Diffusion-Based Generative Modeling of Financial Time Series

by

Mykhailo Briazkalo

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2025

© Mykhailo Briazkalo 2025

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Financial time series demonstrate complex stochastic dynamics such as volatility clustering, heavy tails, and sudden jumps that are difficult to capture with traditional parametric models. Deep generative models offer a flexible alternative for learning unknown data distributions, but their application to financial data remains limited.

In this thesis, we propose a diffusion-based generative framework for modeling financial time series. Building on the Elucidated Diffusion Model, originally developed for image synthesis, we adapt its architecture to multivariate sequential data and integrate the Ambient Diffusion framework as a variance correction mechanism. We provide a theoretical analysis connecting excess variance in standardized outputs with volatility bias and derive an analytical rule for selecting the ambient noise level.

We evaluate the proposed approach using a comprehensive framework that combines statistical similarity, parameter recovery, option pricing, and risk metrics. Across synthetic models (GBM, Heston, Merton) and real-world datasets (SPY, AAPL, NVDA, BTC), our Ambient Diffusion-based method consistently improves distributional alignment, volatility recovery, and option pricing over the EDM baseline, highlighting its potential for quantitative modeling, scenario generation, and risk management.

Acknowledgements

I would first like to thank my supervisor, Justin Wan, for his guidance, support, and mentorship throughout my studies. I am also grateful to my readers, Toshiya Hachisuka and Yaoliang Yu, for their time and valuable feedback in reviewing my work. Finally, I would like to thank Michael Barnett-Cowan for introducing me to research and motivating me to pursue graduate studies.

Dedication

To those who believed in me.

Table of Contents

Author's Declaration ii

Abstract iii

Acknowledgements iv

Dedication v

List of Figures ix

List of Tables xii

1 Introduction 1

2 Background 5

2.1 Time Series Modeling 5

2.2 Stochastic Processes 6

2.2.1 Brownian Motion 6

2.2.2 Stochastic Differential Equations 7

2.2.3 Numerical Solutions of SDEs 9

2.3 Diffusion Models 9

2.3.1 Forward Diffusion Process 9

2.3.2 Reverse Diffusion Process . . . 10
2.3.3 Tweedie's Formula and Neural Network Training . . . 11
2.3.4 Sampling Procedures . . . 12

3 Diffusion Models for Financial Time Series 13
3.1 Problem Formulation . . . 13
3.2 Datasets . . . 13
3.2.1 Synthetic Datasets . . . 14
3.2.2 Real World Market Data . . . 26
3.2.3 Data Preprocessing . . . 27
3.3 Base model: EDM . . . 31
3.3.1 Loss function . . . 31
3.3.2 Preconditioning . . . 32
3.3.3 Training . . . 33
3.3.4 Backbone Neural Network . . . 35
3.3.5 Sampling . . . 36
3.4 Adapting EDM from Images to Time Series . . . 37
3.4.1 Data Preprocessing . . . 37
3.4.2 Mapping Images to Time Series . . . 39
3.4.3 SongUNet Architectural Adaptation from 2D to 1D . . . 40
3.5 Ambient Diffusion . . . 40
3.5.1 Introduction . . . 40
3.5.2 Problem Formulation . . . 41
3.5.3 Ambient Denoising Score Matching . . . 41
3.6 Ambient Diffusion for time series . . . 43
3.6.1 Empirical Motivation . . . 43
3.6.2 Connecting GBM volatility to variance . . . 45
3.6.3 Ambient Diffusion as a Variance Correction Mechanism . . . 48
3.7 Summary . . . 51

4 Evaluation Methodology **52**
4.1 Introduction . . . 52
4.2 Distributional Similarity . . . 53
4.3 Pathwise Time Series Statistics . . . 54
4.4 Time Series Metrics . . . 54
4.5 Financial Risk Metrics . . . 57
4.6 Option Pricing Benchmark . . . 58
4.7 Summary . . . 60

5 Experiments & Results **61**
5.1 Introduction . . . 61
5.2 Experimental Setup . . . 61
5.3 Experiments on Synthetic Datasets . . . 62
5.3.1 Geometric Brownian Motion . . . 63
5.3.2 Correlated GBM . . . 68
5.3.3 Heston Model . . . 74
5.3.4 Merton Jump Diffusion Model . . . 80
5.4 Experiments on Real-World Market Data . . . 82
5.4.1 Single-Asset Data . . . 86
5.4.2 Open-High-Low-Close Structures Data . . . 87
5.5 Summary . . . 94

6 Conclusion **95**

References **97**

APPENDICES **105**

A Additional Results **106**
A.1 Additional Results for the CGBM Dataset . . . 106
A.2 Additional Results for Single-Asset Datasets . . . 106

List of Figures

3.1 Sample trajectories of GBM simulated using the closed-form solution. The parameters used in the simulation are: $\mu = 0.05$, $\nu = 0.1$, $S_0 = 100$, time horizon $T = 1.0$, number of time steps $N = 252$ and number of paths $M = 5$. 16

3.2 Sample trajectories of two correlated GBM processes simulated using the closed-form solution. The parameters used in the simulation are: $\mu = 0.05$, $\nu = 0.1$, $S_0 = 100$, $T = 1.0$, $N = 252$, and correlation $\rho = -0.8$. 19

3.3 Sample simulations from the Heston stochastic volatility model. The top panel shows asset price trajectories, while the bottom panel shows the corresponding variance paths. Parameters used: $\mu = 0.05$, $\kappa = 1.0$, $\bar{v} = 0.05$, $\eta = 0.1$, $v_0 = 0.05$, $\rho = -0.8$, $S_0 = 100$, time horizon $T = 1.0$, and number of time steps $N = 252$. 22

3.4 Sample simulations from the Merton Jump diffusion model. Parameters used: $\mu = 0.05$, $\nu = 0.1$, $\mu_J = -0.05$, $\sigma_J = 0.1$, $\lambda = 0.5$, $S_0 = 100$, time horizon $T = 1.0$, and number of time steps $N = 252$. Noticeable negative jumps occur around $t \approx 0.03$ and $t \approx 0.6$. 25

3.5 Candlestick chart of the S&P 500 index showing daily OHLC prices over a one-year period. Each candlestick captures the open, high, low, and close prices for a single trading day. 27

3.6 S&P 500 index prices and corresponding log returns from 2015 to 2025. The top panel shows the raw price series, while the bottom panel displays the log returns. 30

3.7 Probability density function of noise levels used in training. Most of the values are centered around $\sigma \approx 0.1$. 34

3.8 Distribution of pathwise volatility estimates for GBM: training data (blue) vs. EDM-generated data (orange). The EDM model overestimates volatility. 46

3.9 Distribution of pathwise volatility estimates under Ambient Diffusion with increasing ambient noise levels. Larger $\sigma(t_n)$ values reduce the gap between training and generated data. 47

5.1 Comparison of sample paths from the reference GBM process (left), and those generated by EDM (middle) and Ambient (right) models. 64

5.2 Log return distributions of training and generated data for the GBM dataset. The blue region represents the training distribution, while the orange region shows the generated distribution for the EDM model in (a) and the Ambient model in (b). 65

5.3 Distributions of estimated drift (top) and volatility (bottom) for the GBM dataset. Blue regions represent training data estimates, while orange regions correspond to generated samples from the EDM (left) and Ambient (right) models. 66

5.4 Sample path comparison for CGBM datasets. Top row: 2-asset case; bottom row: 3-asset case. Columns show reference simulation, EDM-generated, and Ambient-generated trajectories. 69

5.5 Estimated pairwise correlation distribution ρ_{12} for the 2-asset CGBM dataset. 71

5.6 Estimated pairwise correlation distributions for the 3-asset CGBM dataset. Each subplot shows the empirical distribution of the sample Pearson correlation ρ_{ij} between asset pairs for both EDM (top row) and Ambient (bottom row) models. 72

5.7 Volatility estimation distributions for the 3-asset CGBM dataset. 73

5.8 Sample paths generated by different models for the Heston dataset. The top-left plot shows reference Monte Carlo simulations from the Heston SDE. All generative models were initialized with the same starting conditions. 77

5.9 Realized volatility estimation across generated paths for the Heston dataset. The Ambient model with formula-based σ_{t_n} better captures the heavy-tailed volatility dynamics. 78

5.10 Sample paths generated for the Merton Jump Diffusion (MJD) dataset. The reference paths (left) are simulated using the true MJD SDE with jumps. The EDM (middle) and Ambient (right) models are trained to replicate these dynamics. 81

5.11 Estimated drift and volatility distributions for the MJD dataset. Left column: EDM model; right column: Ambient model. 83
5.12 Estimated distributions of MJD jump parameters for EDM (left) and Ambient (right). Includes jump mean and jump volatility. 84
5.13 Sample paths of prices for SPY (top row) and BTC (bottom row). Left: original dataset. Right: paths generated by the Ambient model. 88
5.14 Estimated drift and volatility distributions using the Ambient model. Top row: SPY. Bottom row: BTC. 89
5.15 Estimated Value at Risk (VaR) and Conditional Value at Risk (CVaR) distributions using the Ambient model. Top: SPY. Bottom: BTC. 90
5.16 Comparison of real and generated OHLC data using candlestick plots for BTC. 92
5.17 Pairwise pathwise correlation coefficient distributions for BTC OHLC data (training vs generated). 93
5.18 Mean correlation matrices of OHLC data for BTC: training vs generated data. 94

A.1 Drift estimation distributions for the 2-asset CGBM dataset. 107
A.2 Drift estimation distributions for the 3-asset CGBM dataset. 108
A.3 Volatility estimation distributions for the 2-asset CGBM dataset. 109
A.4 Sample paths of prices for AAPL. Left: original dataset. Right: paths generated by Ambient model. 110
A.5 Estimated drift and volatility distributions for AAPL using the Ambient model. 110
A.6 Estimated realized volatility for AAPL: generated vs. training data. 111
A.7 Risk estimates for AAPL: VaR and CVaR distributions under the Ambient model. 111
A.8 Sample paths of prices for NVDA. Left: original dataset. Right: paths generated by Ambient model. 112
A.9 Estimated drift and volatility distributions for NVDA using the Ambient model. 112
A.10 Estimated realized volatility for NVDA: generated vs. training data. 113
A.11 Risk estimates for NVDA: VaR and CVaR distributions under the Ambient model. 113

List of Tables

3.1 Summary of Synthetic and Real-World Datasets Used 28

3.2 Training regimes and loss functions in the Ambient Diffusion framework. . 43

5.1 Training hyperparameters and grid search ranges. Bold indicates the selected configuration for both models. 62

5.2 Simulation Setup for Geometric Brownian Motion 63

5.3 Distributional similarity metrics between generated and training log return distributions (lower is better). 64

5.4 European option prices and relative error compared to theoretical prices. Moneyness is defined relative to $S_0 = 100$. 67

5.5 Simulation setup for Correlated Geometric Brownian Motion (CGBM) datasets. 68

5.6 Distributional similarity metrics for the 2-asset and 3-asset CGBM datasets. Lower values indicate better alignment between generated and training log return distributions. 70

5.7 Exotic option prices and relative errors for the 2-asset Correlated GBM dataset. Relative errors are computed with respect to Monte Carlo prices. Moneyness is defined relative to the average asset price ($S_0 = 100$). 75

5.8 Simulation Setup for the Heston Stochastic Volatility Model 76

5.9 Distributional similarity metrics and absolute errors for the Heston dataset. Mean and standard deviation are computed from log returns; absolute errors measure deviation from the training dataset statistics. 76

5.10 Relative error in European option pricing compared to theoretical prices on the Heston dataset. Moneyness is defined relative to $S_0 = 100$. 79

5.11 Simulation Setup for the Merton Jump Diffusion Model 80
5.12 Distributional similarity metrics for the MJD dataset. Lower values indicate better alignment between the generated and training log return distributions. 81
5.13 European option prices and relative error compared to Monte Carlo prices on the MJD dataset. Moneyness is defined relative to $S_0 = 100$. 85
5.14 Summary of real asset datasets used for training and evaluation. Each path corresponds to a non-overlapping year of daily data (252 steps). 86
5.15 Distributional similarity metrics for real asset datasets. Lower values indicate better alignment between generated and training log return distributions. 86
5.16 Distributional similarity metrics for OHLC log return distributions across real assets. Lower values indicate better alignment between generated and training data. 91

Chapter 1

Introduction

Quantitative finance is built on the premise that financial markets can be studied and modeled through mathematical and statistical tools [79]. A central task in this domain is the modeling of financial time series. A wide range of downstream applications, including portfolio optimization, forecasting, derivative pricing, and trading, depend on accurate time series models. Financial time series, typically representing asset prices or returns, serve as the primary input for these applications and guide decision-making across the industry.

However, real-world market data exhibits complex stochastic dynamics. It often violates standard assumptions such as normality and stationarity, and is subject to stylized facts such as heavy-tailed returns, volatility clustering, leverage effects, and features like sudden jumps or regime shifts [11]. Capturing these dynamics reliably and flexibly has long been a challenging modeling task.

A wide range of traditional approaches has been developed for modeling financial time series. These include statistical models such as ARIMA for trend and seasonality [7], and GARCH models for volatility [23]. In the continuous time domain, stochastic differential equation models like Geometric Brownian Motion, the Heston stochastic volatility model, and jump-diffusion processes are widely used for asset price modeling via Monte Carlo methods [79].

While these traditional models offer theoretical tractability and are interpretable, they rely on strong assumptions about distributional form, stationarity, and linearity. Their performance often depends on manual model selection, careful parameter calibration, and prior domain knowledge. As the complexity and dimensionality of financial data increase,

these limitations become more significant. Capturing non-linear high-dimensional dependencies and dynamic behaviors remains difficult using traditional parametric models. This motivates the search for more flexible and data-driven approaches.

In recent years, deep generative modeling has emerged as a state-of-the-art, non-parametric approach for learning high-dimensional data distributions directly from data. Generative models are capable of capturing data structure without explicit distributional assumptions or manual feature design. For a long time, predominant approaches were Generative Adversarial Networks (GANs) [14, 28] and Variational Autoencoders (VAEs) [44, 20, 44]. While both have been successful in various domains, GANs are known to suffer from instability during training and mode collapse [65], and VAEs often produce overly smooth samples and experience posterior collapse [48]. Since then, generative modeling has evolved to include a variety of approaches and architectures, including (deep) Autoregressive Models [73, 9], Normalizing Flows [59], Energy-Based Models [8, 42], and Diffusion Models [83]. Within the broad family of generative models, diffusion-based approaches [66, 32, 68, 71] are a leading method for tasks such as image, audio, and video generation, outperforming other models in terms of sample quality and diversity [19, 45, 62, 51].

These models are flexible, scalable, and well-suited to capturing complex patterns and dependencies in data. Their ability to generate realistic samples and learn directly from raw observations has made them appealing for a wide range of data-driven modeling tasks and has motivated the application of generative modeling techniques in fields beyond image processing, including finance.

Artificial intelligence (AI) is increasingly used across the financial sector, transforming workflows in trading, risk management, compliance, and customer interaction. In particular, the banking industry is driving rapid growth in AI adoption, with global spending on AI and generative AI projected to exceed 80 billion USD by 2028 [35]. Early applications of AI in finance were dominated by traditional machine learning methods and natural language processing [47, 54]. Specialized large language models like BloombergGPT [80] have shown strong performance in tasks such as sentiment analysis, news classification, and document summarization.

Recent research highlights growing interest in AI-powered decision support, predictive analytics, and explainability in financial models [55, 27, 29]. Studies have explored applications in forecasting macroeconomic indicators [25], modeling credit risk [85], and improving transparency in financial decision-making [64]. Commonly used techniques include ensemble learning, neural networks, fuzzy logic, and hybrid models [55]. Despite broad adoption of AI in finance, most research and applications have focused on natural language processing, classification, regression, or predictive tasks.

By contrast, generative modeling techniques remain relatively underexplored in finance compared to other AI approaches. In the financial time series domain, most prior work has focused on applying GANs to time series generation tasks [78, 21]. For general time series, several diffusion-based models have been proposed, but these primarily address forecasting and imputation on non-financial datasets with strong periodicity and relatively low randomness, such as solar activity or electricity usage [50]. This gap motivates further investigation into the use of generative models for financial data modeling and simulation. To our knowledge, recent studies [58, 72] have attempted to apply diffusion models to financial time series, but they transform sequential data into image-like representations. For example, [58] train directly on price chart images, while [72] use wavelet transforms to convert time series into images. In this work, we take a different path by adapting the model architecture to time series input directly, rather than reformatting the data to match an image-based input format.

Building on this idea, we adapt one of the leading diffusion-based models for image synthesis, EDM [37], with the NCSN++ neural network architecture [71], to multivariate temporal input data. We further incorporate the Ambient Diffusion [18] framework into our work, leading to improved performance. We train the models on synthetic datasets, including Geometric Brownian Motion, Heston, and Merton jump-diffusion models, as well as on real-world financial data. Our evaluation combines statistical similarity measures with finance-specific criteria such as volatility dynamics and option pricing accuracy, allowing us to assess both generative quality and practical relevance.

Contributions. This thesis makes the following contributions:

- We adapt a state-of-the-art image synthesis diffusion model to the domain of multivariate financial time series, introducing architectural and training modifications for sequential data.
- We extend the Ambient Diffusion framework into time series modeling, and provide a theoretical analysis of its role as a variance correction mechanism.
- We propose a two-stage training procedure that calibrates the ambient noise level using a pilot run approach.
- We design a comprehensive evaluation framework combining statistical, stochastic, and finance-specific metrics to assess model performance on both synthetic datasets and real-world market data.

- We provide empirical evidence that the proposed models capture complex financial dynamics better than baseline diffusion methods.

The remainder of this thesis is structured as follows. Chapter 2 provides the necessary background on time series modeling, stochastic differential equations, and diffusion models. Chapter 3 introduces our methodology, including datasets, model architectures, and training procedures. Chapter 4 describes the evaluation framework that combines statistical, financial, and risk-based criteria. Chapter 5 presents experimental results on both synthetic and real-world datasets. Finally, Chapter 6 summarizes our findings and discusses directions for future research.

Chapter 2

Background

In this chapter, we present the mathematical foundations necessary for the remainder of this thesis. We first introduce key concepts in time series modeling, then briefly discuss stochastic processes, and finally review the principles underlying diffusion models.

2.1 Time Series Modeling

We begin by defining the concepts of discrete time series and the generative modeling process.

A discrete time series is a sequence of observations recorded at equally spaced time intervals, formally defined as a collection of random variables $\{X_t\}_{t=1}^{N}$ where t represents discrete time points. Each observation X_t can be either univariate (scalar-valued) or multivariate (vector-valued), depending on whether we observe a single variable or multiple correlated variables at each time point [7].

Generative time series modeling aims to learn the underlying probability distribution governing observed sequential data. We formalize this process through three stages that transform observations into a generative model capable of producing new samples from the learned distribution.

Definition 2.1.1 (Time Series Model)**.** *Let $\mathcal{D} = \{X_{1:N}^{(1)}, X_{1:N}^{(2)}, \ldots, X_{1:N}^{(M)}\}$ be a dataset of M observed time series, where each $X_{1:N}^{(i)} \in \mathbb{R}^{d \times N}$ represents a d-dimensional time series of length N. A time series model $\mathcal{M}_\theta$ parameterized by θ defines a probability distribution $p_\theta(X)$ over the space of possible time series.*

Definition 2.1.2 (Model Learning)**.** *Given training data $\mathcal{D}$, model learning is the process of finding optimal parameters θ^* that maximize the likelihood of the observed data:*

$$\theta^* = \arg\max_\theta \mathcal{L}(\theta) = \arg\max_\theta \sum_{i=1}^{M} \log p_\theta(X_{1:N}^{(i)})$$

Once trained, the model $\mathcal{M}_{\theta^*}$ defines a generative process over time series, allowing new samples to be drawn from the learned distribution:

$$\tilde{X} \sim p_{\theta^*}(X),$$

where the generated samples $\tilde{X}$ should exhibit the same statistical properties and dynamic behavior as the original training data.

2.2 Stochastic Processes

Stochastic processes provide the mathematical framework for modeling systems that evolve randomly over time. In this section, we introduce the key concepts of Brownian motion, stochastic differential equations, and their numerical solutions. These tools form the foundation for both the continuous-time financial models used to generate synthetic data in Section 3.2 and for the theoretical basis of diffusion models.

2.2.1 Brownian Motion

Brownian motion is a fundamental stochastic process that serves as a source of randomness in many continuous-time models. A standard Brownian motion $\{W_t\}_{t\geq 0}$ is characterized by the following properties [36]:

Definition 2.2.1 (Brownian Motion)**.** *A stochastic process $\{W_t\}_{t\geq 0}$ is called a standard Brownian motion if:*

1. *$W_0 = 0$ almost surely;*
2. *The process has independent increments: for any $0 \leq t_1 < t_2 < \cdots < t_n$, the increments $W_{t_2} - W_{t_1}, W_{t_3} - W_{t_2}, \ldots, W_{t_n} - W_{t_{n-1}}$ are independent;*
3. *The increments are normally distributed: $W_t - W_s \sim \mathcal{N}(0, t-s)$ for $0 \leq s < t$;*

4. The paths are continuous: $t \mapsto W_t$ is continuous almost surely.

The independent increments property ensures that the process has no memory, meaning that future changes are independent of past behavior given the current state. The Gaussian increment distribution with variance proportional to the time interval reflects the accumulation of many small, independent random effects. Finally, the continuity of paths is a property that makes Brownian motion suitable for modeling continuous phenomena, despite the random nature of the process.

Brownian motion serves as the building block for constructing more complex stochastic processes through transformations and compositions. In financial modeling, it represents the unpredictable component of asset price dynamics, while in diffusion models, it drives the forward corruption process that gradually transforms data into noise.

