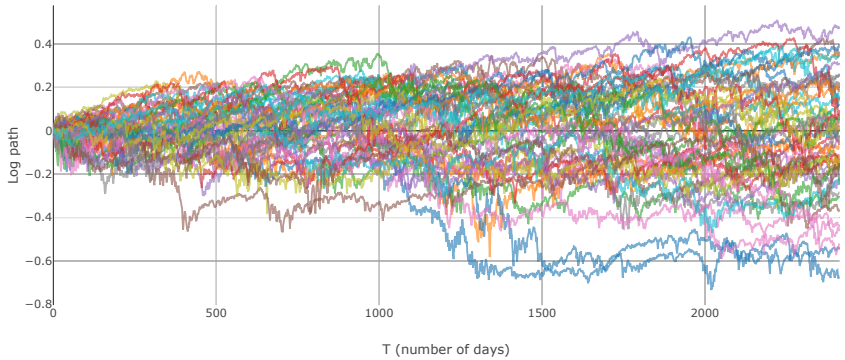
- Complex statistical properties :
  - Heavy tails, volatility clustering
  - Leverage effects, serial dependencies
- Limitations of traditional models :
  - GARCH : Conditional normality assumption
  - Black-Scholes : Constant volatility
- Need for flexible data-driven approaches

Key Motivation

- GANs can learn complex distributions without explicit parametric forms
- Challenge : Temporal dependencies & financial constraints

QuantGANs Framework

- Dual TCN architecture (Generator & Discriminator)
- Key innovations :
  - Stochastic Volatility NN (SVNN)
  - Lambert W transformation
  - Risk-neutral adjustment
- Advantages over traditional models :
  - Captures volatility clustering
  - Models heavy-tailed distributions
  - Maintains temporal consistency



Key Components

Core decomposition

$$X_t = \mu_t + \sigma_t \varepsilon_t$$

- Separately models :
  - Volatility (σ_t)
  - Drift (μ_t)
  - Innovations (ε_t)

Lambert W Transformation

- Handles heavy-tailed distributions
- Enables Gaussian-like processing
- Inverse transform preserves tail behavior

TCN Architecture

- Dilated causal convolutions
- Key features :
  - Maintains temporal ordering
  - Exponential receptive field growth
  - Avoids vanishing gradients
- Receptive Field Size (RFS) :

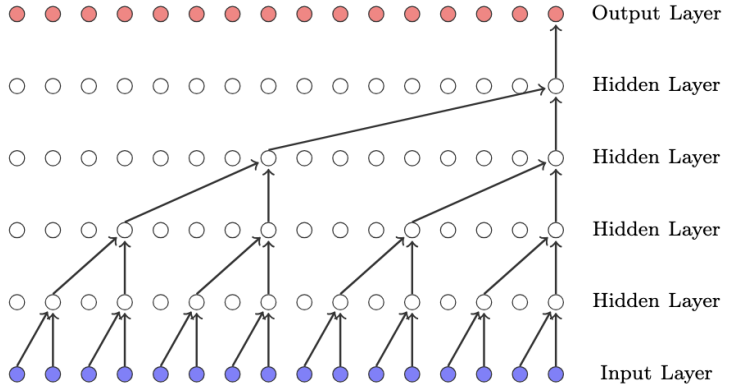
$$\text{RFS} = 1 + (K - 1) \cdot \frac{D^L - 1}{D - 1}$$

- Advantages over RNNs :
  - Parallel processing
  - Stable gradients
  - Long memory

Dilated Causal Convolution

$$[W *_D X]_{m,t} = \sum_{i=1}^{K} \sum_{j=1}^{N_I} W_{i,j,m} \cdot X_{j,t-D(K-i)}$$

- Dilation factor D controls spacing
- Kernel size K determines local context
- Causal : No future leakage



Implementation Benefits

- Stacked layers capture multi-scale patterns
- Skip connections improve gradient flow
- Efficient memory usage

Stochastic Volatility Neural Networks (SVNNs)

Log Return Neural Process

Decompose log returns of the form :

$$R_t = \sigma_t \, \epsilon_t + \mu_t$$

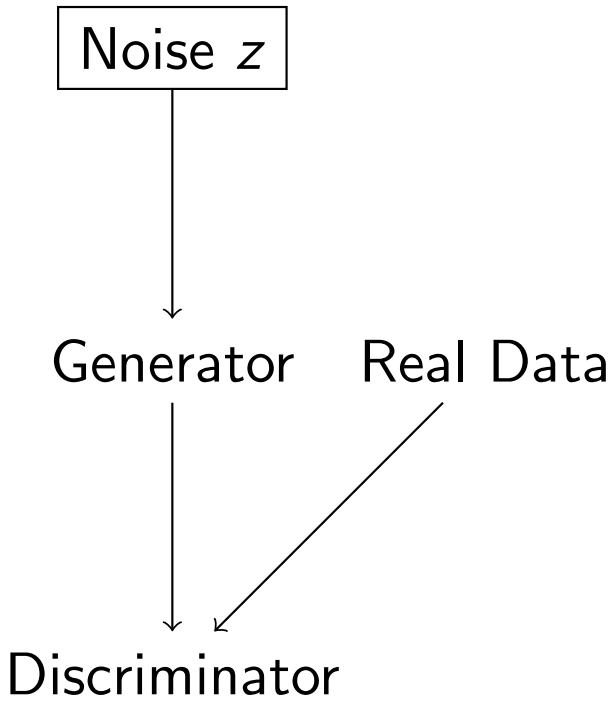
Where :

- σ_t = Volatility process (from TCN).
- μ_t = Drift process (from TCN).
- ϵ_t = Innovation process (from MLP).

GANs : Adversarial Framework

Key Components

- **Generator** G_θ :
  - Creates synthetic data from noise
  - Tries to fool discriminator
- **Discriminator** D_η :
  - Distinguishes real vs fake data
  - Acts as learned loss function

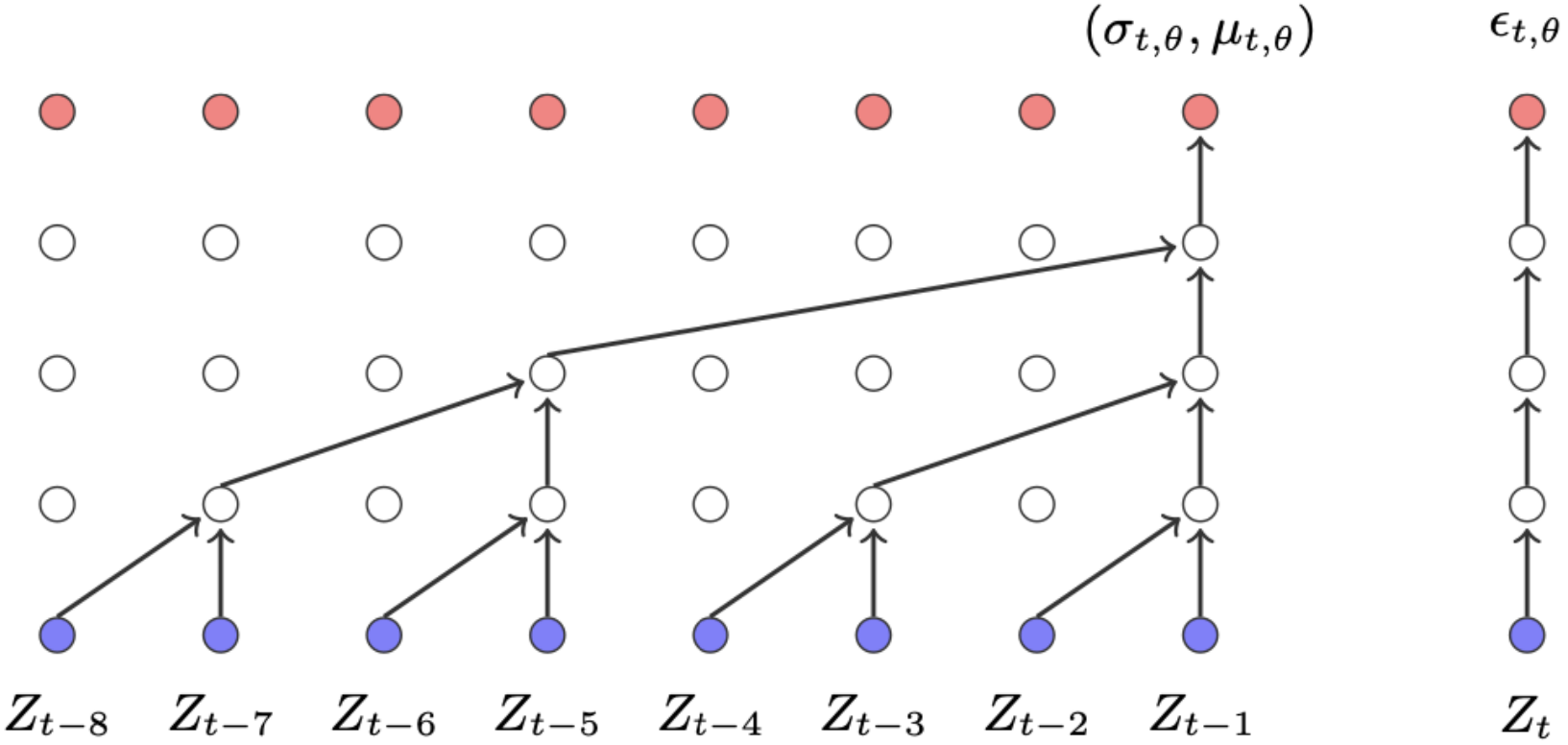


$$\min_{\theta} \max_{\eta} \mathbb{E}[\log D_\eta(x)] + \mathbb{E}[\log(1 - D_\eta(G_\theta(z))]$$

Adversarial Training

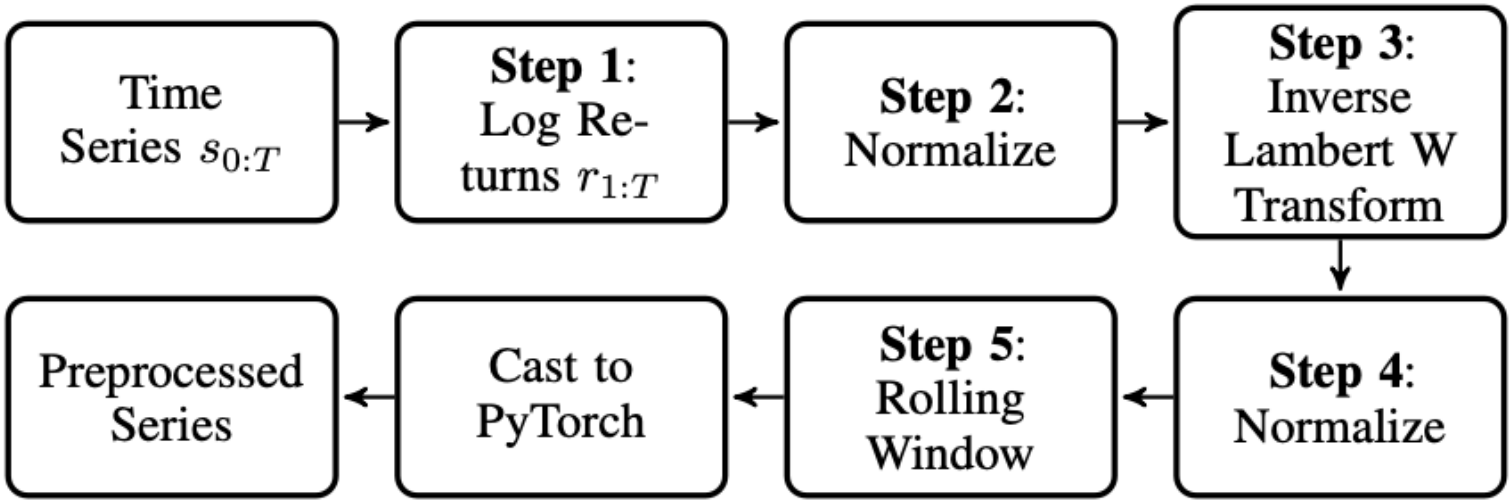
- Zero-sum game between G and D
- Nash equilibrium when $p_G = p_{data}$

SVNN : Architecture Structure



The volatility and drift component are generated by $(\sigma_t, \mu_t) = \mathrm{TCN}(Z_{t-8:t-1})$, whereas the innovation is generated by $\epsilon_t = \mathrm{MLP}(Z_t)$.

Preprocessing Pipeline



Steps :

1. Compute log returns : $r_t = \log\left(\frac{s_t}{s_{t-1}}\right)$.
2. Normalize the returns to zero mean and unit variance.
3. Apply the inverse Lambert W transform to Gaussianize heavy tails.
4. Normalize again.
5. Apply rolling window slicing for training.

Lambert W Transformation

Purpose

Stabilize training by Gaussianizing heavy-tailed returns :

$$Y = U \exp\left(\frac{\delta}{2}U^2\right)\sigma + \mu \quad \text{where} \quad U = \frac{X-\mu}{\sigma}$$

- *Inverse Lambert W* transform applied to historical returns before training.
- Lambert W transform reapplied after generation to recover heavy tails.

Risk-Neutral Distribution Adjustment

Goal

Ensure discounted prices form a martingale under the risk-neutral measure.

Given the log return :