2.2.2 Stochastic Differential Equations

Stochastic differential equations (SDEs) extend ordinary differential equations by incorporating random fluctuations, providing a framework for modeling systems subject to both deterministic trends and random perturbations.

Definition 2.2.2 (Stochastic Differential Equation (SDE))**.** *A stochastic differential equation is an equation of the form:*

$$dX_t = \mu(X_t, t)\, dt + \sigma(X_t, t)\, dW_t \tag{2.1}$$

where X_t is the state variable, $\mu(X_t, t)$ is the drift function, $\sigma(X_t, t)$ is the diffusion function, and W_t is a standard Brownian motion.

The drift term $\mu(X_t, t)\, dt$ represents the deterministic component of the process, capturing trends or mean-reverting behavior. The diffusion term $\sigma(X_t, t)\, dW_t$ introduces randomness, with the magnitude controlled by the diffusion function $\sigma(X_t, t)$. The solution X_t to this SDE describes a continuous-time stochastic process.

Since Brownian motion paths are nowhere differentiable, interpreting SDEs requires a rigorous framework for integration with respect to dW_t. The differential notation dX_t and dW_t is understood in the sense of Itô calculus, which defines stochastic integrals and provides the tools for stochastic differentiation. For an in-depth treatment of SDEs and their analytical solutions, we refer readers to [57].

Itô's lemma is the stochastic calculus analogue of the chain rule and is essential for solving SDEs analytically.

Theorem 2.2.3 (Itô's Lemma)**.** *Let X_t satisfy the SDE $dX_t = \mu(X_t, t)\,dt + \sigma(X_t, t)\,dW_t$, and let $f(x, t)$ be a twice continuously differentiable function. Then $Y_t = f(X_t, t)$ satisfies:*

$$dY_t = \left(\frac{\partial f}{\partial t} + \mu\frac{\partial f}{\partial x} + \frac{1}{2}\sigma^2\frac{\partial^2 f}{\partial x^2}\right)dt + \sigma\frac{\partial f}{\partial x}\,dW_t \tag{2.2}$$

Example: Analytical Solution of Geometric Brownian Motion via Itô's Lemma Consider the SDE for geometric Brownian motion:

$$dS_t = \mu S_t\,dt + \nu S_t\,dW_t,$$

where S_t is the asset price, μ is the drift, ν is the volatility, and W_t is standard Brownian motion.

To obtain an explicit solution for S_t, we first apply Itô's lemma to the function $f(S_t) = \log S_t$:

1. Compute the derivatives:

$$\frac{\partial f}{\partial s} = \frac{1}{S_t}, \qquad \frac{\partial^2 f}{\partial s^2} = -\frac{1}{S_t^2}, \qquad \frac{\partial f}{\partial t} = 0.$$

2. Substitute into Itô's lemma:

$$\begin{aligned} d\log S_t &= \frac{\partial f}{\partial t}\,dt + \mu S_t\frac{\partial f}{\partial s}\,dt + \frac{1}{2}\nu^2 S_t^2\frac{\partial^2 f}{\partial s^2}\,dt + \nu S_t\frac{\partial f}{\partial s}\,dW_t \\ &= 0 + \mu S_t\cdot\frac{1}{S_t}\,dt + \frac{1}{2}\nu^2 S_t^2\cdot\left(-\frac{1}{S_t^2}\right)dt + \nu S_t\cdot\frac{1}{S_t}dW_t \\ &= \left(\mu - \frac{1}{2}\nu^2\right)dt + \nu\,dW_t. \end{aligned}$$

3. Integrate both sides from 0 to t:

$$\log S_t - \log S_0 = \left(\mu - \frac{1}{2}\nu^2\right)t + \nu W_t.$$

4. Exponentiate to solve for S_t:

$$S_t = S_0\exp\left[\left(\mu - \frac{1}{2}\nu^2\right)t + \nu W_t\right].$$

This explicit solution shows that under GBM dynamics, the asset price at time t is log-normally distributed.

2.2.3 Numerical Solutions of SDEs

While some SDEs have closed-form analytical solutions, many practical applications require numerical methods to approximate solutions. The Euler–Maruyama scheme is the most common numerical method for solving SDEs and forms the basis for Monte Carlo simulation of stochastic processes. Consider the SDE:

$$dX_t = \mu(X_t, t)\, dt + \sigma(X_t, t)\, dW_t$$

with initial condition X_0. The Euler–Maruyama approximation with time step Δt is given by:

$$X_{t+\Delta t} = X_t + \mu(X_t, t)\Delta t + \sigma(X_t, t)\sqrt{\Delta t}\, Z \tag{2.3}$$

where $Z \sim \mathcal{N}(0, 1)$ is an independent standard normal random variable at each time step.

The Euler–Maruyama method approximates the continuous-time SDE by discretizing time into small intervals of size Δt. At each step, the random increment $\sqrt{\Delta t}\, Z$ represents the increment of Brownian motion over Δt. This method is used for simulating stochastic processes that do not have analytical solutions, such as the Heston stochastic volatility model discussed in later chapters.

2.3 Diffusion Models

Diffusion models are a class of generative models that learn to generate data by modeling a gradual denoising process. The diffusion model framework consists of several components, which we discuss in this section. We follow the SDE formulation proposed by [71], which provides a unified theoretical perspective on the continuous-time dynamics underlying the diffusion process. Earlier discrete-time formulations, such as DDPM [66, 32] and SMLD [68, 69] can be viewed as particular cases of the SDE formulation. Different model variants can be understood as specific choices of the drift and diffusion functions in the forward SDE discussed momentarily, along with corresponding discretization schemes.

2.3.1 Forward Diffusion Process

The forward diffusion process defines a stochastic evolution that maps samples from the original data distribution to a simple, tractable prior distribution (typically a standard Gaussian) by incrementally injecting noise over time. This gradual corruption process is formulated as a stochastic differential equation.

Let $\{X_t\}_{t=0}^T$ be a stochastic process, and let $p_t(x)$ denote the probability density of X_t. At the start of the process, $X_0 \sim p_0(x) = p_{\text{data}}$ represents clean data drawn from the empirical distribution (i.e., the training dataset). As noise is gradually added over time, the distribution evolves toward a standard Gaussian, reaching $X_T \sim p_T(x) = \mathcal{N}(0, I)$. This evolution is described by the following stochastic differential equation:

$$dX_t = f(X_t, t)\, dt + g(t)\, dW_t, \tag{2.4}$$

where $f(X_t, t)$ is the drift coefficient, $g(t)$ is the diffusion coefficient, and W_t is standard Brownian motion.

The drift term $f(X_t, t)$ controls the deterministic evolution of the process, while the diffusion term $g(t)\, dW_t$ introduces the noise that gradually corrupts the data. The specific choice of f and g determines the noise schedule and the rate at which the original data distribution transforms into the target noise distribution.

A commonly used parametrization is the Variance Exploding (VE) SDE [71, 66]:

$$dX_t = \sqrt{\frac{d[\sigma^2(t)]}{dt}}\, dW_t \tag{2.5}$$

where $\sigma(t)$ is a monotonically increasing noise schedule. In this formulation, $f(X_t, t) = 0$ and $g(t) = \sqrt{\frac{d[\sigma^2(t)]}{dt}}$, meaning the process has no drift and the diffusion coefficient is determined by the derivative of the noise variance schedule.

2.3.2 Reverse Diffusion Process

The reverse diffusion process inverts the forward corruption by gradually removing noise to recover clean data. If the forward process X_t follows the SDE (2.4), then the reverse-time process $\{X_{T-t}\}_{t=0}^T$ evolves according to the following stochastic differential equation [2]:

$$dX_t = [f(X_t, T-t) - g^2(T-t)\nabla_x \log p_{T-t}(X_t)]\, dt + g(T-t)\, d\bar{W}_t \tag{2.6}$$

where $\bar{W}_t$ is a reverse-time Brownian motion (a Brownian motion with time flowing from T to 0) and $\nabla_x \log p_t(x)$ is the score function.

The score function $\nabla_x \log p_t(x)$ represents the gradient of the log-probability density and corresponds to the direction of steepest increase in probability. The reverse SDE shows that if we knew the score function at all time points, we could exactly reverse the forward diffusion process to generate samples from the original data distribution.

2.3.3 Tweedie's Formula and Neural Network Training

Since the true score function $\nabla_x \log p_t(x)$ is unknown, one idea is to train a neural network to approximate it. Learning the score function is a challenging task on its own, and multiple methods have been proposed. Score matching [34] directly estimates the score function using integration by parts to avoid computing the intractable true score. Sliced score matching [70] improves computational efficiency by projecting the estimation onto random directions. However, both methods still require expensive Hessian computations and can be unstable in high dimensions.

Denoising score matching [75] takes a different approach by learning to denoise corrupted data rather than directly estimating score functions. While [75] established the mathematical relationship between denoising and score estimation, the connection to classical statistical theory was not known until later. It was subsequently discovered by [41, 52] that denoising score matching is a special case of a statistical result known as Tweedie's formula [60, 22], which provides a relationship between score functions and conditional expectations.

Theorem 2.3.1 (Tweedie's Formula)**.** *Let $X_0 \sim p_{data}$ and $X_t = X_0 + \sigma(t)Z$ where $Z \sim \mathcal{N}(0, I)$. Then:*

$$\nabla_x \log p_t(x) = \frac{\mathbb{E}[X_0 | X_t = x] - x}{\sigma^2(t)} \tag{2.7}$$

Tweedie's formula shows that the score function can be computed directly from the conditional expectation $\mathbb{E}[X_0 \mid X_t = x]$, which is the optimal denoiser in the mean squared error sense. This means that instead of learning the score function directly, we can train a neural network $D_\theta(x, t)$ to approximate $\mathbb{E}[X_0 \mid X_t = x]$ by solving the denoising objective:

$$\mathcal{L}(\theta) = \mathbb{E}_{t, X_0, Z}\left[\|D_\theta(X_0 + \sigma(t)Z, t) - X_0\|^2\right]. \tag{2.8}$$

Once trained, the (approximated) score function can be recovered using Tweedie's formula:

$$s_\theta(x, t) = \frac{D_\theta(x, t) - x}{\sigma^2(t)}. \tag{2.9}$$

This establishes a connection between score-based generative modeling and denoising problems.

2.3.4 Sampling Procedures

Once a diffusion model is trained, samples can be generated by numerically solving differential equations that reverse the forward diffusion process. The SDE framework provides two approaches: stochastic sampling via the reverse-time SDE and deterministic sampling via the probability flow ODE.

Stochastic sampling directly uses the reverse-time SDE (2.6), starting from pure noise and iteratively applying the learned score function $s_\theta(x, t)$ to guide the reverse dynamics. The SDE can be solved numerically using discretization schemes such as the Euler–Maruyama method.

An important property of diffusion models is that the same marginal distributions $p_t(x)$ that solve the forward and reverse SDEs also satisfy a deterministic ordinary differential equation. The probability flow ODE (2.10) produces trajectories whose marginals match those of the stochastic process and provides a deterministic sampling procedure that generates samples from the same distribution as the reverse SDE.

Theorem 2.3.2 (Probability Flow ODE)**.** *Let $\{X_t\}_{t=0}^T$ be a stochastic process defined by the forward SDE (2.4) with marginals $X_t \sim p_t(x)$. Then, the sample paths of X_t can also be described by the following deterministic ODE [71]:*

$$\frac{dx}{dt} = f(x, t) - \frac{1}{2} g^2(t) \nabla_x \log p_t(x), \tag{2.10}$$

where $\nabla_x \log p_t(x)$ is the score function of the marginal distribution at time t [71].

Similarly to stochastic sampling, the score function $\nabla_x \log p_t(x)$ is approximated by $s_\theta(x, t)$. The resulting ODE can be solved using standard numerical solvers, such as Runge–Kutta methods

Both sampling approaches offer some advantages: stochastic sampling may provide better sample diversity, while deterministic sampling provides adaptive ODE solvers, exact likelihood computation, and reproducible generation. In practice, deterministic sampling is often preferred for its computational efficiency and reproducibility, though the choice depends on specific application requirements.

Chapter 3

Diffusion Models for Financial Time Series

This chapter formulates the task of synthesizing time-series data as a generative modeling task leveraging the diffusion models. First, we discuss the selection, collection, and processing of the training dataset. We then introduce the design choices of the diffusion model and the backbone neural network, along with the necessary adaptations to handle time series data. Finally, we introduce the Ambient Diffusion framework and examine its effects on the quality and statistical properties of the generated time series.

3.1 Problem Formulation

Let $\mathcal{D}_{\text{train}} = \{X^{(i)}\}_{i=1}^{M}$ denote the training dataset, consisting of M time series samples. Each sample $X^{(i)} \in \mathbb{R}^{d \times N}$ is a time series with d features and N time steps. The generative diffusion model, parameterized by θ, is trained to learn the underlying data distribution and is used to produce a set of generated samples $\mathcal{D}_{\text{gen}} = \{\widetilde{X}^{(i)}\}_{i=1}^{M}$.

3.2 Datasets

In this section, we describe the process of collecting and creating the datasets used for model training. The data utilized in this thesis can be broadly categorized into two groups: synthetic and real-world. Synthetic datasets primarily serve as controlled toy examples,

allowing precise manipulation of underlying dynamics and features. In contrast, real-world datasets capture the inherent complexity and stochastic nature of financial markets, where the underlying processes are unknown. Using both types of datasets allows us to assess the practical applicability and effectiveness of the proposed models under controlled (synthetic) and realistic (real-world) scenarios.

3.2.1 Synthetic Datasets

Synthetic datasets are generated by solving stochastic differential equations (SDEs) that describe specific financial processes. When a closed-form solution to the SDE is available, sample paths are drawn directly from the known distribution. In cases where no analytical solution exists, the SDE is solved numerically, and sample trajectories are produced via Monte Carlo simulation [26].

The selection of synthetic datasets in this work is motivated by the desire to cover a range of typical stochastic models used in financial modeling. Each dataset exhibits distinct statistical properties and corresponds to a specific class of real-world market behaviors.

We begin with the standard Geometric Brownian Motion, which models asset prices under the assumption of constant drift and volatility. It serves as a foundational model in mathematical finance and forms the basis for the Black-Scholes model [5]. The Correlated GBM process extends this to the multivariate setting, enabling the modeling of multiple time series with cross-asset dependencies, such as portfolios of correlated securities or multi-feature representations. Next, we consider the Heston stochastic volatility model [30], which relaxes the constant volatility assumption by introducing a separate stochastic process for variance. This allows the model to capture empirically observed phenomena such as volatility clustering and leverage effects. Finally, we use the Merton Jump-Diffusion process [53], which extends GBM by incorporating discontinuous jumps in the asset price. This enables the modeling of sudden, large market movements and better reflects risk dynamics associated with rare events like crashes.

Geometric Brownian Motion

Geometric Brownian Motion (GBM) is one of the most widely known stochastic models in finance and is commonly used to describe the evolution of asset prices. It is governed by the following SDE:

$$dS_t = \mu S_t \, dt + \nu S_t \, dW_t, \tag{3.1}$$

where:

- S_t denotes the asset price at time t,
- μ is the constant drift coefficient, representing the expected return,
- ν is the constant volatility coefficient,
- W_t is a standard Brownian motion.

Note. The volatility coefficient is denoted by ν rather than the conventional σ to avoid ambiguity in later chapters, where σ will be reserved for other components related to diffusion models.

An analytical solution to the SDE (3.1) can be obtained by applying Itô's lemma (2.2.3) to $\log S_t$, yielding:

$$S_t = S_0 \exp\left[\left(\mu - \frac{1}{2}\nu^2\right) t + \nu W_t\right]. \tag{3.2}$$

To simulate M asset price paths, each of length N, we directly sample from the closed-form solution at discrete time intervals using Monte Carlo simulations. The discretized sampling formula is given by:

$$S_t^{(m)} = S_{t-1}^{(m)} \cdot \exp\left[\left(\mu - \frac{1}{2}\nu^2\right) \Delta t + \nu\sqrt{\Delta t} \cdot Z\right], \tag{3.3}$$

where $Z \sim \mathcal{N}(0,1)$ is a standard normal random variable, $m = 1, \ldots, M$, $t = 1, \ldots, N$, and $\Delta t = T/N$ is the time increment over horizon T. Algorithm 1 outlines the simulation procedure, and Figure 3.1 illustrates a set of sample trajectories generated under this scheme.

Correlated Geometric Brownian Motion

While GBM is effective for modeling the trajectory of a single asset price, many financial applications require modeling multivariate time series, such as correlated asset returns, Open-High-Low-Close (OHLC) market data structures, or the joint behavior of a stock and its derivatives. Correlated Geometric Brownian Motion (CGBM) provides a straightforward extension of the GBM framework to capture such multivariate dynamics.

The CGBM model describes a vector of asset prices $\mathbf{S}_t^{(m)} = [S_t^{(m,1)}, \ldots, S_t^{(m,d)}]^\top$, where d denotes the number of correlated assets and $m = 1, \ldots, M$ indexes independent samples. Each sample m evolves according to the system of SDEs:

$$dS_t^{(m,i)} = \mu_i S_t^{(m,i)}\, dt + \nu_i S_t^{(m,i)}\, dW_t^{(m,i)}, \quad i = 1, \ldots, d, \tag{3.4}$$

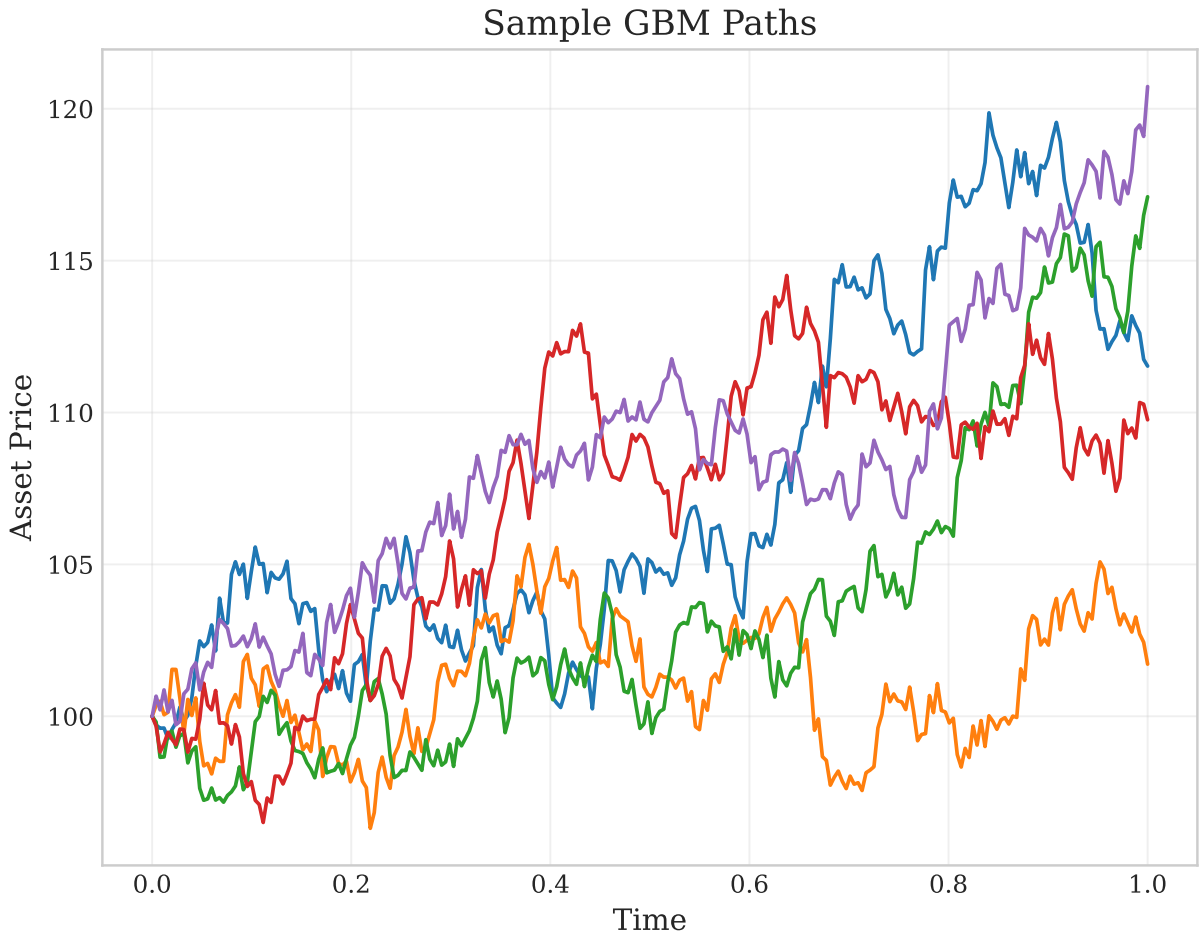


Figure 3.1: Sample trajectories of GBM simulated using the closed-form solution. The parameters used in the simulation are: $\mu = 0.05$, $\nu = 0.1$, $S_0 = 100$, time horizon $T = 1.0$, number of time steps $N = 252$ and number of paths $M = 5$.