$$R_t = \sigma_t \, \epsilon_t + \mu_t$$

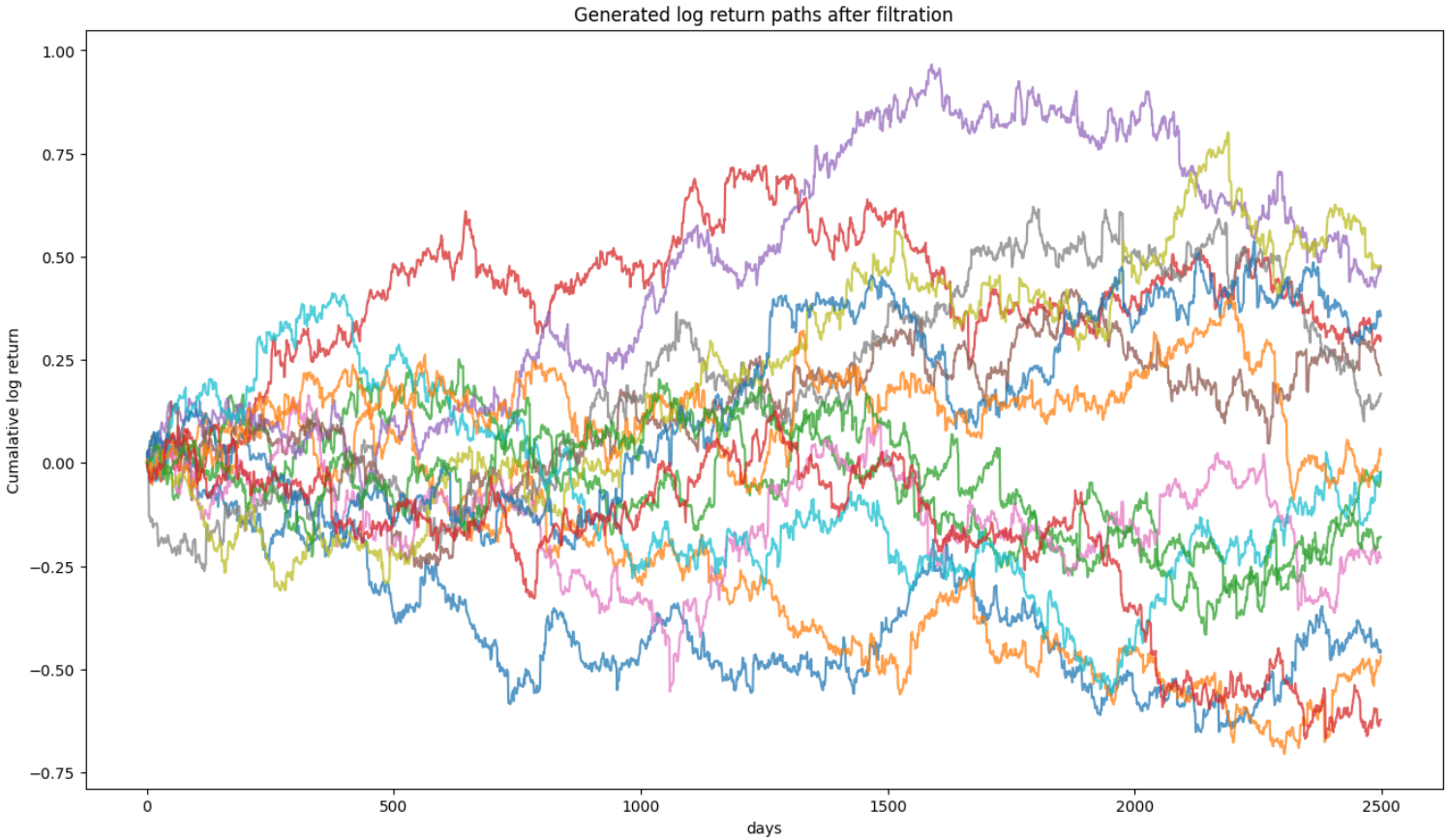
the risk-neutral adjusted return is :

$$R_t^{\mathbb{Q}} = R_t - \log\left(h(\sigma_t, \mu_t)\right) + r$$

where $h(\sigma_t, \mu_t) = \mathbb{E}\left[e^{\sigma_t \epsilon_t + \mu_t}\right]$.

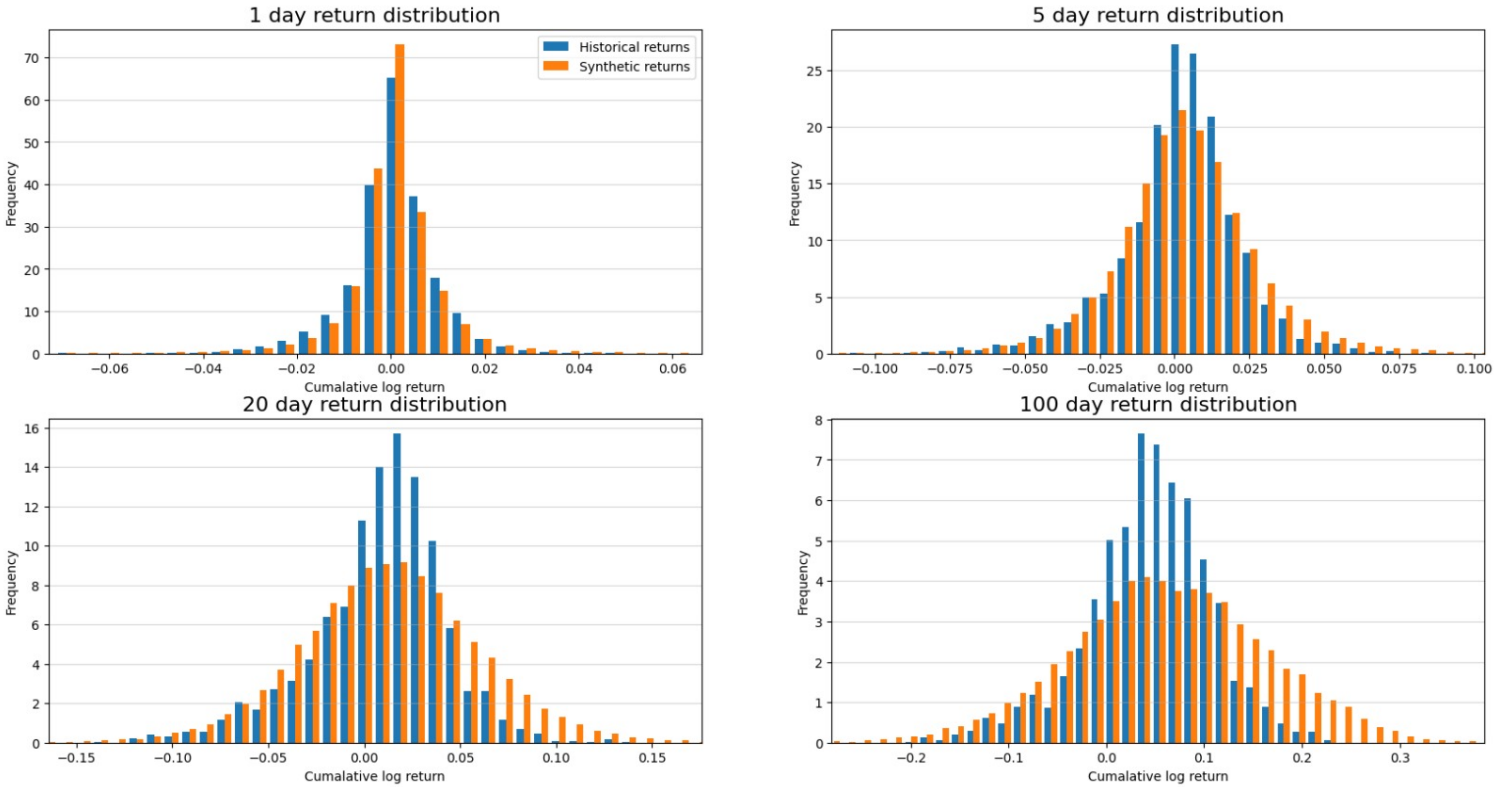
- *Interpretation :* Corrects the generator's drift to enforce no-arbitrage.
- Monte Carlo approximation used if $h(\cdot)$ is not explicit.