Algorithm 1 Simulate Geometric Brownian Motion Paths

Require: Number of samples M, time steps N, drift μ, volatility ν, initial price S_0, time horizon T

$\Delta t \leftarrow T/N$ ▷ Time increment
$S_0^{(m)} \leftarrow S_0$ for all m ▷ Initialize prices
for $m = 1$ to M **do** ▷ Paths loop
  for $t = 1$ to N **do** ▷ Time loop
    $Z \sim \mathcal{N}(0, 1)$ ▷ Sample noise
    $S_t^{(m)} \leftarrow S_{t-1}^{(m)} \times \exp\left[\left(\mu - \frac{1}{2}\nu^2\right)\Delta t + \nu\sqrt{\Delta t} \cdot Z\right]$ ▷ Update
  end for
end for
return $S \in \mathbb{R}^{M \times N}$ ▷ Simulated dataset

where each $W_t^{(m,i)}$ is a Brownian motion, and the collection $\{W_t^{(m,i)}\}_{i=1}^d$ are jointly correlated such that:

$$\mathbb{E}[dW_t^{(m,i)} dW_t^{(m,j)}] = \rho_{ij}\, dt.$$

The parameters are defined as follows:

- $S_t^{(m,i)}$ is the price of the i-th asset at time t in sample m,
- μ_i is the drift coefficient for asset i,
- ν_i is the volatility of asset i,
- $\rho_{ij} \in [-1, 1]$ is the instantaneous correlation between the Brownian motions of assets i and j.

In a sample m, each marginal process $S_t^{(m,i)}$ follows the same analytical solution as in the univariate GBM case (Eq. 3.2):

$$S_t^{(m,i)} = S_0^{(m,i)} \exp\left[\left(\mu_i - \frac{1}{2}\nu_i^2\right) t + \nu_i W_t^{(m,i)}\right]. \tag{3.5}$$

To simulate correlated paths, we first define the correlation matrix $\Sigma = (\rho_{ij})_{i,j=1}^d \in \mathbb{R}^{d \times d}$, where each entry ρ_{ij} specifies the instantaneous correlation between Brownian motions $W_t^{(m,i)}$ and $W_t^{(m,j)}$. We then sample a vector of independent standard normal variables

$\mathbf{Z} \sim \mathcal{N}(\mathbf{0}, I_d)$ and apply a transformation $\tilde{\mathbf{Z}} = L\mathbf{Z}$, where L is a lower triangular matrix such that $LL^\top = \Sigma$. The discretized sampling formula becomes:

$$S_t^{(m,i)} = S_{t-1}^{(m,i)} \cdot \exp\left[\left(\mu_i - \frac{1}{2}\nu_i^2\right)\Delta t + \nu_i\sqrt{\Delta t} \cdot \tilde{Z}_i\right], \tag{3.6}$$

where $\tilde{\mathbf{Z}} = [\tilde{Z}_1, \ldots, \tilde{Z}_d]^\top \sim \mathcal{N}(\mathbf{0}, \Sigma)$ and $\Delta t = T/N$.

This formulation allows for the efficient simulation of multivariate asset price trajectories with a specified correlation structure. As described in Algorithm 2, we use MC simulation to generate the dataset, introducing an additional dimension d representing the number of correlated features per sample. To capture the correlations between assets, we apply Cholesky decomposition to the correlation matrix, obtaining a lower triangular matrix used to transform independent standard normal variables into correlated ones. Figure 3.2 illustrates an example of two negatively correlated asset price paths simulated using the CGBM model.

Algorithm 2 Simulate Correlated Geometric Brownian Motion Paths

Require: Number of samples M, time steps N, features d, drift vector $\mu \in \mathbb{R}^d$, volatility vector $\nu \in \mathbb{R}^d$, initial price $S_0 \in \mathbb{R}^d$, correlation matrix $\Sigma \in \mathbb{R}^{d\times d}$, time horizon T

Compute L such that $\Sigma = LL^\top$ ▷ Cholesky decomposition
$\Delta t \leftarrow T/N$ ▷ Time increment
for $m = 1$ to M **do** ▷ Paths loop
  $S_0^{(m)} \leftarrow S_0$ ▷ Initialize prices
  for $t = 1$ to N **do** ▷ Time loop
    $Z \sim \mathcal{N}(0, I_d)$ ▷ Sample noise
    $\tilde{Z} \leftarrow LZ$ ▷ Correlated noise
    for $i = 1$ to d **do** ▷ Feature loop
      $S_t^{(m,i)} \leftarrow S_{t-1}^{(m,i)} \times \exp\left[\left(\mu_i - \frac{1}{2}\nu_i^2\right)\Delta t + \nu_i\sqrt{\Delta t} \cdot \tilde{Z}_i\right]$ ▷ Update
    end for
  end for
end for
return $S \in \mathbb{R}^{M\times d\times N}$ ▷ Simulated dataset

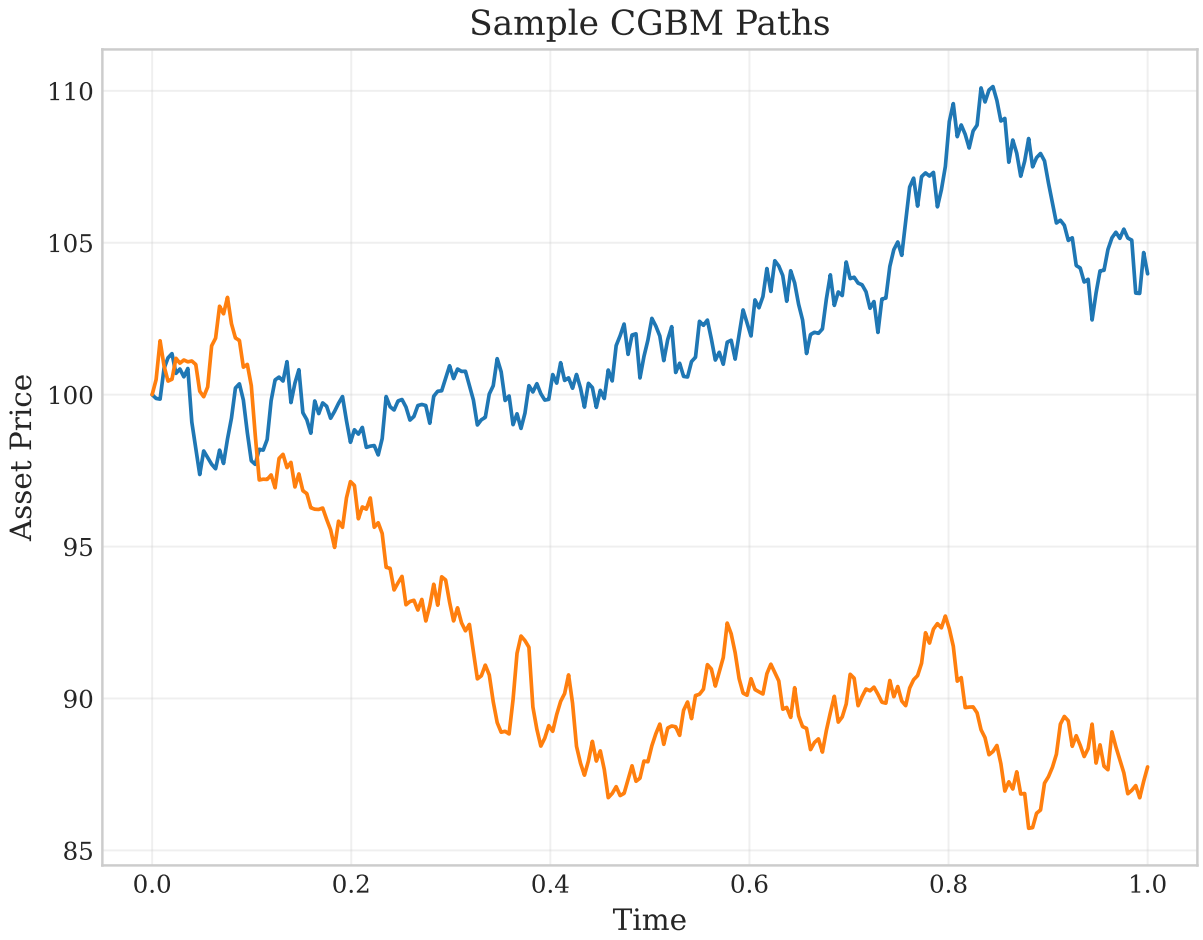


Figure 3.2: Sample trajectories of two correlated GBM processes simulated using the closed-form solution. The parameters used in the simulation are: $\mu = 0.05$, $\nu = 0.1$, $S_0 = 100$, $T = 1.0$, $N = 252$, and correlation $\rho = -0.8$.

Heston Stochastic Volatility Model

The Heston model introduces stochastic volatility into the asset price dynamics. It is defined by the following system of stochastic differential equations:

$$\begin{cases} dS_t = \mu S_t \, dt + \sqrt{v_t} S_t \, dW_t^{(1)} \\ dv_t = \kappa(\bar{v} - v_t) \, dt + \eta \sqrt{v_t} \, dW_t^{(2)} \end{cases} \tag{3.7}$$

where S_t denotes the asset price and v_t represents the instantaneous variance at time t. The asset price evolves similarly to a geometric Brownian motion, but with a time-varying volatility. The variance v_t follows a Cox–Ingersoll–Ross (CIR) process [13], which ensures that volatility remains non-negative.

The parameters of the model are:

- μ: drift of the asset price process,
- κ: rate of mean reversion of the variance process,
- $\bar{v}$: long-term mean level of the variance,
- η: volatility of volatility,
- ρ: correlation between the Brownian motions $W_t^{(1)}$ and $W_t^{(2)}$, such that:

$$dW_t^{(1)} dW_t^{(2)} = \rho \, dt.$$

To ensure the variance process remains strictly positive, the parameters must satisfy the Feller condition:

$$2\kappa\bar{v} > \eta^2.$$

The Heston model captures important stylized facts of financial time series, including volatility clustering and the leverage effect. Volatility clustering refers to the empirical observation that periods of high volatility are likely to be followed by high volatility, and periods of low volatility tend to be followed by low volatility. In the Heston model, this phenomenon arises naturally from the mean-reverting structure of the variance process, which follows a CIR dynamic. The leverage effect describes the asymmetric relationship between asset returns and future volatility: negative returns tend to increase volatility more than positive returns of the same magnitude. In the Heston model, this effect is

captured by a negative correlation parameter $\rho < 0$, which reflects the inverse relationship between the asset price and its instantaneous volatility.

The system of SDEs defining the Heston model does not have a closed-form analytical solution. Therefore, we simulate its dynamics numerically using a combination of methods. The variance process v_t, is discretized using the Euler–Maruyama method. For the asset price process S_t, we use the exact solution assuming piecewise constant variance over each time interval. Given time step Δt, and correlated standard normal random variables $Z_1, Z_2 \sim \mathcal{N}(0,1)$, the discretized update equations are:

$$\begin{aligned} v_t^{(m)} &= v_{t-1}^{(m)} + \kappa\big(\bar{v} - v_{t-1}^{(m)}\big)\Delta t + \eta\sqrt{v_{t-1}^{(m)}}\sqrt{\Delta t}\cdot Z_2, \\ S_t^{(m)} &= S_{t-1}^{(m)} \cdot \exp\left[\left(\mu - \tfrac{1}{2}v_{t-1}^{(m)}\right)\Delta t + \sqrt{v_{t-1}^{(m)}}\sqrt{\Delta t}\cdot Z_1\right], \end{aligned} \tag{3.8}$$

Following the approach in Algorithm 3, we generate a dataset of shape (M, N) where each sample represents a simulated asset price trajectory over N time steps. In the Heston model, the stochastic variance process is simulated concurrently with the asset price. However, the variance trajectories are not included in the final dataset and are not used during model training. They are only used in the simulation process. Figure 3.3 illustrates sample asset price paths along with their corresponding variance trajectories.

Merton Jump Diffusion Model

The Merton Jump Diffusion (MJD) model extends the classical GBM framework by incorporating discontinuous jumps in asset prices, capturing sudden large movements that are often observed in real financial markets. The model adds a jump component to the standard GBM dynamics, resulting in the following SDE:

$$dS_t = \mu S_t\, dt + \nu S_t\, dW_t + S_{t^-}(J-1)\, dP_t, \tag{3.9}$$

where:

- μ is the drift of the continuous component,
- ν is the volatility of the diffusion term,
- W_t is a standard Brownian motion,

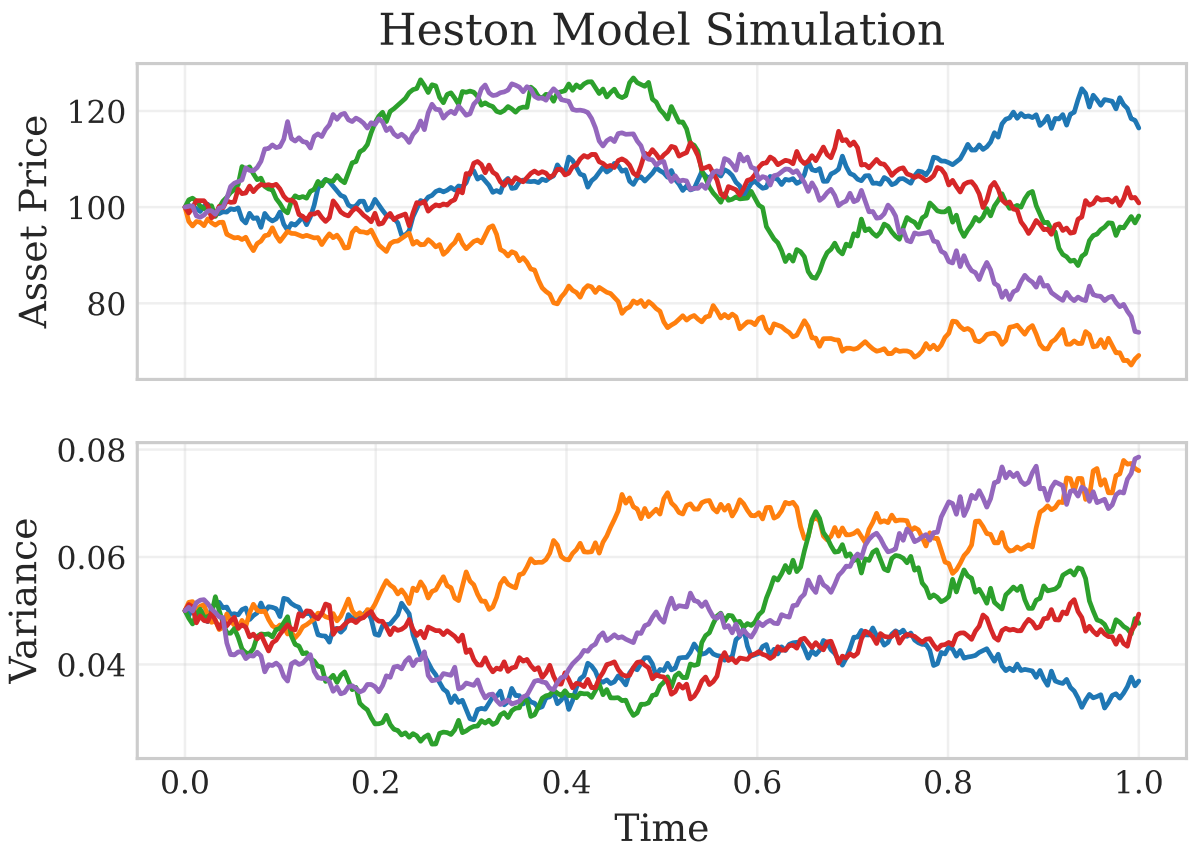


Figure 3.3: Sample simulations from the Heston stochastic volatility model. The top panel shows asset price trajectories, while the bottom panel shows the corresponding variance paths. Parameters used: $\mu = 0.05$, $\kappa = 1.0$, $\bar{v} = 0.05$, $\eta = 0.1$, $v_0 = 0.05$, $\rho = -0.8$, $S_0 = 100$, time horizon $T = 1.0$, and number of time steps $N = 252$.

Algorithm 3 Simulate Heston Model Paths

Require: Number of samples M, time steps N, drift μ, initial price S_0, initial variance v_0, mean reversion κ, long-term variance $\bar{v}$, volatility of variance η, correlation ρ, time horizon T

$\Delta t \leftarrow T/N$ ▷ Time increment
for $m = 1$ to M **do** ▷ Paths loop
  $S_0^{(m)} \leftarrow S_0$, $v_0^{(m)} \leftarrow v_0$ ▷ Initialize prices
  for $t = 1$ to N **do** ▷ Time loop
    $Z_1, Z_2 \sim \mathcal{N}(0,1)$ ▷ Sample noise
    $\tilde{Z}_1 \leftarrow \rho Z_2 + \sqrt{1-\rho^2}\, Z_1$ ▷ Correlated noise
    $v_t^{(m)} \leftarrow \max\left(0,\, v_{t-1}^{(m)} + \kappa(\bar{v} - v_{t-1}^{(m)})\Delta t + \eta\sqrt{v_{t-1}^{(m)}}\sqrt{\Delta t}\cdot Z_2\right)$ ▷ Variance update
    $S_t^{(m)} \leftarrow S_{t-1}^{(m)} \times \exp\left[\left(\mu - \frac{1}{2}v_{t-1}^{(m)}\right)\Delta t + \sqrt{v_{t-1}^{(m)}}\sqrt{\Delta t}\cdot \tilde{Z}_1\right]$ ▷ Price update
  end for
end for
return $S \in \mathbb{R}^{M\times N}$ ▷ Simulated dataset

- P_t is a Poisson process with intensity λ, representing the number of jumps up to time t,
- J is the jump size multiplier, such that $\log J \sim \mathcal{N}(\mu_J, \sigma_J^2)$.

The jump term $S_{t^-}(J-1)\,dP_t$ captures the multiplicative jumps occurring at random times governed by the Poisson process, where S_{t^-} is the asset price just before the jump. When a jump occurs, the asset price is instantaneously multiplied by a lognormally distributed factor J.

To simulate the MJD process numerically, we discretize time into intervals of size Δt, and the update rule becomes:

$$S_t^{(m)} = S_{t-1}^{(m)} \cdot \exp\left[\left(\mu - \frac{1}{2}\nu^2\right)\Delta t + \nu\sqrt{\Delta t}\cdot Z\right] \cdot \prod_{i=1}^{P} J_i, \tag{3.10}$$

where $Z \sim \mathcal{N}(0,1)$, $P \sim \text{Poisson}(\lambda\Delta t)$ is the number of jumps in the current interval, and J_i are i.i.d. jump multipliers sampled from $\log\mathcal{N}(\mu_J, \sigma_J^2)$. If no jumps occur in a given interval ($P = 0$), the product is defined as 1.

Following Algorithm 4, we simulate a dataset of shape (M, N). Figure 3.4 shows example trajectories with jumps.

Algorithm 4 Simulate Merton Jump Diffusion (MJD) Paths

Require: Number of samples M, time steps N, drift μ, volatility ν, jump intensity λ, jump mean μ_J, jump standard deviation σ_J, initial price S_0, time horizon T

$\Delta t \leftarrow T/N$ ▷ Time increment
for $m = 1$ to M **do** ▷ Paths loop
  $S_0^{(m)} \leftarrow S_0$ ▷ Initialize prices
  for $t = 1$ to N **do** ▷ Time loop
    $Z \sim \mathcal{N}(0, 1)$ ▷ Sample noise
    $P \sim \text{Poisson}(\lambda \Delta t)$ ▷ Sample number of jumps
    if $P > 0$ **then**
      $J_1, \ldots, J_P \sim \log \mathcal{N}(\mu_J, \sigma_J^2)$ ▷ Jump sizes
      $J \leftarrow \prod_{i=1}^{P} J_i$ ▷ Total jump
    else
      $J \leftarrow 1$ ▷ No jump
    end if
    $S_t^{(m)} \leftarrow S_{t-1}^{(m)} \times \exp\left[\left(\mu - \frac{1}{2}\nu^2\right)\Delta t + \nu\sqrt{\Delta t} \cdot Z\right] \times J$ ▷ Update
  end for
end for
return $S \in \mathbb{R}^{M \times N}$ ▷ Simulated dataset

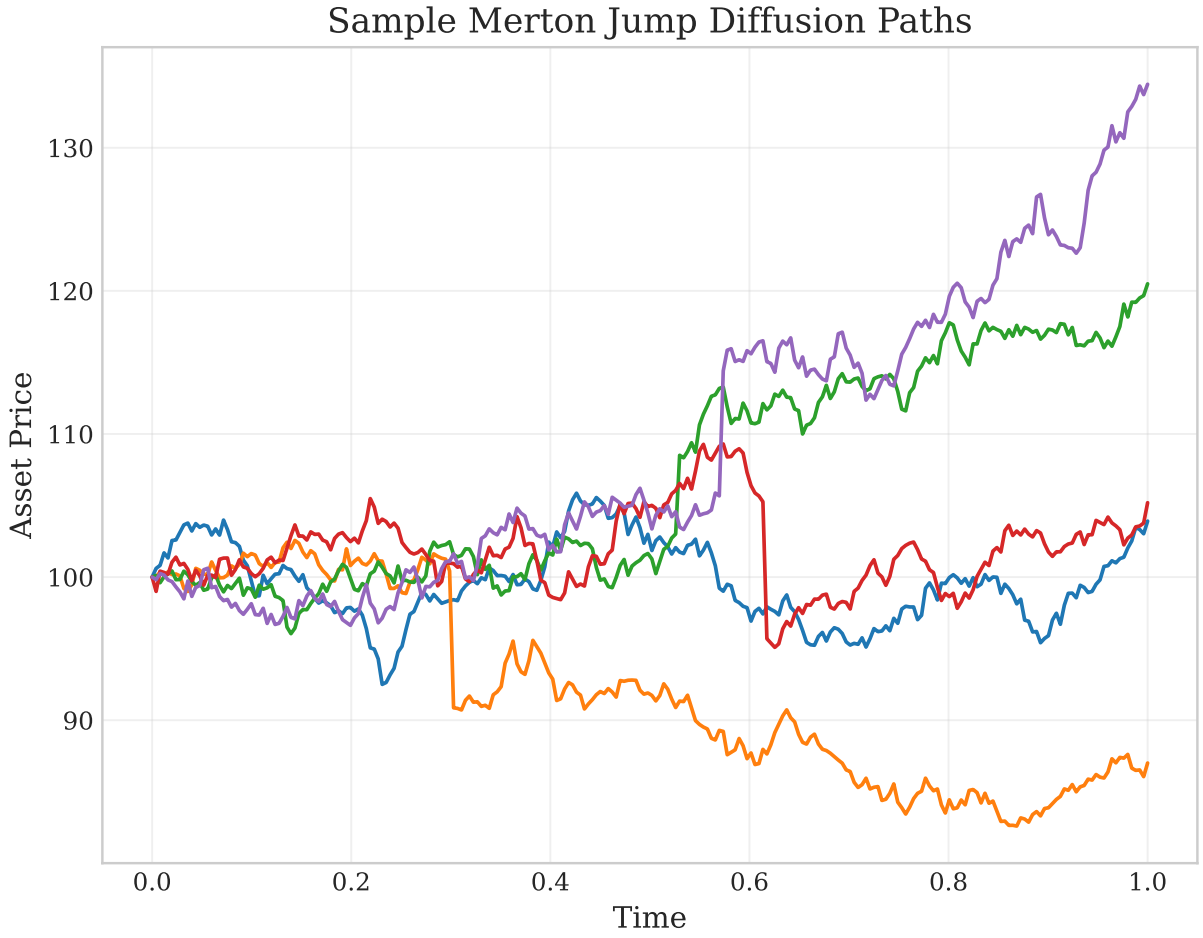


Figure 3.4: Sample simulations from the Merton Jump diffusion model. Parameters used: $\mu = 0.05$, $\nu = 0.1$, $\mu_J = -0.05$, $\sigma_J = 0.1$, $\lambda = 0.5$, $S_0 = 100$, time horizon $T = 1.0$, and number of time steps $N = 252$. Noticeable negative jumps occur around $t \approx 0.03$ and $t \approx 0.6$.