Pure TCN — Paths Simulation



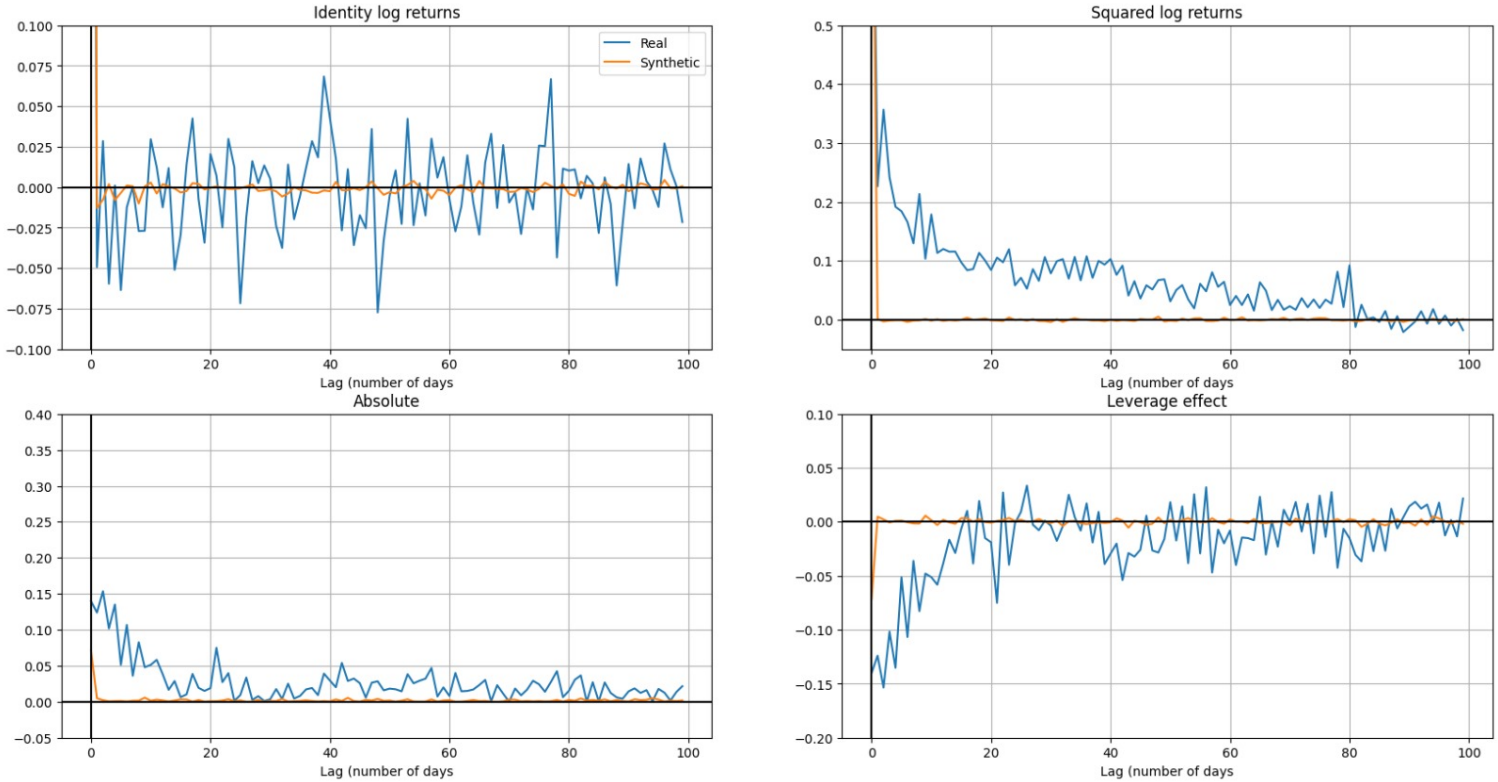
Simulated log-return paths generated by the Pure TCN model.

Pure TCN — Return Distributions



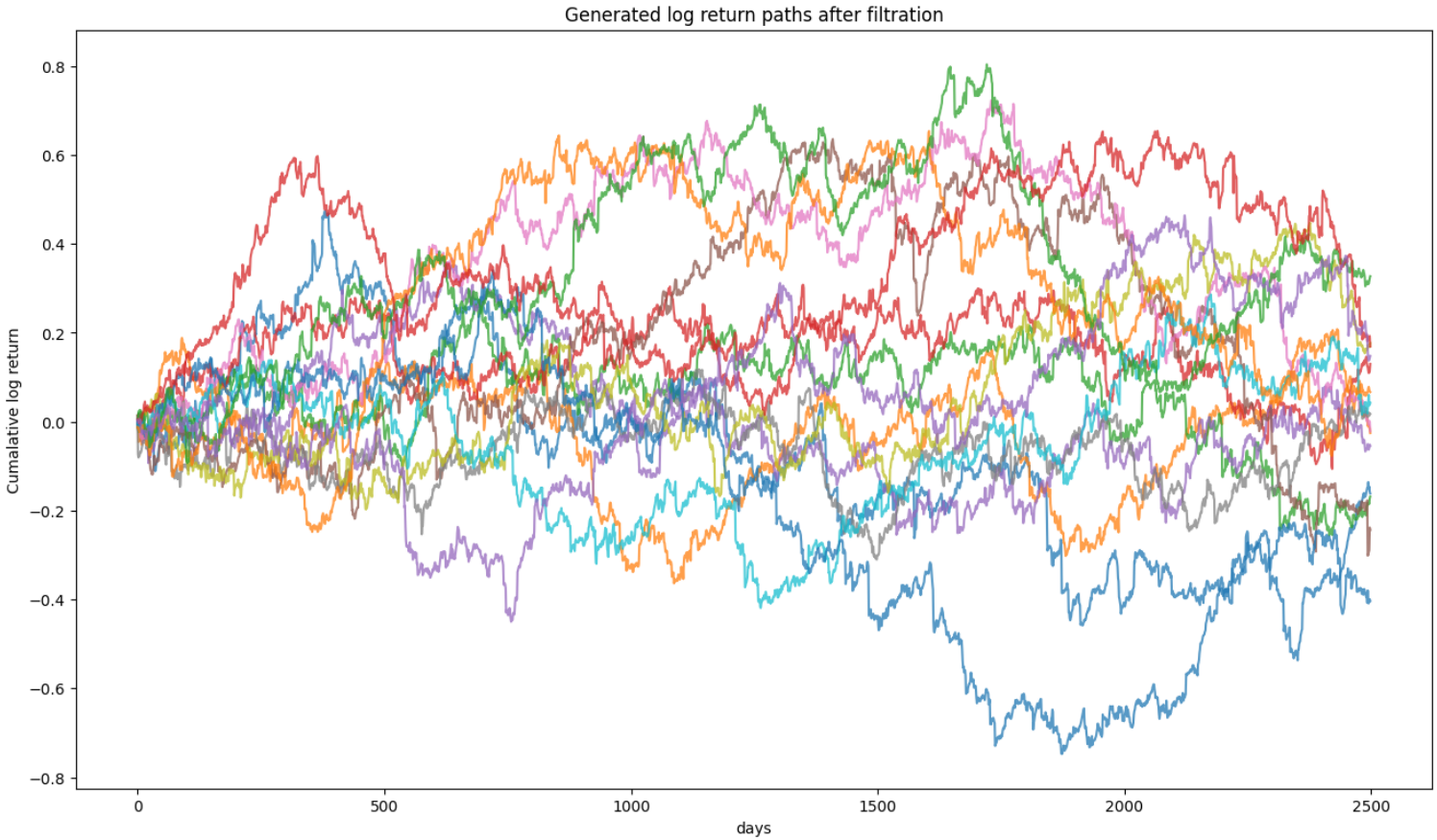
Comparison of return distributions : Synthetic vs Historical (Pure TCN).

Pure TCN — Autocorrelation Functions



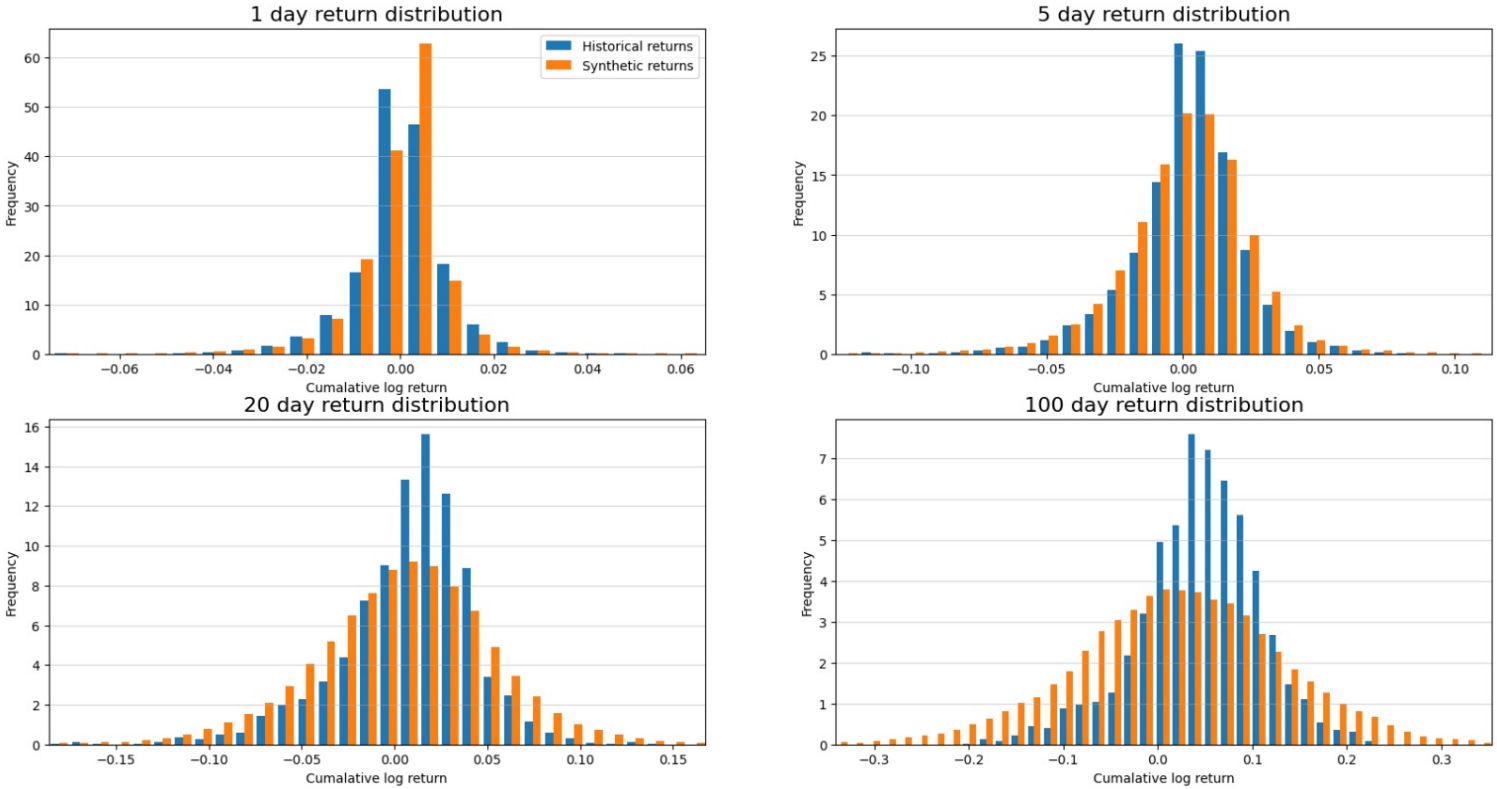
Autocorrelation functions (ACF) of returns, squared returns, and absolute returns (Pure TCN).

C-SVNN — Paths Simulation



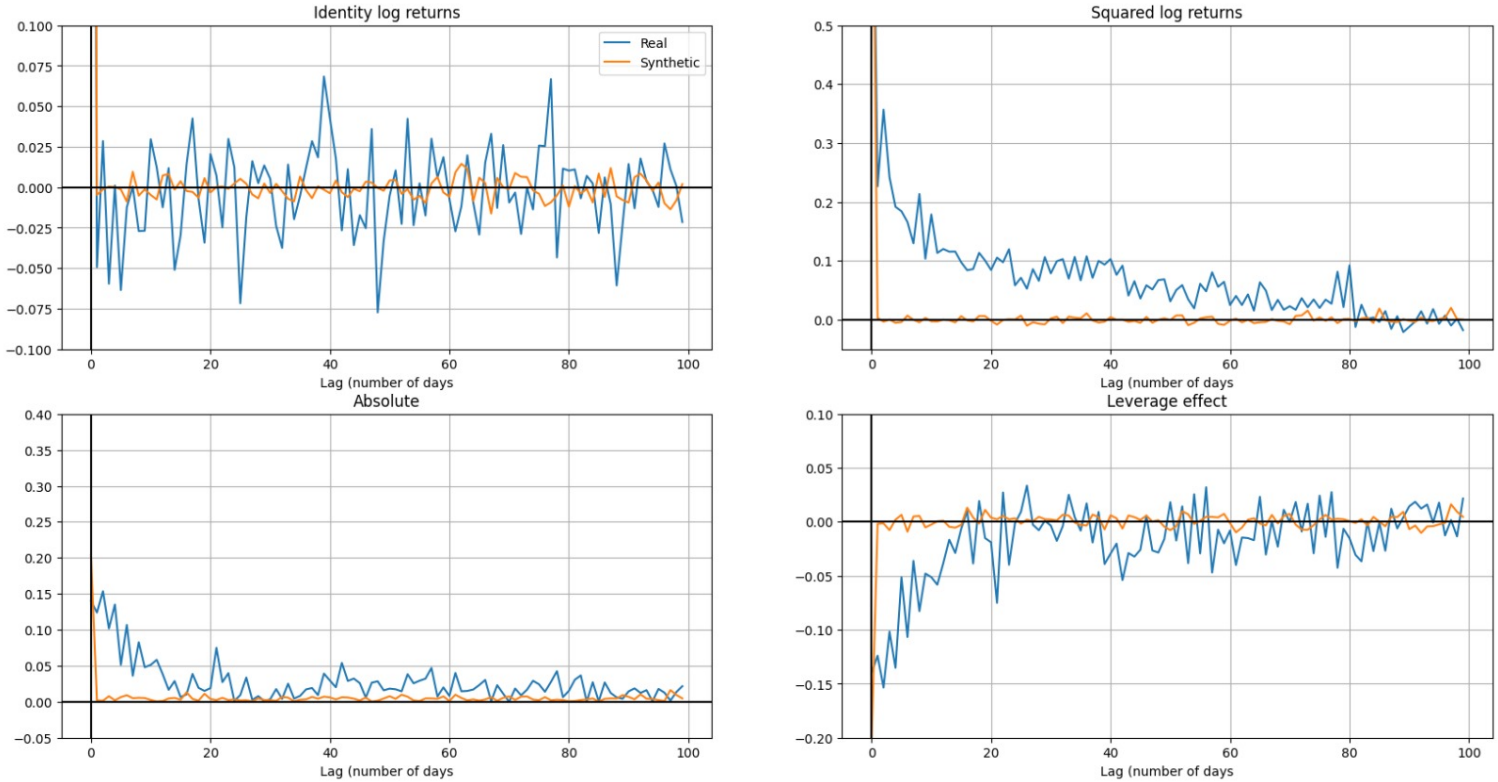
Simulated log-return paths generated by the C-SVNN model.