3.2.2 Real World Market Data

The real-world market data used in this work consists of historical asset prices retrieved from Yahoo Finance. We construct several types of training datasets, each tailored to support a different downstream task. Specifically, we consider two variants: a single-asset dataset for modeling individual asset dynamics and a dataset based on OHLC (open, high, low, close) prices to incorporate richer market structure and intra-period behavior. Detailed descriptions of each variant are provided in the following subsections.

Single-Asset Dataset

The single-asset dataset consists of daily adjusted closing prices for a given asset. The historical prices are initially represented as a one-dimensional array of length L, where L is the total number of available daily observations. A non-overlapping sliding window of size N is then applied resulting in a collection of $M = \lfloor L/N \rfloor$ windows. After this segmentation, the resulting training tensor has shape (M, d, N) with $d = 1$ for the single-feature channel:

$$\text{shape} = (M,\, d,\, N) = (M,\, 1,\, N), \qquad M = \left\lfloor \tfrac{L}{N} \right\rfloor.$$

With this preprocessing, each m-th subsequence, for $m = 1, \ldots, M$, is treated as an independent sample path of length N. This makes the dataset shape conceptually identical to that of the synthetic datasets described earlier.

Training on a single-stock dataset is beneficial when the objective is to model the temporal dynamics of a specific asset of interest. That enables the generative model to capture asset-specific return patterns, volatility behavior, and other characteristics.

Multivariate Market Data: OHLC

Multivariate market data refers to daily Open-High-Low-Close (OHLC) prices of an asset. Such data can be constructed from the historical price records of either a single asset or multiple assets. Each sample in this dataset contains $d = 4$ correlated features corresponding to the OHLC components, observed over N time steps, resulting in a dataset of shape (M, d, N), where M is the number of samples. OHLC data is typically visualized using a candlestick chart, where each "candlestick" represents one time step (e.g., a day). The rectangular body spans from the open to the close prices, with color indicating whether the asset closed higher (green) or lower (red) than it opened. Thin vertical lines extend



Figure 3.5: Candlestick chart of the S&P 500 index showing daily OHLC prices over a one-year period. Each candlestick captures the open, high, low, and close prices for a single trading day.

from the body to the high and low prices of the day, capturing intra-period price extremes. Figure 3.5 displays the daily OHLC price data of the S&P 500 index over a one-year period.

Modeling time series with this structure is particularly useful in trading applications, where richer market signals are required. OHLC-format data enables more realistic modeling for tasks such as strategy backtesting, trade signal generation, and market behavior simulation.

Table 3.1 provides an overview of all datasets used in this work, including both synthetic datasets generated from financial models and real-world datasets collected from market data. For each dataset type, the table reports the shape of the resulting data tensor.

3.2.3 Data Preprocessing

Raw asset price data, whether empirical or simulated, possesses several characteristics that make it challenging to model directly. Since prices represent absolute asset values, they are scale-dependent, exhibit strong autocorrelation (i.e., today's price depends on yesterday's),

Table 3.1: Summary of Synthetic and Real-World Datasets Used

Dataset Type	**Shape**	**Features**
Synthetic		
Geometric Brownian Motion (GBM)	$(M, 1, N)$	Single asset price
Correlated GBM (CGBM)	(M, d, N)	d correlated assets
Heston Stochastic Volatility	$(M, 1, N)$	Single asset price
Merton Jump Diffusion (MJD)	$(M, 1, N)$	Single asset price
Real-World		
Single-Stock	$(M, 1, N)$	Single asset price
OHLC (Open-High-Low-Close)	$(M, 4, N)$	OHLC prices

and often follow persistent trends. These properties contribute to the non-stationarity of financial time series.

In the context of data modeling, non-stationarity poses a challenge because key statistical properties, such as the mean, variance, and autocorrelation, can change over time. To address these limitations, it is common practice to apply transformations that make the time series approximately stationary.

Weak (Second-Order) Stationarity. A stochastic process $\{X_t\}_{t\in\mathbb{Z}}$ is said to be *weakly stationary* (or second-order stationary) if it satisfies the following conditions:

1. $\mathbb{E}[X_t] = \mu \quad \forall t \in \mathbb{Z}$ (constant first moment),
2. $\mathbb{E}[X_t^2] < \infty \quad \forall t \in \mathbb{Z}$ (finite second moment),
3. $\mathrm{Cov}(X_r, X_s) := \gamma_X(r, s) = \gamma_X(t + r, t + s), \quad \forall\, t, r, s \in \mathbb{Z}$ (shift invariant covariance)

In other words, the mean is time-invariant, the variance is finite, and the covariance depends only on the lag between observations, and not on the specific time indices.

Logarithmic returns

A typical transformation in financial time series analysis is the logarithmic return, defined as:

$$r_t = \ln\left(\frac{S_t}{S_{t-1}}\right) \tag{3.11}$$

where S_t denotes the asset price at time t. We refer to this transformation as *log returns* throughout the remainder of this work.

Log returns measure relative (percentage) changes in asset price, making them scale-independent. They are commonly assumed to be weakly stationary, with constant mean, finite variance, and autocovariance that depends only on the lag. Figure 3.6 demonstrates the effect of the log return transformation on a typical financial time series. In many practical scenarios, it is also reasonable to approximate the distribution of log returns as Gaussian (e.g., in the Black–Scholes model). This assumption is particularly relevant in the context of generative diffusion models and will be revisited in later chapters.

We can now expand the notation established in Section 3.1: $\mathcal{D}_{\text{train}} = \{X^{(i)}\}_{i=1}^{M}$ and $\mathcal{D}_{\text{gen}} = \{\widetilde{X}^{(i)}\}_{i=1}^{M}$ denote the training and generated datasets, respectively, where each sample consists of log return sequences. The corresponding price series reconstructed from these log returns are denoted by $\mathcal{P}_{\text{train}} = \{S^{(i)}\}_{i=1}^{M}$ and $\mathcal{P}_{\text{gen}} = \{\widetilde{S}^{(i)}\}_{i=1}^{M}$. Price series are recovered by exponentiating cumulative log returns, anchored to a reference initial value S_0:

$$S_t = S_0 \cdot \exp\left(\sum_{k=1}^{t} r_k\right) \tag{3.12}$$

Before training, it is beneficial to normalize the log return series. A standard choice is z-score normalization (also called standardization), which uses the sample mean and variance of the training dataset. It ensures that the transformed series has zero mean and unit variance, which is beneficial for stable and efficient neural network training. The standardized log return z_t is defined as:

$$z_t = \frac{r_t - \hat{\mu}}{\hat{\sigma}} \tag{3.13}$$

where $\hat{\mu}$ and $\hat{\sigma}$ denote the sample mean and standard deviation of the log returns, respectively. These statistics are computed globally across the entire training dataset. This global normalization is preserved during training and used during the reverse transformation to recover log return values. The reverse transformation (denormalization) is given by:

$$r_t = z_t \cdot \hat{\sigma} + \hat{\mu} \tag{3.14}$$

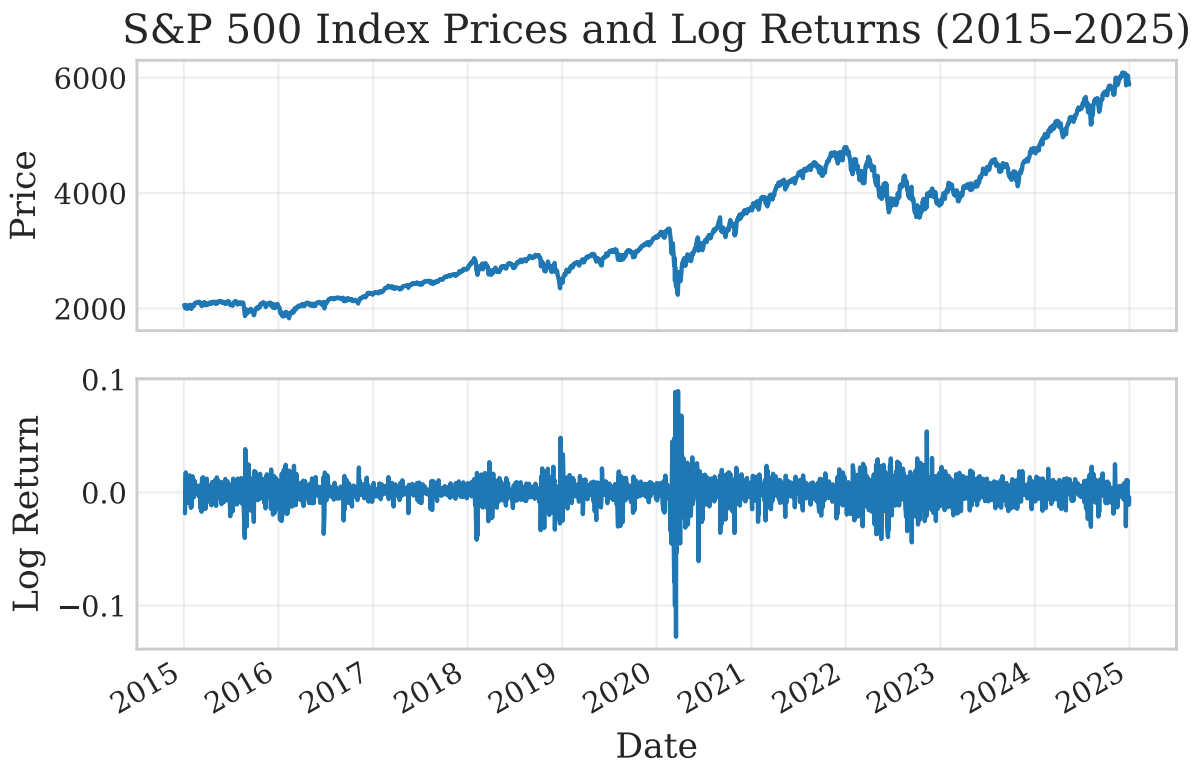


Figure 3.6: S&P 500 index prices and corresponding log returns from 2015 to 2025. The top panel shows the raw price series, while the bottom panel displays the log returns.

3.3 Base model: EDM

In this subsection, we describe the diffusion model architecture that serves as the baseline and starting point for all subsequent developments presented in this work.

All models in this thesis are built on top of the Elucidated Diffusion Model (EDM) framework [37]. As introduced in Chapter 2, diffusion models are a class of generative models that learn data distributions by reversing a noise-adding process through score function estimation. The EDM formulation extends this general framework by providing clear design principles for parameterization, training, and sampling. In particular, EDM differs from standard diffusion models such as DDPM [32] or score-based SDE approaches [71] in four key ways: (i) it reparameterizes noise levels in a manner that improves coverage of both low- and high-noise regimes, (ii) it introduces explicit preconditioning that stabilizes training across a wide range of noise scales, (iii) it unifies commonly used loss functions under a systematic noise-dependent weighting framework, and (iv) it shows that different samplers can be interpreted as numerical solvers of the same reverse-time dynamics in the noise parameter $\sigma(t)$. EDM provides a generalized notation and implementation under which previous approaches, such as DDPM [32], SMLD [68], and DDIM [56], can be treated as special cases or parameterizations of EDM. This makes EDM a natural choice as the baseline framework for our work.

Another advantage of EDM is its modularity. It separates the core building blocks of a diffusion model (backbone neural network, loss function, training procedure, sampler, and preconditioning) so that individual components can be studied and modified in isolation. In what follows, we describe each of these components as used in our experiments.

3.3.1 Loss function

EDM uses a weighted denoising score matching loss [37] that balances training across different noise levels. The loss function is formulated as:

$$\mathcal{L}(\theta) = \mathbb{E}_{t,X_0,Z}\left[\lambda(\sigma(t))\left\|D_\theta(X_0 + \sigma(t)Z, \sigma(t)) - X_0\right\|_2^2\right], \tag{3.15}$$

where D_θ is the neural network parameterized by weights θ, $X_0 \sim p_{\text{data}}$ is a clean training sample, $Z \sim \mathcal{N}(0, I)$ is standard Gaussian noise, and t is sampled such that $\sigma(t)$ follows the training noise distribution $p_{\text{train}}(\sigma)$. The noise-level dependent weighting function is defined as

$$\lambda(\sigma(t)) = \frac{\sigma^2(t) + \sigma_{\text{data}}^2}{\big(\sigma(t) \cdot \sigma_{\text{data}}\big)^2},$$

where σ^2_{data} denotes the variance of the input data distribution.

Here, $\sigma(t)$ controls the amount of corruption applied during training. The weighting function rescales the loss so that errors are normalized across noise levels, ensuring that neither low-noise nor high-noise regimes dominate optimization. We refer to [37] for the exact weighting form derivation. The objective is therefore to train the network D_θ to approximate the conditional expectation $\mathbb{E}[X_0 \mid X_t]$, which is the optimal denoiser by Theorem 2.3.1.

3.3.2 Preconditioning

In EDM, the denoiser is not trained directly as an arbitrary network $D_\theta(x;\sigma)$. Instead, EDM introduces a preconditioning scheme that expresses D_θ in terms of a raw neural network F_θ together with carefully chosen scaling functions. This ensures stable training dynamics across all noise levels σ, which are sampled from a distribution p_{train} described in Section 3.3.3.

Formally, the preconditioned denoiser is defined as

$$D_\theta(x;\sigma) = c_{\text{skip}}(\sigma)\, x \;+\; c_{\text{out}}(\sigma)\, F_\theta(c_{\text{in}}(\sigma)\, x;\; c_{\text{noise}}(\sigma))\,, \tag{3.16}$$

where F_θ is the raw neural network and the scaling coefficients are given by

$$c_{\text{in}}(\sigma) = \frac{1}{\sqrt{\sigma^2 + \sigma^2_{\text{data}}}}, \tag{3.17}$$

$$c_{\text{out}}(\sigma) = \frac{\sigma \cdot \sigma_{\text{data}}}{\sqrt{\sigma^2 + \sigma^2_{\text{data}}}}, \tag{3.18}$$

$$c_{\text{skip}}(\sigma) = \frac{\sigma^2_{\text{data}}}{\sigma^2_{\text{data}} + \sigma^2}, \tag{3.19}$$

$$c_{\text{noise}}(\sigma) = \tfrac{1}{4}\ln(\sigma). \tag{3.20}$$

Each coefficient serves a specific role:

- $c_{\text{in}}(\sigma)$ rescales the noisy input so that it has approximately unit variance across noise levels. If $X = X_0 + \sigma Z$ with $\text{Var}(X_0) = \sigma^2_{\text{data}}$, then $\text{Var}(X) = \sigma^2_{\text{data}} + \sigma^2$.Dividing by $\sqrt{\sigma^2_{\text{data}} + \sigma^2}$ therefore normalizes the variance to 1.
- $c_{\text{out}}(\sigma)$ rescales the network output so that the effective training targets also have unit variance, similarly to c_{in} .

- $c_{\text{skip}}(\sigma)$ controls how much of the noisy input is passed directly to the output through a skip connection.
- $c_{\text{noise}}(\sigma)$ rescales the noise level before passing it into the network.

We refer to [37] for the derivation details of the preconditioning coefficients.

Taking this reparameterization into account, the training objective Eq. (3.15) can be equivalently expressed in terms of the raw network F_θ as

$$\begin{aligned}\mathcal{L}(\theta) = \mathbb{E}_{\sigma, X_0, Z}\Big[\lambda(\sigma)\, c_{\text{out}}(\sigma)^2 \Big\| F_\theta\big(c_{\text{in}}(\sigma)(X_0 + \sigma Z),\, c_{\text{noise}}(\sigma)\big) \\ - \tfrac{1}{c_{\text{out}}(\sigma)}\Big(X_0 - c_{\text{skip}}(\sigma)(X_0 + \sigma Z)\Big)\Big\|_2^2\Big],\end{aligned} \tag{3.21}$$

where σ is understood as a function of t, with full notation omitted for brevity.

This reformulation clarifies the role of the skip connection. When $c_{\text{skip}}(\sigma) \approx 0$ (large noise levels), the effective target reduces to the clean sample X_0, so the network is trained to reconstruct the underlying signal from heavily corrupted inputs. When $c_{\text{skip}}(\sigma) \approx 1$ (small noise levels), the target is proportional to the noise σZ, so the network is trained to learn the residual noise in lightly corrupted inputs. For intermediate values, the target is a mixture of signal and noise.

In this way, preconditioning ensures that inputs and targets have consistent variance across all noise levels, prevents error amplification at large σ, and provides the network with the flexibility to learn the most stable prediction target at each noise level.

We emphasize that Eq. (3.15) and Eq. (3.21) are mathematically equivalent formulations of the same objective. Equation (3.15) expresses the loss in terms of the preconditioned denoiser D_θ, while Eq. (3.21) rewrites it in terms of the raw network F_θ. Throughout this thesis, D_θ should be understood as shorthand for the preconditioned network defined in Eq. (3.16).

3.3.3 Training

EDM's training procedure incorporates several key improvements over traditional diffusion model training approaches. The framework uses a novel noise sampling strategy compared to earlier methods. Instead of sampling noise levels uniformly, EDM uses a log-normal distribution:

$$\ln(\sigma) \sim \mathcal{N}(P_{\text{mean}}, P_{\text{std}}^2) \tag{3.22}$$

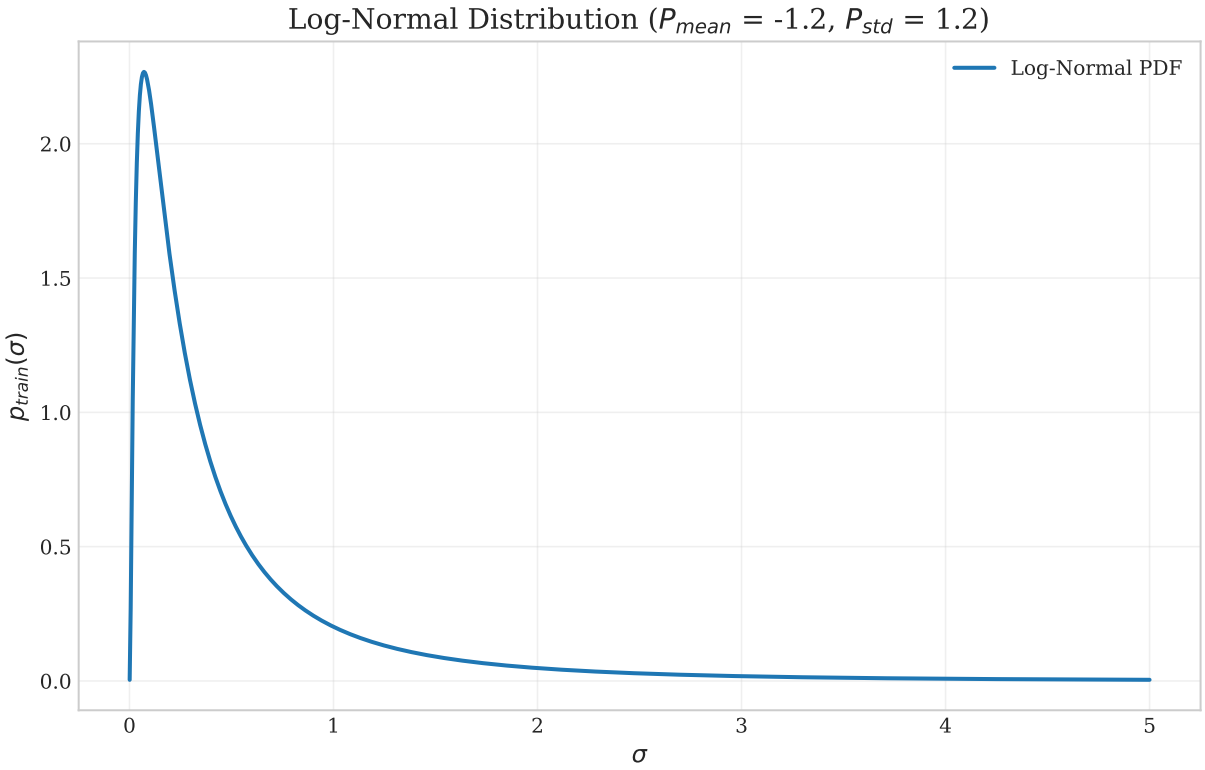


Figure 3.7: Probability density function of noise levels used in training. Most of the values are centered around $\sigma \approx 0.1$.

where $P_{\text{mean}} = -1.2$ and $P_{\text{std}} = 1.2$. We denote the corresponding probability density function of σ by $p_{\text{train}}(\sigma)$. This distribution concentrates training effort on relevant noise levels, avoiding excessive focus on extreme noise regimes that contribute little to final sample quality. Figure 3.7 demonstrates the shape of the distribution and the range of possible values.

The training algorithm samples clean data from the dataset and corrupts it with appropriately scaled Gaussian noise. For each training iteration, a clean sample $X_0 \sim p_{\text{data}}$ is drawn from the empirical data distribution defined by the training dataset, a noise level $\sigma \sim p_{\text{train}}(\sigma)$ is sampled from the log-normal distribution defined in (3.22), noisy input is then created as $X_0 + \sigma Z$, where $Z \sim \mathcal{N}(0, I)$ is standard Gaussian noise. The network is then trained to denoise this corrupted sample using the weighted loss function described in previous sections.

Optimization is performed using the standard Adam [43] optimization with exponential moving averages of model parameters for improved stability and performance, following [37].

Algorithm 5 EDM Training

Require: Dataset $\mathcal{D}$, network F_θ, parameters σ_{data}, $P_{\text{mean}} = -1.2$, $P_{\text{std}} = 1.2$

function Precondition(σ) ▷ Compute scaling coefficients
  $c_{\text{skip}} \leftarrow \frac{\sigma_{\text{data}}^2}{\sigma^2+\sigma_{\text{data}}^2}$
  $c_{\text{out}} \leftarrow \frac{\sigma \cdot \sigma_{\text{data}}}{\sqrt{\sigma^2+\sigma_{\text{data}}^2}}$
  $c_{\text{in}} \leftarrow \frac{1}{\sqrt{\sigma^2+\sigma_{\text{data}}^2}}$
  $c_{\text{noise}} \leftarrow \frac{1}{4}\ln\sigma$
  return $c_{\text{skip}}, c_{\text{out}}, c_{\text{in}}, c_{\text{noise}}$
end function
while not converged **do** ▷ Training loop
  $X_0 \sim \mathcal{D}$ ▷ Sample clean data
  $\ln\sigma \sim \mathcal{N}(P_{\text{mean}}, P_{\text{std}}^2)$ ▷ Sample log noise level
  $\sigma \leftarrow \exp(\ln\sigma)$
  $Z \sim \mathcal{N}(0, I)$ ▷ Sample standard noise
  $X_t \leftarrow X_0 + \sigma Z$ ▷ Corrupt data
  $c_{\text{skip}}, c_{\text{out}}, c_{\text{in}}, c_{\text{noise}} \leftarrow$ Precondition(σ)
  $\hat{X}_0 \leftarrow c_{\text{skip}} \cdot X_t + c_{\text{out}} \cdot F_\theta(c_{\text{in}} \cdot X_t; c_{\text{noise}})$ ▷ Denoised prediction
  $\lambda(\sigma) \leftarrow \frac{\sigma^2+\sigma_{\text{data}}^2}{(\sigma \cdot \sigma_{\text{data}})^2}$ ▷ Weighting
  $\mathcal{L} \leftarrow \lambda(\sigma) \cdot \|\hat{X}_0 - X_0\|_2^2$ ▷ Loss
  Update θ using $\nabla_\theta \mathcal{L}$ ▷ Gradient step
end while
return θ

3.3.4 Backbone Neural Network

EDM is architecture-agnostic and can work with various U-Net-based neural network architectures [63]. In our implementation, we use the SongUNet architecture with NCSN++ configuration [71]. SongUNet uses a standard U-Net structure with encoder-decoder symmetry. Through downsampling operations, the encoder progressively reduces spatial resolution while expanding the number of feature channels, and the decoder reverses this

process to reconstruct the output through upsampling. The core building block is the UNetBlock, which combines group normalization, residual connections, and conditioning mechanisms.

The architecture processes data at multiple scales through a hierarchical approach. As the network moves deeper into the encoder, the spatial dimensions decrease while the number of channels increases according to predefined multipliers. Self-attention blocks [74] are placed at intermediate scales to capture long-range dependencies, balancing context with computational efficiency. This multi-scale design enables the network to capture both fine-grained details and broader structural patterns.

The architecture supports timestep conditioning through positional embeddings, which inform the network about the current noise level σ_t during the diffusion process. Additionally, it can incorporate class labels for conditional generation and other auxiliary information when needed.

Since the original SongUNet model was designed for image processing tasks, we propose our adaptation for time series data in Section 3.4.

3.3.5 Sampling

For our experiments, we use EDM's deterministic sampling procedure, which is based on solving the probability flow ODE (2.10) introduced by Song et al. [71]. In the case of the variance-exploding (VE) formulation (2.5), this ODE takes the form

$$\frac{dx}{dt} = -\dot{\sigma}(t)\,\sigma(t)\,\nabla_x \log p(x;\sigma(t)), \tag{3.23}$$

where $\sigma(t)$ is the same noise parameter introduced during training (Section 3.3.3), now parameterized continuously as a function of time t. Note that during training, there is no particular ordering of noise levels σ; only their distribution matters. In contrast, during sampling, we integrate from $t = T$ (large noise, $\sigma \approx \sigma_{\max}$) to $t = 0$ (almost clean, $\sigma \approx \sigma_{\min}$). Thus, the noise level must be expressed as a time-dependent function $\sigma(t)$. As discussed by Karras et al. [37], a common choice is simply $\sigma(t) = t$, which makes $\sigma(t)$ and t interchangeable and simplifies the ODE (3.23) to

$$\frac{dx}{dt} = \frac{x - D_\theta(x;\sigma(t))}{t}, \tag{3.24}$$

where the score function $\nabla_x \log p(x;\sigma(t))$ is approximated with a neural network D_θ via Tweedie's formula (Theorem 2.3.1).

In practice, numerical integration requires discretization. The continuous trajectory $\sigma(t)$ is represented by a finite sequence of timesteps $\{t_i\}_{i=0}^{N-1}$, with $\sigma_i = \sigma(t_i) = t_i$ since we set $\sigma(t) = t$. EDM defines these timesteps directly using a polynomial schedule:

$$\sigma_i = \left(\sigma_{\max}^{1/\rho} + \frac{i}{N-1}\left(\sigma_{\min}^{1/\rho} - \sigma_{\max}^{1/\rho}\right)\right)^{\rho}, \tag{3.25}$$

where $\rho = 7$, $\sigma_{\max} = 80$, and $\sigma_{\min} = 0.002$, chosen following [37] to balance truncation error across noise levels. This schedule places more steps at lower noise levels, where fine details of the signal are recovered, and fewer steps at high noise levels, where only coarse structure is determined.

To integrate the ODE (3.24), EDM employs Heun's second-order method, also known as the improved Euler scheme. At each step, the method first computes an Euler update to estimate the next state, then evaluates the derivative at this predicted point and averages it with the initial derivative for the final update. This two-stage procedure reduces truncation error compared to the simple Euler scheme and leads to higher sample quality with fewer steps [37]. The complete sampling procedure is provided in Algorithm 6.

3.4 Adapting EDM from Images to Time Series

The original EDM was designed for image generation tasks, where both inputs and outputs are typically RGB images with spatial structure. To apply EDM to time series data, several modifications are necessary to account for the temporal data specifics. These include adjustments to the model architecture, input representation, and sampling procedure, all of which represent a part of our contributions and are discussed in the following subsections.

3.4.1 Data Preprocessing

In the original EDM implementation for images, input data is normalized before training by scaling each RGB channel to a fixed range, typically either $[0, 1]$ or $[-1, 1]$. To mimic this preprocessing step, while accounting for the statistical properties of log return data, we standardize the training data using z-score normalization to have zero mean and unit variance, as described in Section 3.2.3.

Algorithm 6 EDM Deterministic Sampling

Require: Network D_θ, steps N, $\sigma_{\min} = 0.002$, $\sigma_{\max} = 80$, $\rho = 7$
Ensure: Generated sample X

function NoiseSchedule($N, \sigma_{\min}, \sigma_{\max}, \rho$)
for $i = 0$ to $N - 1$ **do**
    $\sigma_i \leftarrow \left(\sigma_{\max}^{1/\rho} + \frac{i}{N-1}\left(\sigma_{\min}^{1/\rho} - \sigma_{\max}^{1/\rho}\right)\right)^\rho$
end for
$\sigma_N \leftarrow 0$
return $\{\sigma_0, \sigma_1, \ldots, \sigma_N\}$
end function

$\{\sigma_0, \sigma_1, \ldots, \sigma_N\} \leftarrow$ NoiseSchedule($N, \sigma_{\min}, \sigma_{\max}, \rho$)
Sample $Z \sim \mathcal{N}(0, I)$ ▷ Initial noise
$X \leftarrow \sigma_0 \cdot Z$ ▷ Initialize at $\sigma_{\max}$
for $i = 0$ to $N - 1$ **do** ▷ Sampling loop
    $d \leftarrow \frac{X - D_\theta(X; \sigma_i)}{\sigma_i}$ ▷ Euler direction
    $X_{\text{euler}} \leftarrow X + (\sigma_{i+1} - \sigma_i)\, d$
    if $\sigma_{i+1} > 0$ **then** ▷ Second-order correction
        $d' \leftarrow \frac{X_{\text{euler}} - D_\theta(X_{\text{euler}}; \sigma_{i+1})}{\sigma_{i+1}}$
        $X \leftarrow X + (\sigma_{i+1} - \sigma_i)\, \frac{d + d'}{2}$
    else
        $X \leftarrow X_{\text{euler}}$ ▷ Final Euler step
    end if
end for
return X

3.4.2 Mapping Images to Time Series

For humans, images and time series data are conceptually and visually distinct structures. From a modeling perspective, however, they share more similarities than it seems at first glance. Images are typically represented as tensors of shape (c, w, h), where c denotes the number of color channels (typically 3 for RGB or 1 for grayscale), and w, h correspond to the image width and height in pixels, respectively. A multivariate time series sample, on the other hand, is typically represented as a tensor of shape (d, N), where d is the number of correlated features and N is the number of time steps.

In our implementation, we propose the following mapping from image data to time series:

1. Image width and height are treated as structurally equivalent in the original EDM pipeline, meaning they can be switched and that will not affect the training process. Since time series data require one fewer dimension, we collapse the height dimension by setting $h = 1$, flattening the 2D spatial input into a 1D temporal structure.

2. The image width dimension w is mapped directly to the time series length N, representing the temporal axis. No changes are required in this dimension.

3. The channel dimension c is mapped to the time series feature dimension d, extending the accepted range of values from $c \in \{1, 3\}$ to arbitrary $d \geq 1$. This includes both univariate ($d = 1$) and multivariate ($d > 1$) time series. The intuition behind this choice is that correlated time series features (e.g., correlated asset prices) behave similarly to correlated image channels (e.g., RGB), as they are processed jointly but not spatially ordered. In other words, the model is invariant to any internal ordering of this dimension (i.e., RGB $\equiv$ BGR). Likewise, correlated asset price vectors are treated equivalently by the model ($[S_t^1, S_t^2] \equiv [S_t^2, S_t^1]$) .

This mapping allows us to reuse the image-based diffusion model architectures for time series data with minimal modifications by treating a time series as a single-channel or multi-channel one-dimensional "image".

Summary of Mapping:

$$\begin{aligned} \text{Image tensor:} \quad & x \in \mathbb{R}^{c \times w \times h} \\ \text{Time series tensor:} \quad & x' \in \mathbb{R}^{d \times N \times 1} \\ \text{Mapping:} \quad & c \mapsto d, \quad w \mapsto N, \quad h \mapsto 1 \end{aligned}$$

3.4.3 SongUNet Architectural Adaptation from 2D to 1D

Following the conceptual mapping from images to time series described above, the SongUNet architecture required modifications to accommodate the dimensional transformation from (c, w, h) to $(d, N, 1)$.

The primary adaptation involved replacing all 2D convolutional operations with their 1D counterparts to operate along the temporal dimension N rather than spatial dimensions w, h . This change affected both standard convolutions and the up/downsampling operations used for multi-scale processing in the UNet architecture. The convolutional kernels and resampling filters were reduced from 2D to 1D, operating along the temporal sequence rather than across spatial dimensions. The self-attention mechanism was modified in a similar fashion to operate on sequential data. Throughout the network, tensor operations were adjusted to the reduced dimensionality. This included modifications to normalization layers, skip connections, and the embedding injection mechanism.

These modifications preserved the architectural characteristics that make SongUNet effective for diffusion modeling, while making the architecture suitable for time series data as defined by our dimensional mapping.

3.5 Ambient Diffusion

3.5.1 Introduction

Most generative models, including diffusion-based, require a large amount of high-quality training data [24, 46]. In real-world scenarios, however, clean data can be limited, while low-quality or corrupted data, such as noisy, blurry, or downsampled samples, are far more available. Traditional diffusion models have generally disregarded such data, following the principle of "garbage in, garbage out". This effect has been empirically demonstrated: state-of-the-art diffusion models perform poorly when trained on corrupted data, as they are unable to distinguish between the intrinsic data characteristics and external corruption. Instead, they tend to learn both the signal and the corruption [15, 18].

Recent research has addressed this challenge by proposing methods to effectively utilize corrupted data [76, 38, 6, 17]. One of the most promising of these approaches is *Ambient Diffusion* [18], a framework that extends diffusion models to handle imperfect data through a modified training objective. In particular, Ambient Diffusion introduces an Ambient Denoising Score Matching (ADSM) loss, which allows corrupted samples to be injected at

later stages of the diffusion process, where fine-grained detail is less important and the effects of corruption can be ignored, but the samples still carry useful information about coarse structure.

Classical diffusion models assume access to clean training samples drawn from the target distribution p_0. Ambient Diffusion relaxes this assumption by enabling training using only corrupted observations, under the condition that the corruption process is known. With this condition, Ambient Diffusion can recover the underlying clean distribution even when no clean samples are available during training. This approach is particularly effective when combined with even small quantities of clean data. As demonstrated in recent work [15], training with a mixture of just 10% clean and 90% noisy data achieves nearly the same performance as training on 100% clean data, while substantially outperforming models trained on either limited (10%) clean or corrupted samples alone.

3.5.2 Problem Formulation

In the Ambient Diffusion setting, we assume access to a training dataset consisting only of corrupted observations, rather than clean samples. For simplicity, we focus on additive Gaussian corruption, although the framework extends to more general linear noise models.

Instead of clean samples $X_0 \sim p_{\text{data}}$, we observe only corrupted samples during training:

$$X_{t_n} = X_0 + \sigma(t_n)Z \tag{3.26}$$

where $Z \sim \mathcal{N}(0, I)$ and $\sigma(t_n) > 0$ represents the known noise level of our training data. Following [15, 18], we refer to $\sigma(t_n)$ as the "nature noise level" – the inherent corruption level in the available dataset.

Similarly to (3.15), the objective is to learn the optimal denoiser $\mathbb{E}[X_0|X_t]$ for all noise levels, despite only having access to corrupted training data at level $\sigma(t_n)$. In other words, we only have access to training samples with noise levels $\sigma(t)$, such that $t \in [t_n, T]$.

3.5.3 Ambient Denoising Score Matching

The core contribution of ambient diffusion lies in the Ambient Denoising Score Matching (ADSM) loss, which enables learning optimal denoisers using only corrupted training data.

The challenge is that the clean conditional expectation $\mathbb{E}[X_0 \mid X_t]$ cannot be learned directly, since clean data is unavailable. Instead, we observe corrupted samples X_{t_n}, and the question is: *can we relate* $\mathbb{E}[X_0 \mid X_t]$ *to* $\mathbb{E}[X_{t_n} \mid X_t]$*, which can be learned from data?*

To see the idea, recall that we can always construct noisier samples from already corrupted ones:

$$X_t = X_{t_n} + \sqrt{\sigma(t)^2 - \sigma(t_n)^2}\, Z, \quad Z \sim \mathcal{N}(0, I), \quad t > t_n.$$

Thus, we can create a supervised learning setup where the input is the more noisy sample X_t and the target is the less noisy version X_{t_n}. Intuitively, the model learns to "remove the additional noise" that separates X_t from X_{t_n}.

The key theoretical result formalizing this idea is the following relationship between conditional expectations:

Lemma 3.5.1 (Connecting Conditional Expectations)**.** *Let $X_{t_n} = X_0 + \sigma(t_n)Z_1$ and $X_t = X_0 + \sigma(t)Z_2$, where $Z_1, Z_2 \sim \mathcal{N}(0, I)$ are independent. Then, for any $t > t_n$ [16]:*

$$\mathbb{E}[X_0 \mid X_t] = \frac{\sigma(t)^2}{\sigma(t)^2 - \sigma(t_n)^2}\, \mathbb{E}[X_{t_n} \mid X_t] - \frac{\sigma(t_n)^2}{\sigma(t)^2 - \sigma(t_n)^2}\, X_t. \tag{3.27}$$

This lemma shows that if we can estimate $\mathbb{E}[X_{t_n} \mid X_t]$ from corrupted data, then we can recover the clean conditional expectation $\mathbb{E}[X_0 \mid X_t]$ analytically. Based on Lemma 3.5.1, by rearranging the terms, the Ambient Denoising Score Matching loss is formulated as

$$J_{\text{ADSM}}(\theta) = \mathbb{E}_{X_{t_n}, X_t, t}\left[\left\| \frac{\sigma(t)^2 - \sigma(t_n)^2}{\sigma(t)^2}\, D_\theta(X_t, \sigma(t)) + \frac{\sigma(t_n)^2}{\sigma(t)^2} X_t - X_{t_n} \right\|^2\right], \tag{3.28}$$

where $D_\theta(X_t, \sigma(t))$ is the denoiser defined in Eq. (3.16).

This formulation transforms the problem of learning $\mathbb{E}[X_0 \mid X_t]$, which requires clean data, into the tractable problem of learning $\mathbb{E}[X_{t_n} \mid X_t]$, which can be learned directly from corrupted samples. Compared to alternative approaches that rely on consistency losses or sampling-based training procedures [16], the ADSM loss is computationally efficient and theoretically justified. In fact, it guarantees exact recovery of the clean conditional expectation in the limit of an infinite number of corrupted samples [15].

Mixed dataset training

Following prior work [15], the ADSM loss is integrated into the EDM framework by training on a mixed dataset composed of both clean and noisy samples, with a proportion p of the dataset consisting of corrupted observations. This approach assumes that we know which samples are corrupted and which are clean. In practice, for a given dataset $\mathcal{D}$, each clean sample X_0 is corrupted by adding Gaussian noise $\mathcal{N}(0, \sigma_{t_n}^2)$ with probability

Table 3.2: Training regimes and loss functions in the Ambient Diffusion framework.

Training Case	**Loss Used**	**Target**	**Data Required**
Clean only ($p = 0$)	EDM loss ($\mathcal{L}_{\text{EDM}}$)	X_0	Clean samples only
Noisy only ($p = 1$)	ADSM loss (J_{ADSM})	X_{t_n}	Corrupted samples only
Mixed ($0 < p < 1$)	EDM or ADSM	X_0 or X_{t_n}	Both clean and corrupted samples

p. This results in a split of the dataset into two disjoint subsets: the noisy subset $\mathcal{D}_{\text{noisy}}$, containing p percent of the samples, and the clean subset $\mathcal{D}_{\text{clean}}$, containing $1 - p$ percent of the samples.

The training procedure alternates dynamically between the two types of samples: when a clean sample is selected, the standard EDM loss (3.15) is applied; when a corrupted sample is selected, the ADSM loss (3.28) is used instead. The complete training procedure is outlined in Algorithm 7. In the special case where $p = 1$, the entire dataset is corrupted and training is performed exclusively with the ADSM loss. Conversely, when $p = 0$, all samples are clean and the training reduces to the standard EDM procedure described in Algorithm 5.

Table 3.2 summarizes the possible training regimes and corresponding loss functions in the ambient diffusion framework.

3.6 Ambient Diffusion for time series

3.6.1 Empirical Motivation

In this section, we introduce our proposed variance correction method based on the Ambient Diffusion framework and explain the motivation behind it. This method is a central contribution of the thesis and addresses key limitations of standard training procedures.