C-SVNN — Return Distributions



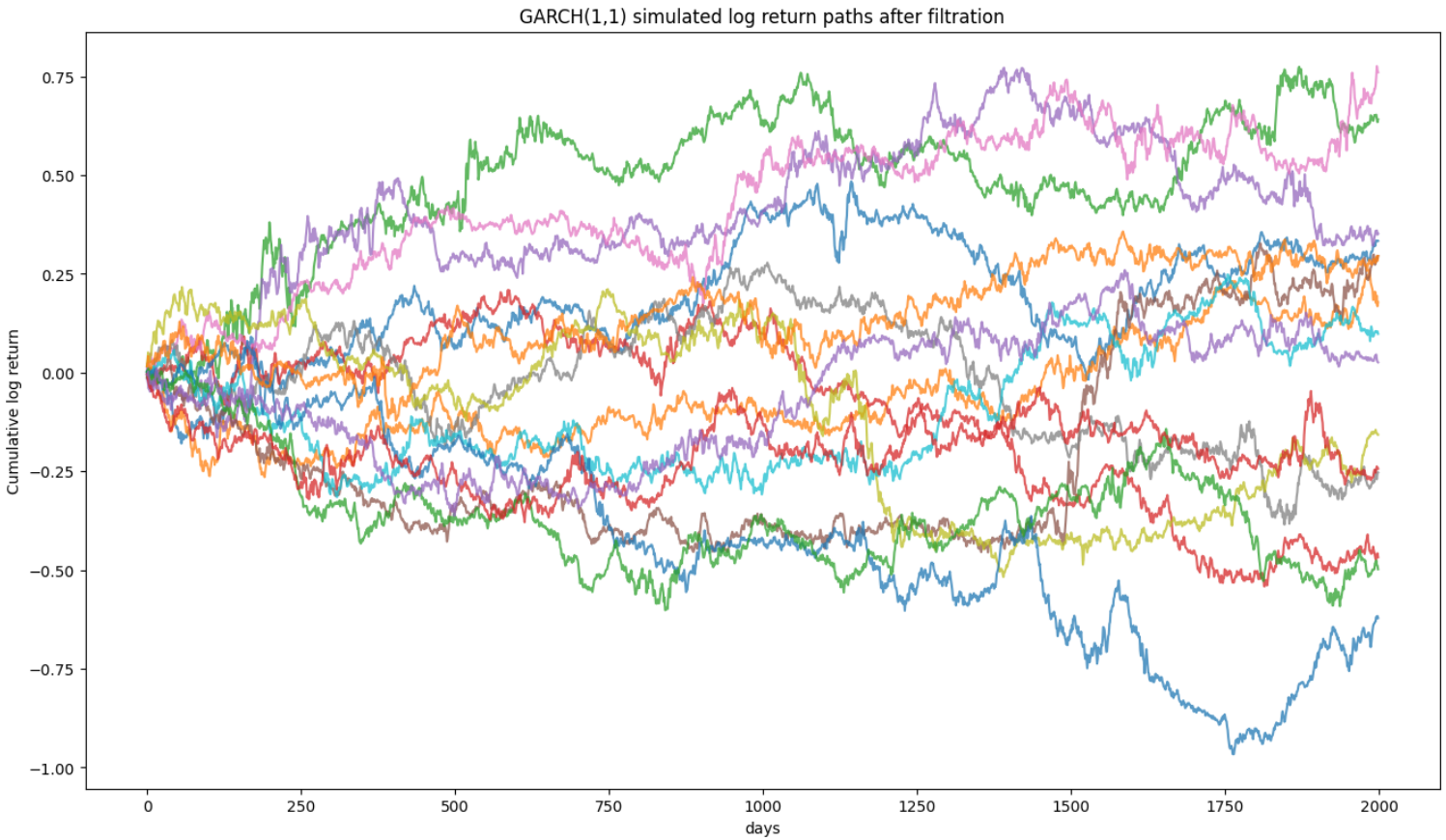
Comparison of return distributions : Synthetic vs Historical (C-SVNN).

C-SVNN — Autocorrelation Functions



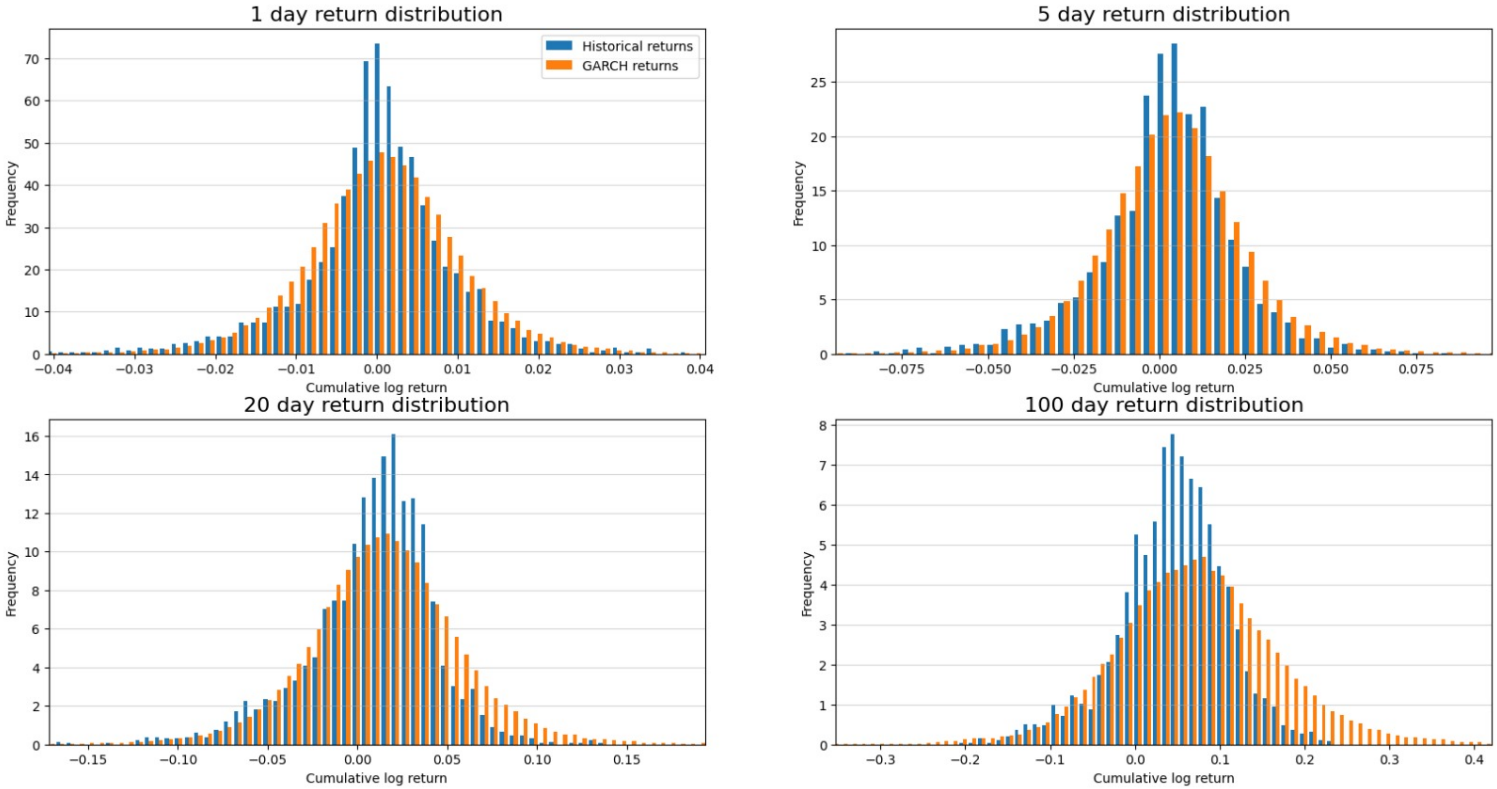
Autocorrelation functions (ACF) of returns, squared returns, and absolute returns (C-SVNN).

GARCH(1,1) — Paths Simulation



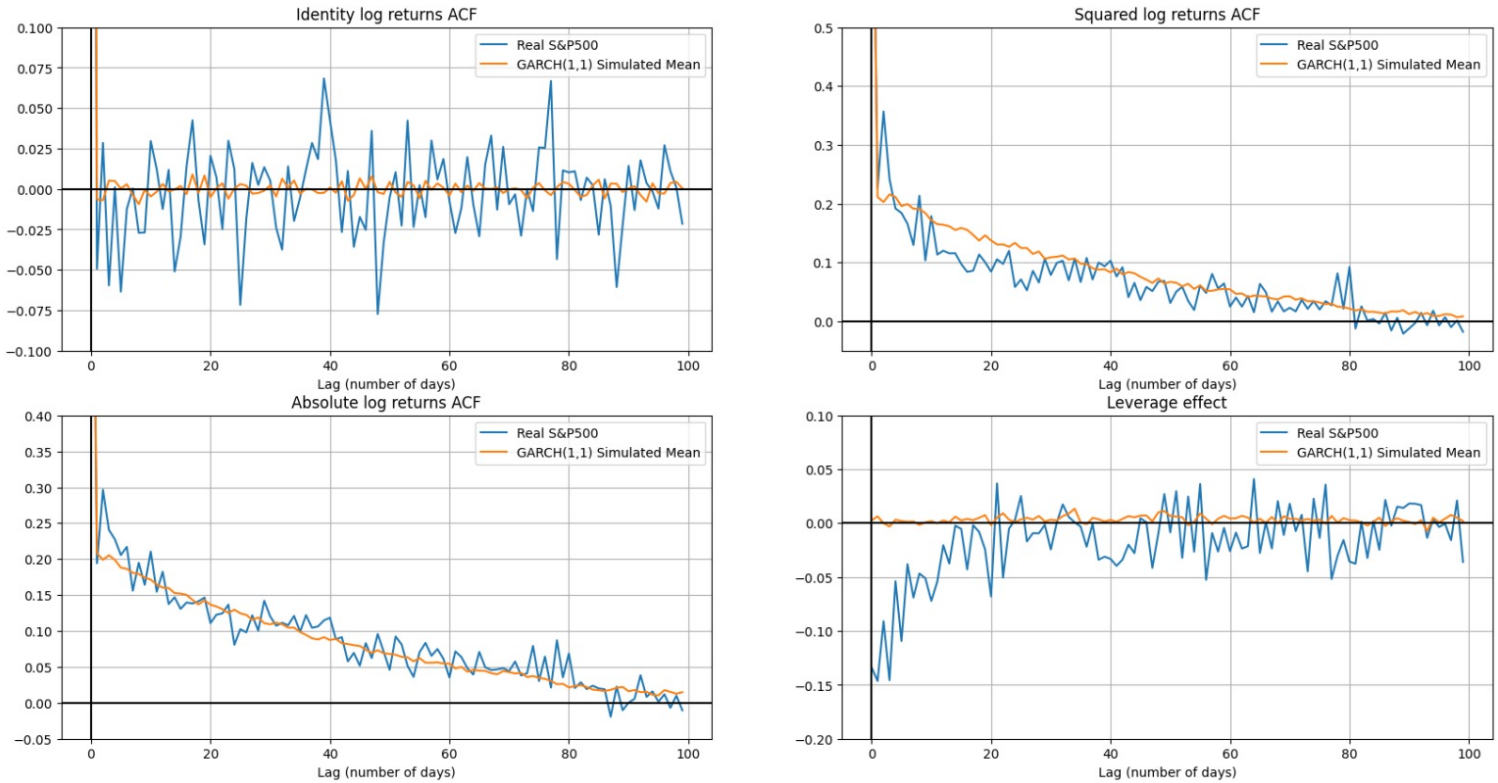
Simulated log-return paths generated by the GARCH(1,1) model.

GARCH(1,1) — Return Distributions



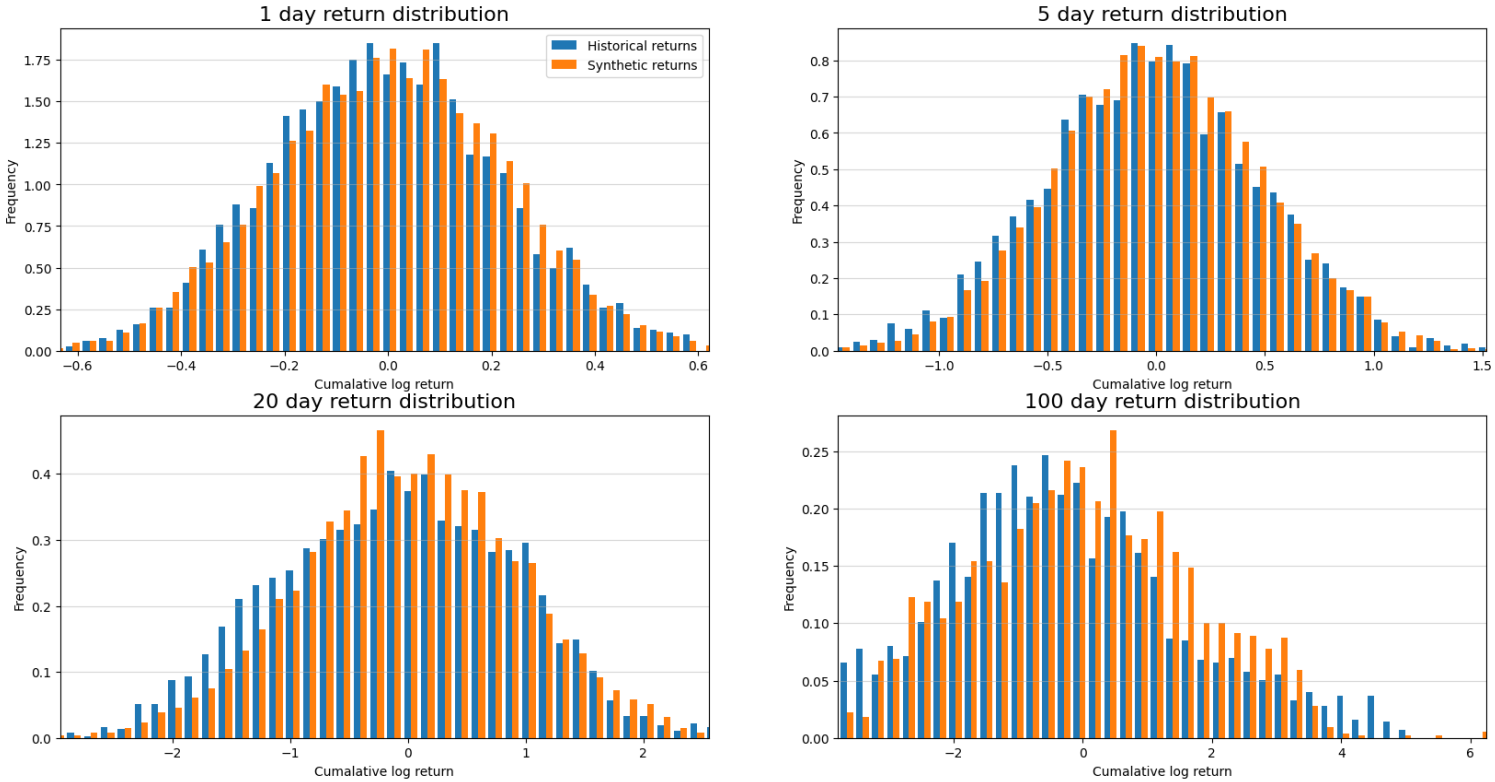
Comparison of return distributions : Synthetic vs Historical (GARCH(1,1)).