Preliminary experiments with the EDM model on Geometric Brownian Motion (GBM) yielded generally promising results. However, a systematic discrepancy was observed in a key characteristic of the generated samples – their volatility. As illustrated in Figure 3.8, the blue region shows the distribution of pathwise volatility estimates for the training data (as defined in 4.4.1), while the orange region shows the corresponding distribution for data

Algorithm 7 Mixed Ambient-EDM Training (Clean and Noisy Samples)

Require: Clean dataset $\mathcal{D}_{\text{clean}}$, noisy dataset $\mathcal{D}_{\text{noisy}}$, network D_θ, noise scale σ_{data}, log-noise mean P_{mean}, log-noise std P_{std}, ambient noise level $\sigma(t_n)$, batch size B

while not converged **do** ▷ Training loop
  Sample batch $\mathcal{B}$ of size B from $\mathcal{D}_{\text{clean}} \cup \mathcal{D}_{\text{noisy}}$
  $\mathcal{L}_{\text{batch}} \leftarrow 0$ ▷ Initialize batch loss
  for each sample X in $\mathcal{B}$ **do**
    Sample $\ln \sigma \sim \mathcal{N}(P_{\text{mean}}, P_{\text{std}}^2)$ ▷ Sample noise level
    $\sigma \leftarrow \exp(\ln \sigma)$
    if $X \in \mathcal{D}_{\text{clean}}$ **then** ▷ Clean sample branch
      $X_0 \leftarrow X$
      Sample $Z \sim \mathcal{N}(0, I)$ ▷ Sample standard noise
      $X_t \leftarrow X_0 + \sigma Z$ ▷ Corrupt input
      $\hat{X}_0 \leftarrow D_\theta(X_t, \sigma)$ ▷ Denoised prediction
      $\mathcal{L} \leftarrow \lambda(\sigma) \cdot \|\hat{X}_0 - X_0\|_2^2$ ▷ Compute loss
    else ▷ Noisy sample branch (Ambient training)
      $X_{t_n} \leftarrow X$ ▷ Observed noisy sample
      Sample $Z_{\text{res}} \sim \mathcal{N}(0, I)$ ▷ Residual noise
      $X_t \leftarrow X_{t_n} + \sqrt{\sigma^2 - \sigma(t_n)^2}\, Z_{\text{res}}$ ▷ Add residual noise
      $\hat{X}_0 \leftarrow D_\theta(X_t, \sigma)$ ▷ Denoised prediction
      $\hat{X}_{t_n} \leftarrow \frac{\sigma^2 - \sigma(t_n)^2}{\sigma^2}\, \hat{X}_0 + \frac{\sigma(t_n)^2}{\sigma^2}\, X_t$ ▷ Project to noisy target (ADSM loss)
      $\mathcal{L} \leftarrow \lambda(\sigma) \cdot \|\hat{X}_{t_n} - X_{t_n}\|_2^2$ ▷ Compute loss
    end if
    $\mathcal{L}_{\text{batch}} \leftarrow \mathcal{L}_{\text{batch}} + \mathcal{L}$
  end for
  Update θ using $\nabla_\theta \left(\mathcal{L}_{\text{batch}}/B\right)$ ▷ Gradient descent step
end while
return θ

generated by the EDM model. The generated series consistently exhibit higher estimated volatility than the ground truth.

In contrast, when training with Ambient Diffusion, we observed a noticeable improvement. Specifically, we trained three Ambient Diffusion models with increasing levels of injected dataset noise, denoted by $\sigma(t_n)$. Figure 3.9 displays the resulting volatility distributions. As the ambient noise level increases, the gap between the real and generated distributions narrows.

In other words, introducing a controlled amount of corruption during training and applying the Ambient Diffusion procedure (Algorithm 7) improves the alignment of volatility distributions between training and generated data.

In the following sections, we investigate this effect and propose our solution to find the optimal ambient noise parameter $\sigma(t_n)$ values.

3.6.2 Connecting GBM volatility to variance

We first want to draw a connection between the volatility of GBM and the raw input and output of the diffusion model – standardized log returns.

Lemma 3.6.1 (Distribution of GBM log returns)**.** *Let the asset price follow a geometric Brownian motion*

$$dS_t = \mu S_t\, dt + \nu S_t\, dW_t, \tag{3.29}$$

with constants $\mu \in \mathbb{R}$ and $\nu > 0$. For a fixed interval $\Delta t > 0$, the log return

$$r_t = \log\left(\frac{S_t}{S_{t-\Delta t}}\right) \tag{3.30}$$

is Gaussian with

$$r_t \sim \mathcal{N}\left(\left(\mu - \tfrac{1}{2}\nu^2\right)\Delta t,\ \nu^2 \Delta t\right). \tag{3.31}$$

Proof. Define $Y_t := \log S_t$. By Itô's lemma (Thm. 2.2.3), we have

$$dY_t = \frac{1}{S_t}\left(\mu S_t\, dt + \nu S_t\, dW_t\right) - \frac{1}{2}\frac{1}{S_t^2}\left(\nu^2 S_t^2\right) dt \tag{3.32}$$

$$= \left(\mu - \tfrac{1}{2}\nu^2\right) dt + \nu\, dW_t. \tag{3.33}$$

The log return over $[t - \Delta t,\, t]$ satisfies

$$r_t = Y_t - Y_{t-\Delta t}. \tag{3.34}$$

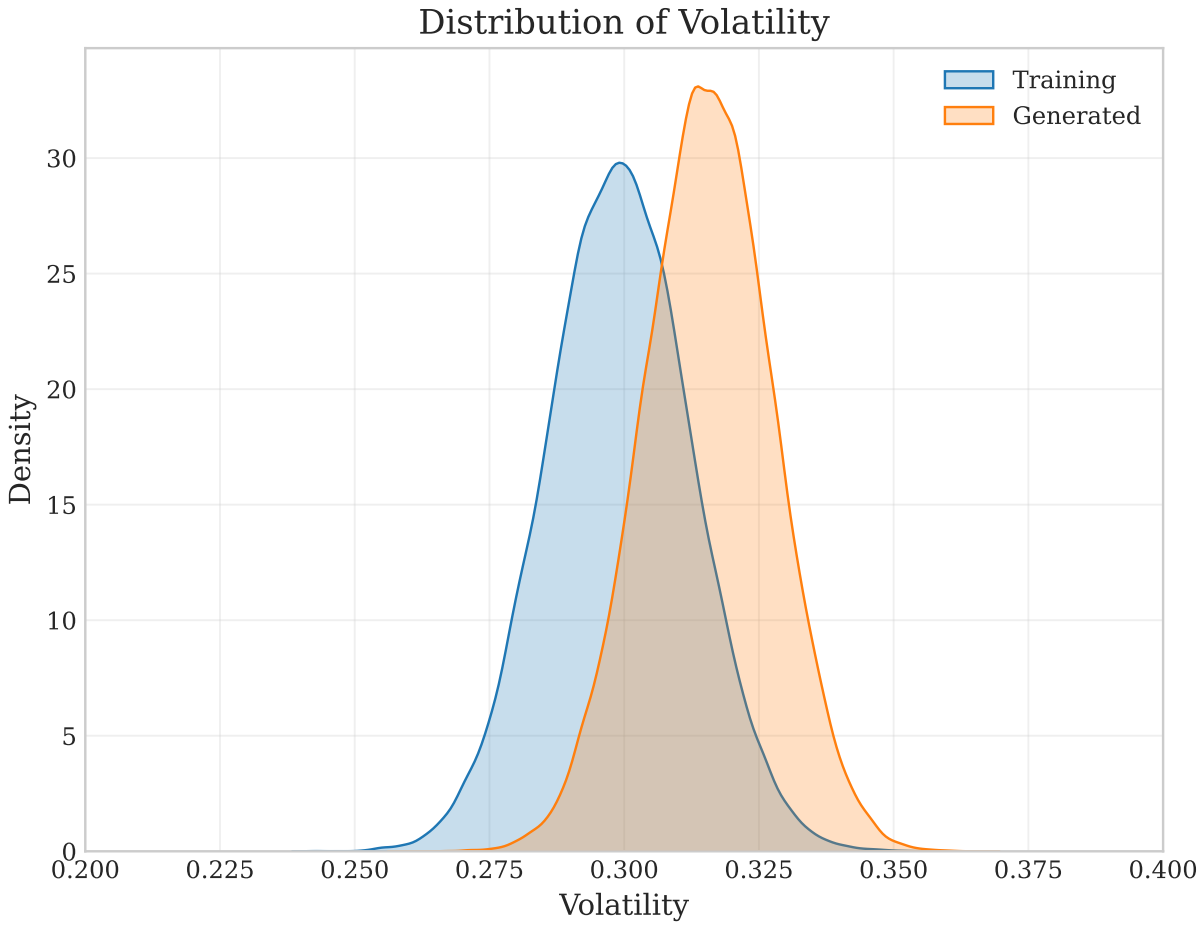
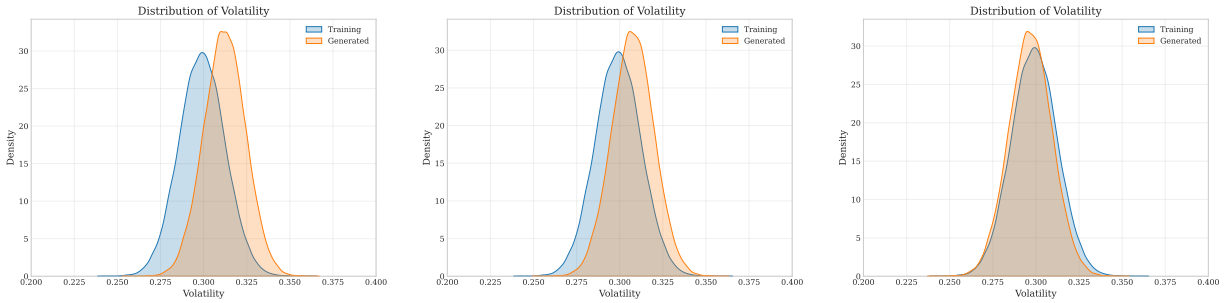


Figure 3.8: Distribution of pathwise volatility estimates for GBM: training data (blue) vs. EDM-generated data (orange). The EDM model overestimates volatility.



(a) $\sigma(t_n) = 0.005$ (b) $\sigma(t_n) = 0.010$ (c) $\sigma(t_n) = 0.016$

Figure 3.9: Distribution of pathwise volatility estimates under Ambient Diffusion with increasing ambient noise levels. Larger $\sigma(t_n)$ values reduce the gap between training and generated data.

By integrating dY_t over $[t - \Delta t,\, t]$, we obtain

$$r_t = \int_{t-\Delta t}^{t} \left(\mu - \tfrac{1}{2}\nu^2\right) ds \; + \; \int_{t-\Delta t}^{t} \nu \, dW_s \tag{3.35}$$

$$= \left(\mu - \tfrac{1}{2}\nu^2\right) \Delta t + \nu \int_{t-\Delta t}^{t} dW_s \tag{3.36}$$

By definition 2.2.1, the Brownian increment satisfies

$$\int_{t-\Delta t}^{t} dW_s \sim \mathcal{N}(0, \Delta t). \tag{3.37}$$

Hence

$$r_t \sim \mathcal{N}\left(\left(\mu - \tfrac{1}{2}\nu^2\right) \Delta t,\; \nu^2 \Delta t\right). \tag{3.38}$$

□

By Lemma 3.6.1, the variance of log returns satisfies $\mathrm{Var}(r_t) = \nu^2 \Delta t$. Hence, after z-score standardization as in Eq. (3.13),

$$z_t \;=\; \frac{r_t - \mathbb{E}[r_t]}{\sqrt{\mathrm{Var}(r_t)}} \;\sim\; \mathcal{N}(0, 1),$$

which links the diffusion model's standardized outputs directly to the GBM volatility through $\mathrm{Var}(r_t) = \nu^2 \Delta t$.

Next, we show that any increase in variance of standardized log returns directly increases the volatility of GBM data.

Proposition 3.6.2 (Variance inflation scales GBM volatility)**.** *Suppose standardized log returns z_t have variance* $\mathrm{Var}(z_t) = 1$*. If instead a model produces standardized returns with inflated variance* $\mathrm{Var}(z'_t) = 1 + \epsilon^2$*, then the corresponding GBM volatility parameter ν is biased upward according to*

$$\nu'^2 = \nu^2(1 + \epsilon^2).$$

Proof. By Lemma 3.6.1, the variance of log returns is $\mathrm{Var}(r_t) = \nu^2 \Delta t$. Reversing the z-score normalization (Eq. 3.14) for z'_t yields

$$r'_t = \nu\sqrt{\Delta t}\, z'_t + \mathbb{E}[r_t],$$

so

$$\mathrm{Var}(r'_t) = \nu^2 \Delta t \cdot \mathrm{Var}(z'_t) \tag{3.39}$$
$$= \nu^2 \Delta t(1 + \epsilon^2). \tag{3.40}$$

Dividing both sides by Δt gives the result. □

This shows that any excess variance in standardized outputs translates directly into an overestimation of GBM volatility. Hence, correcting variance inflation at the standardized level is equivalent to correcting volatility bias in the generated time series.

3.6.3 Ambient Diffusion as a Variance Correction Mechanism

As discussed in the previous section, when the EDM model is trained on GBM data, the generated samples often exhibit excess volatility, which is caused by the generated data variance overshoot, i.e.,

$$\omega = \mathrm{Var}_{\mathrm{EDM}} > 1, \tag{3.41}$$

where ω denotes the observed variance of the generated samples.

We found that Ambient Diffusion introduces a variance shrinkage effect through the ADSM loss (3.28). In the ambient setting, the output of the denoiser D_θ is multiplied by the following scaling factor:

$$g(\sigma(t)) = \frac{\sigma(t)^2 - \sigma(t_n)^2}{\sigma(t)^2} \tag{3.42}$$
$$= 1 - \frac{\sigma(t_n)^2}{\sigma(t)^2}. \tag{3.43}$$

We propose to approximate this factor by a constant by taking its expectation $\mathbb{E}[g(\sigma)]$. Since $\sigma(t_n)$ is fixed during training, we simplify the notation by writing it as σ_{t_n}. As discussed in Section 3.3.3, the timesteps t are sampled such that $\sigma(t)$ follows a log-normal distribution with parameters $P_{\text{mean}} = -1.2$ and $P_{\text{std}} = 1.2$. From the probability density function shown in Figure 3.7, we observe that most values of $\sigma(t)$ are centered around 0.1. In fact, the mode of the log-normal distribution, which corresponds to the peak of the PDF, is given by:

$$\sigma_{\text{mode}} = \exp\left(P_{\text{mean}} - P_{\text{std}}^2\right). \tag{3.44}$$

Moreover, since $\mathbb{E}[\sigma^{-2}] = e^{-2P_{\text{mean}}+2P_{\text{std}}^2} = \sigma_{\text{mode}}^{-2}$ for a log-normal schedule, the average shrinkage equals

$$\bar{g} = \mathbb{E}[g(\sigma)] = 1 - \frac{\sigma_{t_n}^2}{\sigma_{\text{mode}}^2},$$

which justifies the constant approximation used below.

This means that for most training steps, the training happens in σ_{mode} noise regime and the scaling factor $g(\sigma(t))$ is approximately constant. Therefore, we propose to treat it as such and denote:

$$\gamma := \bar{g} = 1 - \frac{\sigma_{t_n}^2}{\sigma_{\text{mode}}^2}. \tag{3.45}$$

This interpretation suggests that, under the constant approximation with respect to diffusion time t, the ADSM loss introduces a consistent shrinkage factor $\gamma < 1$ that cancels the variance overshoot of standard EDM training. As a result, the target total variance of the generated samples becomes:

$$\gamma^2\omega = 1. \tag{3.46}$$

Solving for γ yields the optimal shrinkage factor:

$$\gamma = \frac{1}{\sqrt{\omega}}. \tag{3.47}$$

Equating the empirical shrinkage factor derived from the ADSM loss,

$$\gamma = 1 - \frac{\sigma_{t_n}^2}{\sigma_{\text{mode}}^2}, \tag{3.48}$$

with the optimal correction in (3.46), we obtain:

$$1 - \frac{\sigma_{t_n}^2}{\sigma_{\text{mode}}^2} = \frac{1}{\sqrt{\omega}}. \tag{3.49}$$

Solving for σ_{t_n} gives the optimal ambient parameter:

$$\sigma_{t_n}^2 = \left(1 - \frac{1}{\sqrt{\omega}}\right) \sigma_{\text{mode}}^2, \tag{3.50}$$

where ω is the empirically measured variance of the data generated by the EDM model, and σ_{mode} is the mode of the log-normal noise schedule.

As discussed in Section 3.5.3, it is often useful to train on a mixed dataset where a fraction p of samples use the ambient loss and a fraction $1 - p$ use the original EDM loss. In this setting, only a proportion p of training steps apply shrinkage. To achieve the same overall variance correction, the per-sample shrinkage must be amplified by a factor of $1/p$. This modifies the ambient noise level as follows:

$$\sigma_{t_n} = \frac{\sigma_{\text{mode}}}{p} \cdot \sqrt{1 - \frac{1}{\sqrt{\omega}}}. \tag{3.51}$$

To address the issue of variance amplification, based on Eq. (3.51), we propose a two-stage training procedure that incorporates an optimally tuned ambient diffusion component.

Two-Stage Training Procedure

Our training approach consists of:

1. **Pilot Run**: Train a standard EDM model using Algorithm 5 (or equivalently Algorithm 7 with $p = 0$) with reduced computational resources (e.g., fewer epochs) to estimate the variance of generated samples $\omega = \text{Var}(\mathcal{D}_{\text{generated_pilot}})$.
2. **Full Training**: Using the measured ω, compute the optimal ambient noise level with Eq. (3.51).Train the final model using Algorithm 7 with this computed σ_{t_n}.

This two-stage approach ensures that the generated samples maintain the same statistical properties as the training data, leveraging the benefits of ambient diffusion training.

3.7 Summary

In summary, this chapter presented a methodology for generative modeling of financial time series using diffusion models. We outlined the rationale for dataset selection and described the collection and preprocessing of both synthetic and real-world market data. Based on the EDM framework, we introduced models adaptations for time series input data and proposed a variance correction mechanism based on the Ambient Diffusion framework. These developments lay the groundwork for the experimental validation and empirical analysis in the subsequent chapters.

Chapter 4

Evaluation Methodology

4.1 Introduction

Assessing the performance of a generative model is a non-trivial task. Unlike traditional supervised machine learning, where models can be evaluated by comparing predictions to known ground truth on an unseen test set, generative modeling lacks such explicit ground truth references. Since the true underlying data distribution is unknown, it is difficult to determine whether the model has successfully learned it and captured its characteristics. As a result, evaluation in generative modeling typically focuses on comparing the distributions of the generated and real data, rather than evaluating the accuracy of individual samples. This comparison relies on statistical metrics and tests designed to measure distributional similarity between two datasets.

Moreover, the choice of evaluation strategy is not universal. It is dependent on both the data modality and the specific task. For example, generative models for natural images are commonly evaluated using perceptual quality metrics such as Fréchet Inception Distance (FID) [31], Inception Score (IS) [65], or structural similarity indices (SSIM) [77]. In audio generation, evaluation focuses on signal reconstruction fidelity or perceptual sound quality [40, 10, 4]. Similarly, the nature of the task, such as conditional or unconditional generation, or solving inverse problems (e.g., reconstruction or denoising), also shapes the appropriate evaluation methodology [67, 39].

In the context of this work, where the goal is the unconditional generation of financial time series, perceptual metrics are not meaningful. Instead, our evaluation framework focuses on assessing how closely the distribution of the generated data matches the distribution of the training data. To this end, we use a variety of quantitative metrics, statistical

tests, and visualization techniques that evaluate both general statistical properties of time series (e.g., moments, correlation) and financial-specific features (e.g., volatility, Value at Risk).

We follow the notation established in Section 3.1, where $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{gen}}$ denote the training and generated datasets, respectively, consisting of log return sequences. The corresponding reconstructed price series are denoted by $\mathcal{P}_{\text{train}}$ and $\mathcal{P}_{\text{gen}}$.

4.2 Distributional Similarity

To assess the similarity between the real and generated data distributions, we report two complementary metrics: the Wasserstein distance [3], which captures global distributional shifts and the Kolmogorov–Smirnov test [49], which provides a statistical hypothesis test to assess the significance of distributional differences. Distributional similarity metrics are computed on the log return datasets $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{gen}}$.

Kolmogorov–Smirnov (KS) Test

The Kolmogorov–Smirnov test is a non-parametric statistical test that quantifies the maximum distance between the empirical cumulative distribution functions of two samples. Given two cumulative distributions $F(x)$ and $G(x)$, the KS statistic is defined as:

$$D_{KS} = \sup_x |F(x) - G(x)| \tag{4.1}$$

where $\sup_x$ denotes the supremum over all values of x.

In this work, we compute the KS statistic between the real and generated log return distributions, along with the associated p-value. The KS test is sensitive to both location and shape differences and provides a hypothesis test for distributional similarity without assuming any specific distributional form.

Wasserstein Distance

The Wasserstein distance measures the minimal "effort" required to transform one distribution into another. For distributions with cumulative distribution functions $F(x)$ and $G(x)$, the Wasserstein distance is given by:

$$W_1(F, G) = \int_{-\infty}^{\infty} |F(x) - G(x)| \, dx \tag{4.2}$$

In the context of generative time series modeling, the Wasserstein distance captures global shifts between real and generated distributions and reflects the alignment of the distributions. Unlike the KS test, which measures only the maximum deviation, the Wasserstein distance reflects the cumulative differences across the entire distribution.

4.3 Pathwise Time Series Statistics

In addition to comparing marginal distributions, we evaluate the distributions of key pathwise time series statistics, computed for each individual sample in $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{gen}}$. Specifically, we compute the mean, variance, skewness, and kurtosis and analyze the resulting distributions of these statistics for real and generated data.

The mean and variance capture the average return and volatility of an asset, while skewness and kurtosis describe asymmetry and tail risk of returns. Higher-order moments are important for a better understanding of extreme events and the non-Gaussian nature of financial returns. Negative skewness in asset returns is associated with large negative shocks (e.g., market crashes), while high kurtosis corresponds to fat-tail behavior.

4.4 Time Series Metrics

In addition to the general statistical metrics described in the previous section, we introduce a set of evaluation metrics that specifically target properties of financial time series described in Section 3.2. These metrics assess whether the generative models can reproduce temporal and structural characteristics such as drift, constant and time-varying volatility, jump behavior, and cross-asset correlations.

Drift & Volatility Estimation

We compute pathwise estimations of the annualized drift and volatility for each generated and real sample trajectory. The distribution of estimates is then compared across real and generated datasets to assess whether the generative model captures the typical average return and realized volatility observed in the training data.