GARCH(1,1) — Autocorrelation Functions



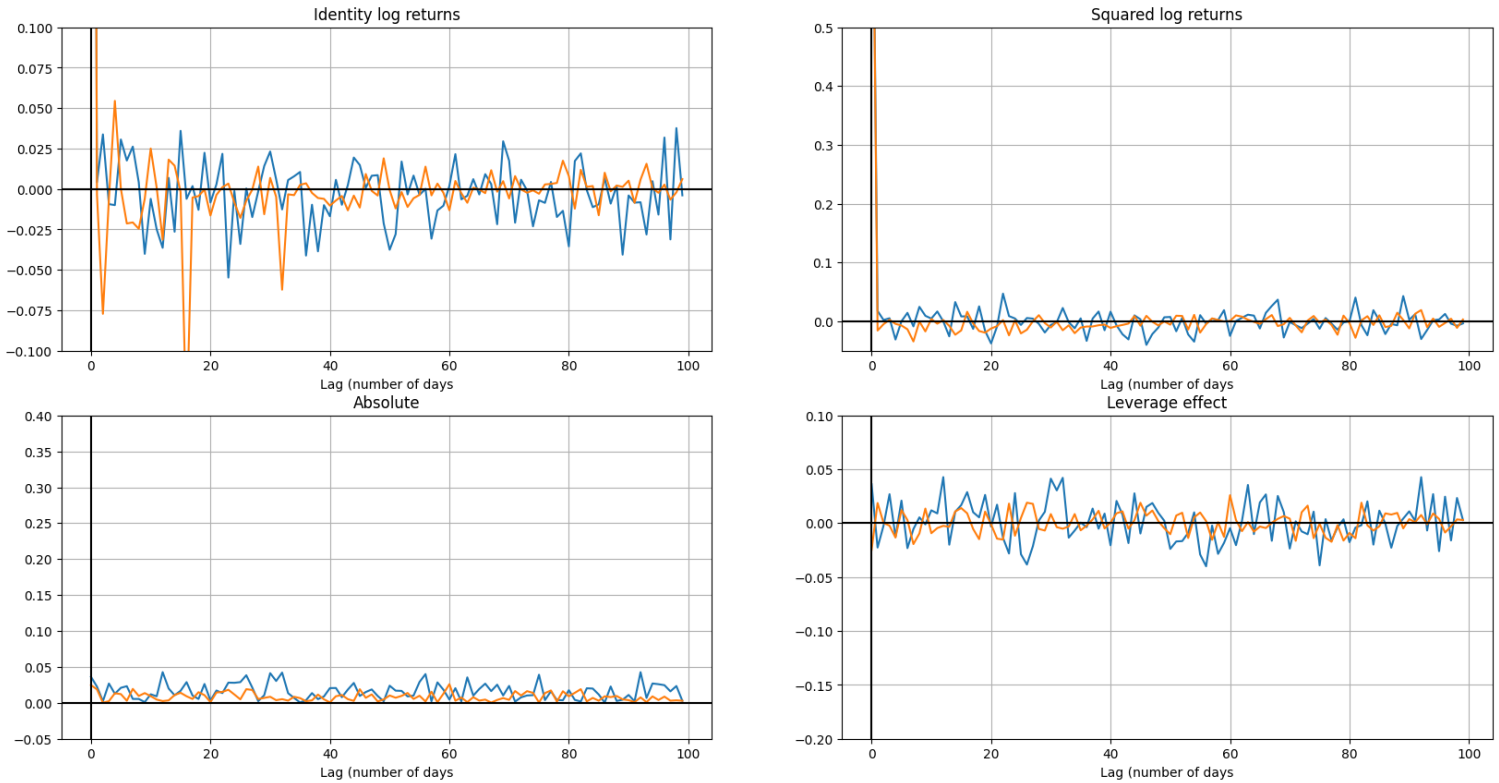
Comparison of return distributions : Synthetic vs Historical GARCH(1,1) generated Data.

Testing with GARCH(1,1) data



Comparison of return distributions : Synthetic vs Real GARCH(1,1) generated Data.

Testing with GARCH(1,1) data



Autocorrelation functions (ACF) of returns, squared returns, and absolute returns of generated data.

Metrics Comparison : Pure TCN, C-SVNN and GARCH(1,1)

Metric	Pure TCN	C-SVNN	GARCH(1,1)
EMD(1)	0.0010	0.0010	0.0011
DY(1)	0.0210	0.0158	0.0017
EMD(5)	0.0031	0.0030	0.0024
DY(5)	0.0254	0.0229	0.0083
EMD(20)	0.0108	0.0125	0.0070
DY(20)	0.0418	0.0447	0.0272
EMD(100)	0.0329	0.0415	0.0308
DY(100)	0.0595	0.0653	0.0540
Leverage Effect	0.0797	0.0977	0.2086

Table – Comparison of EMD, DY and Leverage Effect between Pure TCN, C-SVNN and GARCH(1,1) models.

Limitations & Challenges

- Mode collapse remains a challenge when training GANs on financial time series.
- Training stability is highly sensitive to hyperparameters :
  - Learning rate, number of critic iterations, weight clipping range.
    $\rightarrow$ To address this, we used weight clipping (WGAN) to enforce the Lipschitz constraint and stabilize the learning process.
- Requires careful preprocessing (Gaussianization, normalization) to further stabilize training.

Conclusion

- Quant GANs provide a promising framework for generating realistic financial time series.
- The proposed TCN and C-SVNN architectures effectively capture stylized facts of financial data.
- Future directions :
  - Explore alternative GAN objectives (e.g., WGAN-GP).
  - Extend to multi-asset or portfolio-level generation.
  - Combine with reinforcement learning for financial decision-making tasks.

Thank you for your attention !