For each time series sample, we estimate the annualized drift and volatility under the assumption of a continuous-time GBM process. The volatility is estimated as the square root of the average squared log returns, scaled by the time horizon. The drift is estimated

from the total log return over the trajectory, adjusted by the variance correction term associated with the GBM solution. These estimators are used for datasets where the GBM assumption is reasonable, in particular GBM, CGBM, and real-world datasets.

Definition 4.4.1 (Pathwise Estimators of Drift and Volatility)**.** *Let $\{r_t\}_{t=1}^N$ denote the sequence of log returns over a time horizon T. The annualized volatility $\hat{\nu}$ is estimated as:*

$$\hat{\nu} = \sqrt{\frac{1}{T}\sum_{t=1}^{N} r_t^2} \tag{4.3}$$

The annualized drift $\hat{\mu}$ is estimated as:

$$\hat{\mu} = \frac{1}{T}\sum_{t=1}^{N} r_t + \frac{1}{2}\hat{\nu}^2 \tag{4.4}$$

Realized Volatility Estimation for Heston Model

To evaluate whether the models accurately capture the temporal evolution of volatility, we compute the instantaneous realized volatility over time for both real and generated datasets. Realized volatility at each time step is estimated similarly to the annualized volatility and scaled by the number of time steps per year to express the result in annualized terms. This metric is used to assess whether the model is able to capture non-constant volatility patterns, such as in the Heston stochastic volatility model.

Definition 4.4.2 (Realized Volatility)**.** *Let $\{r_t^{(i)}\}_{i=1}^M$ denote the log returns across M sample paths at time step t, and let N denote the number of time steps. The realized volatility at time step t is estimated as:*

$$\widehat{\nu}_t = \sqrt{N \times \frac{1}{M}\sum_{i=1}^{M}\left(r_t^{(i)}\right)^2} \tag{4.5}$$

Maximum Likelihood Estimation for Merton Jump Diffusion

For the Merton Jump Diffusion model, we are interested in assessing the behavior of jumps, which is defined by three parameters: the jump intensity λ, the average jump size μ_J, and the standard deviation of jump sizes σ_J. Combined with the drift μ and diffusion volatility

ν, this yields a five-parameter model. Simple estimators, such as those presented in previous sections, are not sufficient to accurately estimate all five parameters simultaneously. To address this, we resort to Maximum Likelihood Estimation (MLE) to jointly estimate the full set of MJD model parameters. The MLE procedure allows us to infer the parameters that maximize the likelihood of observing the given data under the MJD model.

Definition 4.4.3 (Maximum Likelihood Estimation)**.** *Let $\mathcal{D} = \{x_1, x_2, \ldots, x_n\}$ be a dataset consisting of n i.i.d. samples, and let $p(x \mid \psi)$ be the probability density function of the model parameterized by ψ.*

The likelihood function is defined as:

$$L(\psi) = p(x_1, x_2, \ldots, x_n \mid \psi) = \prod_{i=1}^{n} p(x_i \mid \psi)$$

The maximum likelihood estimate of ψ, denoted $\hat{\psi}_{\text{MLE}}$, is the value that maximizes the log-likelihood:

$$\hat{\psi}_{\text{MLE}} = \arg\max_{\psi} \log L(\psi) = \arg\max_{\psi} \sum_{i=1}^{n} \log p(x_i \mid \psi)$$

Definition 4.4.4 (Merton Jump Diffusion Probability Density Function)**.** *The Merton Jump Diffusion model assumes that log returns at time T follow the density [12]:*

$$p_T(x) = e^{-\lambda T} \sum_{k=0}^{\infty} \frac{(\lambda T)^k}{k!} \frac{1}{\sqrt{2\pi\,(\nu^2 T + k\sigma_J^2)}} \exp\left(-\frac{(x - \mu T - k\mu_J)^2}{2\,(\nu^2 T + k\sigma_J^2)} \right) \tag{4.6}$$

where x denotes the observed log return over horizon T.

In practice, we follow the MLE definition 4.4.3 and numerically minimize the negative log-likelihood associated with Eq. 4.6 to obtain the optimal parameter estimates for both the real and generated datasets. The estimated parameters are then compared to the known ground truth parameters of the MJD process.

Correlation Estimation for Correlated Geometric Brownian Motion (CGBM)

For multivariate time series, we assess whether the models can accurately reproduce the dependency structure between different time series components. Specifically, we estimate the empirical correlation matrices of asset returns for both real and generated datasets.

Definition 4.4.5 (Pathwise Correlation Estimation)**.** *Given a dataset $\mathcal{D} \in \mathbb{R}^{M\times d\times N}$ containing log returns $\boldsymbol{r}^{(i)} \in \mathbb{R}^{d\times N}$, $i = 1, \ldots, M$, we compute the empirical correlation matrix $\widehat{\Sigma}^{(i)} \in \mathbb{R}^{d\times d}$ for each path:*

$$\widehat{\Sigma}^{(i)} = \mathrm{Corr}(\boldsymbol{r}^{(i)}) \tag{4.7}$$

where $\mathrm{Corr}(\cdot)$ *denotes the correlation matrix computed across the d assets.*

The dataset-wide average correlation matrix $\widehat{\Sigma} \in \mathbb{R}^{d\times d}$ is then computed as the elementwise mean:

$$\widehat{\Sigma} = \frac{1}{M}\sum_{i=1}^{M} \widehat{\Sigma}^{(i)} \tag{4.8}$$

We compare the resulting average correlation matrices between the real and generated datasets to assess the model's ability to capture cross-asset dependency structure.

4.5 Financial Risk Metrics

We compute Value at Risk (VaR) and Conditional Value at Risk (CVaR) [61] for each sample path in the training and generated datasets and compare the resulting distributions of these risk measures.

Definition 4.5.1 (VaR and CVaR for Returns)**.** *Let R be the random variable representing returns, and let $\alpha \in (0, 1)$ denote the confidence level (e.g., $\alpha = 0.95$).*

- ***VaR:***

  $$\mathrm{VaR}_\alpha(R) = \inf\{r \in \mathbb{R} : P(R \le r) \ge 1 - \alpha\}$$

  VaR is the return threshold such that the probability of returns falling below this value is at least $1 - \alpha$.

- ***CVaR:***

  $$\mathrm{CVaR}_\alpha(R) = \mathbb{E}\left[R \,|\, R \le \mathrm{VaR}_\alpha(R)\right]$$

  CVaR is the expected return conditional on returns being below the corresponding VaR threshold.

These risk measures are widely used in finance to quantify potential losses and assess the tail behavior of asset returns. We apply VaR and CVaR primarily to the real-world datasets to evaluate the models' ability to capture the risks associated with significant asset losses.

4.6 Option Pricing Benchmark

Finally, we use an option pricing task as a practical benchmark for assessing model performance through a quantitative, application-driven evaluation.

An option is a financial derivative that gives the holder the right, but not the obligation, to buy (call) or sell (put) an underlying asset at a pre-agreed strike price on a predefined expiration date. In this work, we focus on European style options, which can only be exercised at maturity. At the expiry time T, the payoff of a European call option is defined as:

$$V_T = \max(S_T - K, 0),$$

where S_T is the underlying asset price at time T, and K is the strike price. Similarly, for a put option, the payoff is:

$$V_T = \max(K - S_T, 0)$$

For multivariate time series ($d > 1$), we evaluate the generative model using basket-style derivatives that depend jointly on multiple asset prices. In particular, we consider three option types: Max, Basket, and Spread [84].

Max Options

In the multivariate setting, a max option is written on the maximum value among several assets. The payoff for a European max call option is:

$$V_T = \max\left(\max_j S_T^{(j)} - K,\, 0\right)$$

and for a max put:

$$V_T = \max\left(K - \max_j S_T^{(j)},\, 0\right)$$

where $S_T^{(j)}$ is the terminal value of the j-th asset.

Basket Options

A basket option is written on the average price of a portfolio of assets. Assuming an equally weighted basket, the average price at maturity is:

$$\bar{S}_T = \frac{1}{d}\sum_{j=1}^{d} S_T^{(j)}$$

The corresponding call and put payoffs are:

$$V_T^{\text{call}} = \max(\bar{S}_T - K,\, 0), \qquad V_T^{\text{put}} = \max(K - \bar{S}_T,\, 0)$$

Spread Options

Spread options are written on the difference between two asset prices. For a call option based on the spread between asset i and asset j, the payoff is:

$$V_T = \max(S_T^{(i)} - S_T^{(j)} - K,\, 0)$$

and the corresponding put option has the payoff:

$$V_T = \max(K - (S_T^{(i)} - S_T^{(j)}),\, 0)$$

Option pricing problem

The option pricing problem consists of determining the fair value of an option today, given the dynamics of the underlying asset, the option contract parameters, and market conditions. The fair value is defined as the discounted expected payoff at maturity, expressed in present value terms under the risk-neutral probability measure $\mathbb{Q}$ [33]:

$$V_0 = e^{-rT}\, \mathbb{E}^{\mathbb{Q}}[V_T] \tag{4.9}$$

The price of a European option depends on several factors, including the current price of the underlying asset S_0, the strike price K, the time to maturity T, the volatility of the underlying asset ν, and the risk-free interest rate r. Option prices are also sensitive to the statistical properties and dynamics of the underlying time series, such as the presence of jumps or stochastic volatility.

This sensitivity makes option pricing a useful benchmark for assessing the quality of generated time series, as option values aggregate information about both the distributional features and temporal dynamics of the underlying process. To assess model performance, we compare option prices computed using generated data to a closed-form solution (such as Black-Scholes), when available. Otherwise, we use Monte Carlo estimates on the training data as the ground truth.

4.7 Summary

In this chapter, we have presented a comprehensive evaluation framework for generative models of financial time series, which constitutes an additional contribution of our work. By combining distributional similarity metrics, pathwise statistics, time series–specific properties, risk-based measures, and option pricing results, we assess both the statistical fidelity and the practical utility of generated data. In the next chapter, we apply this evaluation methodology to assess and compare the generative models developed in this work.

Chapter 5

Experiments & Results

5.1 Introduction

In this chapter, we describe the experimental setup and present the computational results of our work. We begin by outlining the training procedures and hyperparameters used across all models, followed by a detailed analysis of the results for each dataset using the evaluation methodology introduced in the previous chapter. Results are grouped by dataset type: synthetic and real-world.

5.2 Experimental Setup

For each dataset, we present two sets of training results corresponding to the algorithms described in Alg. 5 and Alg. 7. We refer to the models yielded by these algorithms as **EDM** and **Ambient**, respectively. The Ambient variant incorporates our proposed variance correction method described in Section 3.6. After training, we generate samples using the sampling procedure described in Algorithm 6. The resulting generated dataset $\mathcal{D}_{\text{gen}}$ is evaluated according to the framework introduced in Chapter 4.

Throughout this work, we use a fixed random number generator seed to ensure reproducibility. Training and generation were performed on NVIDIA RTX 6000 Ada and NVIDIA L40S GPUs.

Table 5.1: Training hyperparameters and grid search ranges. Bold indicates the selected configuration for both models.

Hyperparameter	Values
Batch size	32, **64**, 128
Learning rate	**0.001**, 0.0001
Epochs	10, **25**, 50, 100

Training Hyperparameters

All models were trained and evaluated using a shared set of hyperparameters, selected via grid search. The grid search results are shown in Table 5.1, with bold values indicating the chosen configuration. All other settings were kept unchanged from the original implementations [37, 15].

Ambient Diffusion Parameters

For Ambient training, the proportion of corrupted data p was selected via grid search over the range [0.1, 1.0] in increments of 0.1. The optimal value was typically at 0.4 or 0.5, depending on the dataset. This balance ensures there are sufficient corrupted samples for the variance correction mechanism to be effective, while retaining enough clean samples for the model to learn the uncorrupted distribution. The ambient corruption level σ_{t_n} was chosen using Eq. 3.51, based on variance estimates obtained from a pilot run as described in Section 3.6.

5.3 Experiments on Synthetic Datasets

In this section, we present training and evaluation results on four synthetic datasets: Geometric Brownian Motion (GBM), Correlated GBM (CGBM), the Heston Stochastic Volatility Model, and the Merton Jump-Diffusion Model. These datasets were introduced in Section 3.2 and serve as controlled benchmarks for assessing the model's ability to capture key properties of financial time series.

Table 5.2: Simulation Setup for Geometric Brownian Motion

Parameter	**Value**
Drift (μ)	0.05
Volatility (ν)	0.3
Initial Price (S_0)	100
Number of Time Steps (N)	256
Number of Paths (M)	100,000
Time Horizon (T)	1.0

5.3.1 Geometric Brownian Motion

Setup

We begin with the GBM dataset, introduced in Section 3.2.1, which serves as a baseline for evaluating the ability of diffusion models to replicate basic stochastic dynamics with constant drift and volatility.

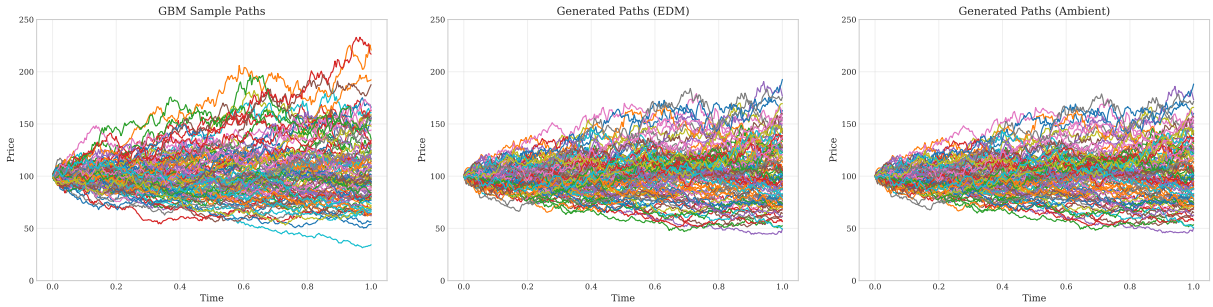
The dataset was generated via a Monte Carlo simulation using the closed-form discretization of the GBM SDE. The simulation parameters are summarized in Table 5.2.

Generated Samples

After training, we generate 100,000 samples using both EDM and Ambient models for direct comparison with the training dataset. Both approaches produce visually similar samples to the reference GBM process. Figure 5.1 shows 100 sample paths from the training dataset and generated by each model.

Distributional Similarity

To quantify model performance beyond visual results, we analyze distributional similarity between the generated dataset $\mathcal{D}_{\text{gen}}$ and the training dataset $\mathcal{D}_{\text{train}}$. Specifically, we compute two metrics introduced in Chapter 4: the Kolmogorov–Smirnov (K–S) statistic and the Wasserstein distance. These metrics quantify the divergence between the log return distributions of real and generated data, with lower values indicating closer alignment. These results are reported in Table 5.3. While both models achieve low values, the



(a) GBM (b) EDM (c) Ambient

Figure 5.1: Comparison of sample paths from the reference GBM process (left), and those generated by EDM (middle) and Ambient (right) models.

Table 5.3: Distributional similarity metrics between generated and training log return distributions (lower is better).

Metric	**EDM**	**Ambient**
K-S Statistic	0.01287	**0.00084**
Wasserstein Distance	0.00081	**0.00003**

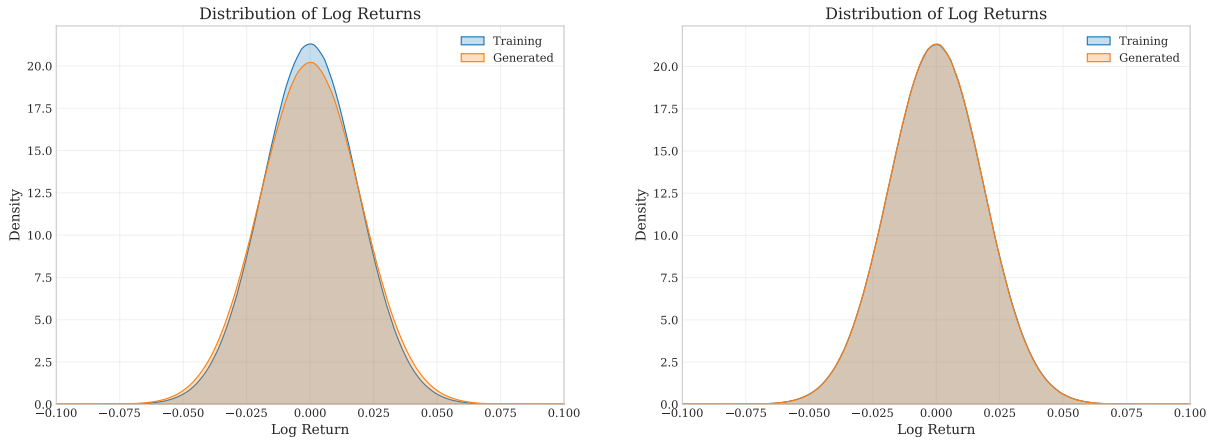
Ambient model significantly outperforms EDM across both metrics, indicating a better distributional match.

To support these findings visually, Figure 5.2 shows the estimated log return densities for both models and the training data. The Ambient model achieves a near-perfect overlap, while EDM slightly overestimates the variance, resulting in a wider distribution. This confirms the numerical observation that Ambient provides a closer approximation to the training data distribution.

Drift and Volatility Estimation

We evaluate the models' ability to reproduce the underlying GBM parameters by analyzing pathwise drift and volatility estimates. Using the estimators from Definition 4.4.1, we compute annualized drift and volatility for each sample path in both the training and generated datasets.

Figure 5.3 shows the distributions of estimated parameters. Both models successfully center their drift estimates around the true value of $\mu = 0.05$, with the Ambient



(a) EDM (b) Ambient

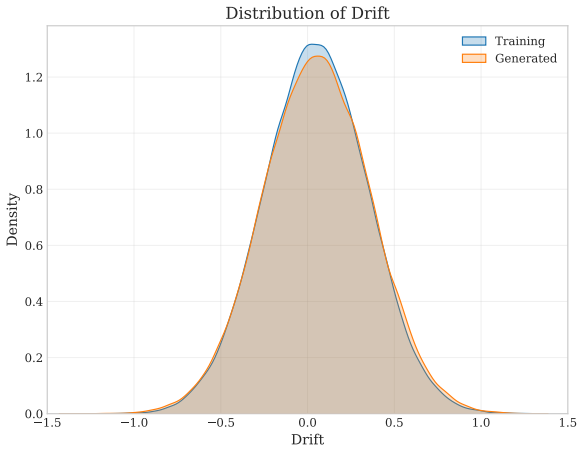
Figure 5.2: Log return distributions of training and generated data for the GBM dataset. The blue region represents the training distribution, while the orange region shows the generated distribution for the EDM model in (a) and the Ambient model in (b).

model achieving slightly better overlap. For volatility estimation, the difference is more pronounced: EDM exhibits significant overestimation with a distribution shifted toward higher values, while the Ambient model closely matches the training data distribution centered around the true volatility of $\nu = 0.3$. This confirms that the Ambient approach effectively addresses the variance amplification issue observed in standard EDM training, resulting in more accurate parameter recovery.

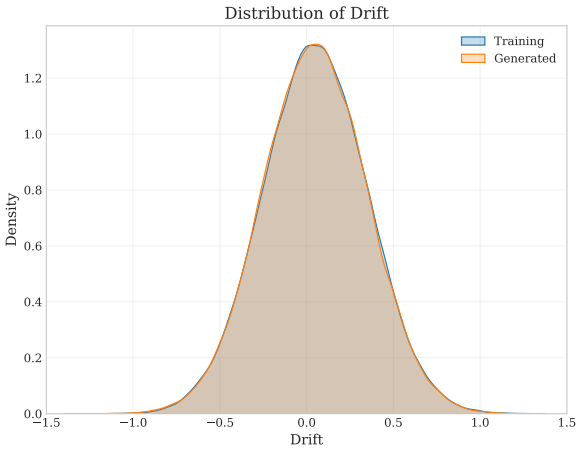
Option Pricing Results

Next, we present option pricing results for European call and put options in Table 5.4. To evaluate model performance across a range of market scenarios, we consider multiple strike prices K, covering out-of-the-money (OTM), at-the-money (ATM), and in-the-money (ITM) settings for both option types. The table reports theoretical prices computed using the Black–Scholes formula, alongside prices derived from data generated by the EDM and Ambient models, with relative errors shown for each.

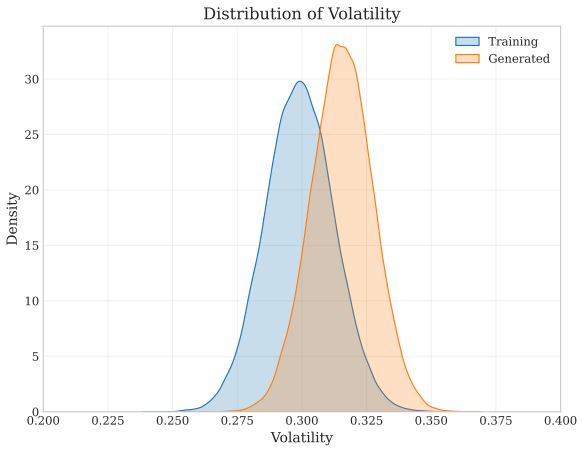
Across nearly all strike prices and option types, the Ambient model consistently achieves lower pricing errors than EDM. This is particularly noticeable in the OTM region, where the relative errors for EDM increase significantly, exceeding 11% for deep OTM puts. In contrast, Ambient keeps its errors under 3% consistently. For ATM and ITM scenarios,



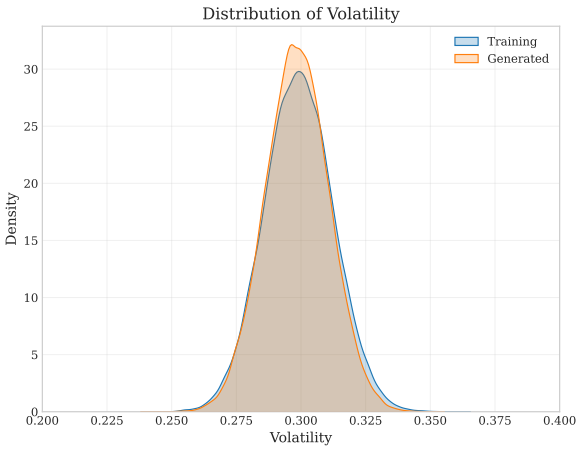
(a) EDM – Drift



(b) Ambient – Drift



(c) EDM – Volatility



(d) Ambient – Volatility

Figure 5.3: Distributions of estimated drift (top) and volatility (bottom) for the GBM dataset. Blue regions represent training data estimates, while orange regions correspond to generated samples from the EDM (left) and Ambient (right) models.

Table 5.4: European option prices and relative error compared to theoretical prices. Moneyness is defined relative to $S_0 = 100$.

Strike Price K	**Theoretical Price**	**EDM Price**	**Ambient Price**	**Error (EDM)**	**Error (Ambient)**	**Moneyness**
Call Options						
70	34.395	34.740	34.076	-1.00%	**0.93%**	
80	26.462	26.889	26.162	-1.61%	**1.14%**	ITM
90	19.697	20.200	19.435	-2.55%	**1.33%**	
100	14.231	14.767	14.006	-3.76%	**1.58%**	ATM
110	10.020	10.545	9.831	-5.24%	**1.89%**	
120	6.904	7.379	6.743	-6.88%	**2.34%**	OTM
130	4.673	5.073	4.535	-8.55%	**2.96%**	
Put Options						
70	0.981	1.091	0.989	-11.12%	**-0.79%**	
80	2.560	2.752	2.587	-7.48%	**-1.05%**	OTM
90	5.308	5.575	5.373	-5.03%	**-1.22%**	
100	9.354	9.654	9.456	-3.21%	**-1.09%**	ATM
110	14.655	14.945	14.793	-1.98%	**-0.94%**	
120	21.052	21.291	21.217	-1.14%	**-0.79%**	ITM
130	28.333	28.498	28.522	**-0.58%**	-0.67%	

both models show stronger alignment with the theoretical values, but Ambient still performs slightly better in most cases.

Summary

The GBM experiments demonstrate the effectiveness of the Ambient Diffusion approach for simple financial time series generation. While both EDM and Ambient models successfully capture the basic dynamics of the GBM process, Ambient consistently outperforms EDM across all evaluation metrics. The key advantage lies in Ambient's ability to correct the variance amplification issue present in the standard EDM model. That results in better distributional alignment, more accurate parameter recovery, and improved performance in practical applications such as option pricing.

Table 5.5: Simulation setup for Correlated Geometric Brownian Motion (CGBM) datasets.

Parameter	**Dataset 1 (2 assets)**	**Dataset 2 (3 assets)**
Drift (μ)	[0.05, 0.05]	[0.05, 0.03, 0.07]
Volatility (ν)	[0.2, 0.2]	[0.2, 0.15, 0.25]
Initial Price (S_0)	[100, 100]	[100, 100, 100]
Number of Time Steps (N)	256	256
Number of Paths (M)	100,000	100,000
Time Horizon (T)	1.0	1.0
Correlation Coefficients (ρ_{ij})	$\rho_{12} = 0.3$	$\rho_{12} = 0.8$ $\rho_{13} = 0.4$ $\rho_{23} = 0.2$

5.3.2 Correlated GBM

Setup

We increase the complexity by introducing multivariate dependencies via Correlated Geometric Brownian Motion. This tests the models' ability to capture correlations between multiple assets with varying underlying dynamics.

We evaluate two scenarios of increasing complexity. In the first case, we use two correlated assets with identical parameters (drift and volatility) to isolate the models' ability to capture correlation structure. The second scenario uses three assets with different pairwise correlations and distinct underlying parameters, corresponding to more realistic market conditions. Parameters for both datasets are presented in Table 5.5.

Generated Samples

Figure 5.4 shows representative sample paths for both datasets. Visual inspection confirms that both models successfully generate realistic trajectories, though quantitative analysis is needed to assess the accuracy of correlation recovery.



(a) Dataset 1 – Training (b) Dataset 1 – EDM (c) Dataset 1 – Ambient

(d) Dataset 2 – Training (e) Dataset 2 – EDM (f) Dataset 2 – Ambient

Figure 5.4: Sample path comparison for CGBM datasets. Top row: 2-asset case; bottom row: 3-asset case. Columns show reference simulation, EDM-generated, and Ambient-generated trajectories.

Table 5.6: Distributional similarity metrics for the 2-asset and 3-asset CGBM datasets. Lower values indicate better alignment between generated and training log return distributions.

Dataset	Model	K-S Statistic	Wasserstein Distance
2-Asset	EDM	0.01234	0.00051
	Ambient	**0.00421**	**0.00017**
3-Asset	EDM	0.01116	0.00048
	Ambient	**0.00292**	**0.00013**

Distributional Similarity

We first assess distributional alignment using the K-S statistic and Wasserstein distance computed on the log return distributions across all assets. Table 5.6 shows that Ambient consistently outperforms EDM for both datasets. This demonstrates the robustness of the variance correction mechanism in multivariate settings.

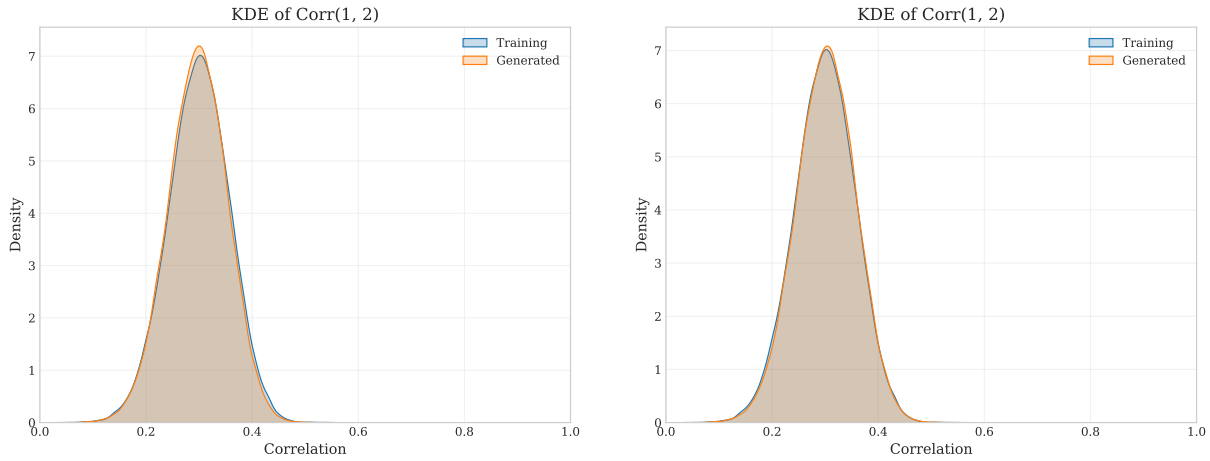
Correlation Analysis

We next perform correlation analysis through pathwise estimation of pairwise correlation coefficients and report the resulting distributions of estimates. Using Definition 4.4.5, we compute empirical correlations for each sample path and compare the distributions between training and generated data.

Figures 5.5 and 5.6 show the distributions of estimated correlation coefficients. Both models successfully capture the cross-asset correlations for both datasets with similar performance. For the 2-asset case (Figure 5.5), both EDM and Ambient accurately recover the target correlation of $\rho_{12} = 0.3$, with distributions closely matching the training data. Similarly, for the 3-asset case (Figure 5.6), both models effectively reproduce all three pairwise correlations ($\rho_{12} = 0.8$, $\rho_{13} = 0.4$, $\rho_{23} = 0.2$), demonstrating their ability to capture complex multivariate dependency structures.

Drift and Volatility Estimation

Both models successfully capture drift dynamics for both datasets. We provide complete drift estimation results in Appendix A and focus here on volatility estimation, where the



(a) EDM – ρ_{12} (b) Ambient – ρ_{12}

Figure 5.5: Estimated pairwise correlation distribution ρ_{12} for the 2-asset CGBM dataset.

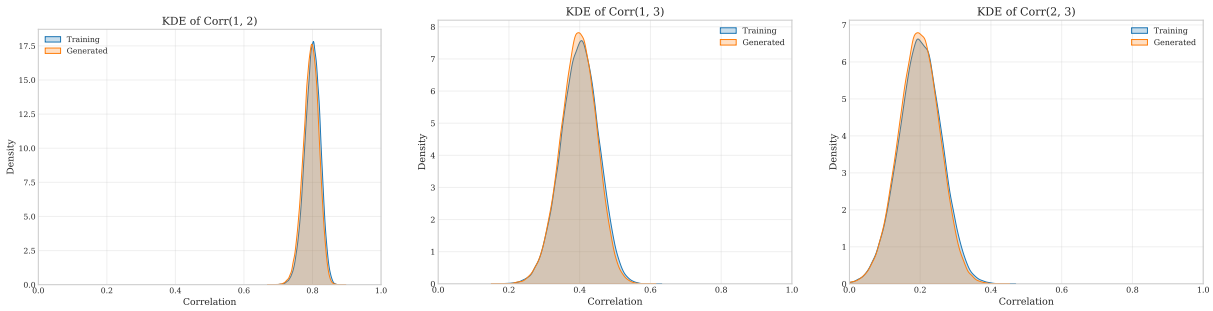
variance correction benefits of Ambient model are most apparent. We present results for the more complex 3-asset case with different volatility parameters ($\nu = [0.2, 0.15, 0.25]$), as the 2-asset case shows similar patterns (also provided in Appendix A).

Figure 5.7 shows the volatility estimation distributions for each asset. Consistent with the univariate GBM results, EDM exhibits systematic overestimation of volatility across all three assets, with distributions shifted toward higher values than the true parameters. In contrast, Ambient maintains close alignment with the training data distributions, accurately recovering the distinct volatility levels for each asset. This demonstrates that the variance correction mechanism remains effective in multivariate settings with different volatility parameters across assets.

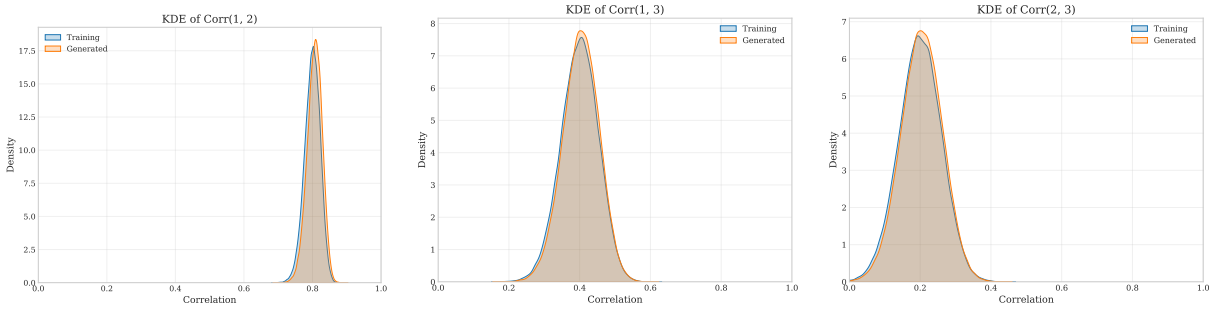
Option Pricing Results

We evaluate the practical utility of the generated samples through exotic option pricing on the 2-asset CGBM dataset. We consider basket options (written on the average of the asset prices), max options (on the maximum of the asset prices), and spread options (on the difference between two asset prices) across different moneyness levels. Monte Carlo pricing using the training data serves as the benchmark, with relative errors reported for both models.

The results are shown in Table 5.7. Both EDM and Ambient models produce prices

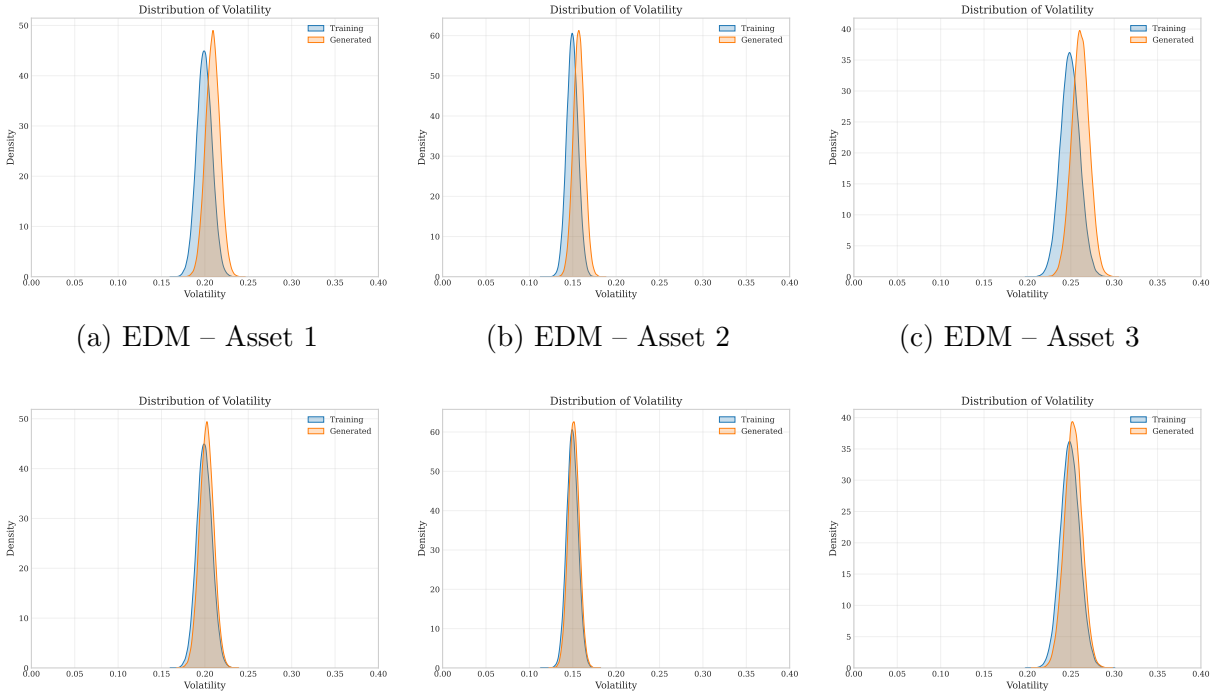


(a) EDM – ρ_{12} (b) EDM – ρ_{13} (c) EDM – ρ_{23}



(d) Ambient – ρ_{12} (e) Ambient – ρ_{13} (f) Ambient – ρ_{23}

Figure 5.6: Estimated pairwise correlation distributions for the 3-asset CGBM dataset. Each subplot shows the empirical distribution of the sample Pearson correlation ρ_{ij} between asset pairs for both EDM (top row) and Ambient (bottom row) models.



(a) EDM – Asset 1 (b) EDM – Asset 2 (c) EDM – Asset 3

(d) Ambient – Asset 1 (e) Ambient – Asset 2 (f) Ambient – Asset 3

Figure 5.7: Volatility estimation distributions for the 3-asset CGBM dataset.

that are generally close to the Monte Carlo reference, but Ambient achieves consistently lower relative errors across most strikes and payoffs. In particular, the spread of pricing errors is significantly wider for EDM, reaching up to 12.45% for deep out-of-the-money spread options, whereas Ambient errors remain within a narrow band of ±3% throughout. This highlights Ambient's improved stability. For call options, Ambient maintains sub-1% error even at OTM strikes, while EDM shows a clear upward bias. EDM tends to overprice the options, especially as moneyness decreases, while Ambient remains better aligned with the ground truth. Overall, the Ambient model demonstrates better robustness across both option structures and strike levels.

Summary

The CGBM experiments demonstrate the models' ability to handle multivariate financial time series with various complexity levels. While both EDM and Ambient successfully capture correlation structures across different asset combinations, Ambient consistently outperforms EDM in distributional similarity and volatility estimation, maintaining the variance correction benefits observed in the univariate case. Furthermore, option pricing results confirm the practical relevance of these improvements: Ambient produces significantly more accurate prices across a range of exotic derivatives, including basket, max, and spread options.

5.3.3 Heston Model

Setup

We evaluate model performance on data generated from the Heston stochastic volatility model, introduced in Section 3.2.1. This model presents a more complex structure compared to GBM, featuring a latent volatility process with mean-reverting dynamics. The setup used for generating the dataset is summarized in Table 5.8. The parameters correspond to a market environment with mean-reverting volatility that increases over time due to the initial variance ($v_0 = 0.1$) being below the long-run level ($\bar{v} = 0.4$). The negative correlation ($\rho = -0.3$) captures the empirically observed leverage effect where price declines tend to increase volatility.

The Heston model presents unique challenges due to its latent stochastic volatility process. We evaluate three approaches: standard EDM training, Ambient training with formula-based corruption level (using Equation 3.51), and Ambient training with manually tuned corruption level ($\sigma_{t_n} = 0.015$).

Table 5.7: Exotic option prices and relative errors for the 2-asset Correlated GBM dataset. Relative errors are computed with respect to Monte Carlo prices. Moneyness is defined relative to the average asset price ($S_0 = 100$).

Strike Price K	**Monte Carlo Price**	**EDM Price**	**Ambient Price**	**Error (EDM)**	**Error (Ambient)**	**Moneyness**
Basket Call						
90	15.715	15.936	15.785	1.41%	**0.45%**	ITM
100	9.003	9.211	9.045	2.31%	**0.46%**	ATM
110	4.512	4.681	4.533	3.73%	**0.45%**	OTM
120	1.989	2.108	2.002	5.96%	**0.66%**	OTM
Basket Put						
90	1.328	1.352	1.307	1.82%	**-1.59%**	OTM
100	4.129	4.140	4.080	**0.27%**	-1.20%	ATM
110	9.150	9.122	9.080	**-0.31%**	-0.77%	ITM
120	16.140	16.061	16.062	-0.49%	**-0.48%**	ITM
Max Call						
90	24.388	24.840	24.489	1.85%	**0.42%**	ITM
100	16.417	16.859	16.503	2.69%	**0.52%**	ATM
110	10.115	10.507	10.174	3.88%	**0.58%**	OTM
120	5.723	6.034	5.758	5.42%	**0.60%**	OTM
Max Put						
90	0.602	0.613	0.593	1.76%	**-1.57%**	OTM
100	2.143	2.145	2.118	**0.06%**	-1.16%	ATM
110	5.353	5.305	5.302	**-0.91%**	-0.97%	ITM
120	10.474	10.344	10.398	-1.25%	**-0.73%**	ITM
Spread Call						
5	7.188	7.513	7.004	4.51%	**-2.56%**	OTM
15	3.954	4.217	3.849	6.67%	**-2.64%**	OTM
25	1.991	2.180	1.941	9.47%	**-2.51%**	OTM
35	0.927	1.042	0.904	12.45%	**-2.40%**	OTM

Table 5.8: Simulation Setup for the Heston Stochastic Volatility Model

Parameter	**Value**
Mean Reversion Rate (κ)	1.0
Long-Run Variance ($\bar{v}$)	0.4
Volatility of Volatility (η)	0.1
Correlation (ρ) between Brownian motions	-0.3
Initial Variance (v_0)	0.1
Initial Price (S_0)	100
Number of Time Steps (N)	256
Number of Paths (M)	100,000
Time Horizon (T)	1.0

Table 5.9: Distributional similarity metrics and absolute errors for the Heston dataset. Mean and standard deviation are computed from log returns; absolute errors measure deviation from the training dataset statistics.

Model	**K-S Statistic**	**Wasserstein Distance**	**Mean Error (Absolute)**	**Std Error (Absolute)**
EDM	0.01256	0.00122	**0.00001**	0.00153
Ambient (tuned)	0.00613	0.00057	0.00004	0.00072
Ambient (formula)	**0.00405**	**0.00028**	0.00011	**0.00034**

Figure 5.8 shows representative sample paths generated by each approach compared to the reference Monte Carlo simulation. Visual inspection reveals that all models successfully generate realistic price trajectories with varying volatility patterns characteristic of the Heston process.

Distributional and Statistical Analysis

Table 5.9 shows the trade-offs between different approaches. The formula-based Ambient model achieves the best distributional similarity (lowest K-S statistic and Wasserstein distance) but introduces drift distortion. The tuned Ambient model provides a balanced compromise, outperforming EDM in distributional metrics while maintaining better drift accuracy than the formula-based approach.



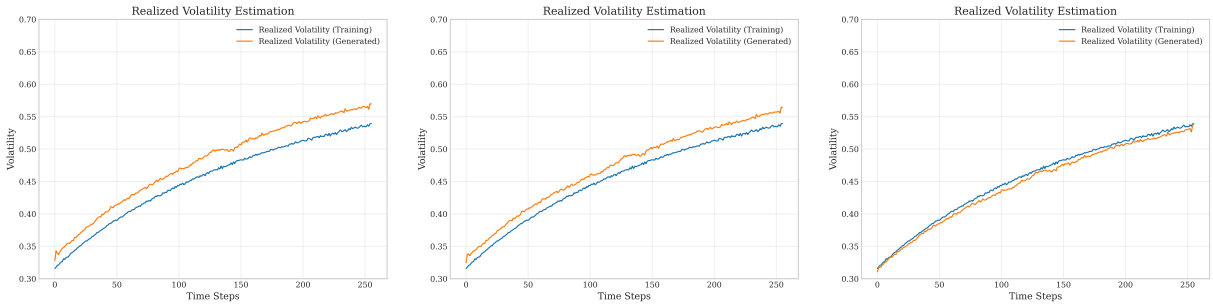
(a) Monte Carlo

(b) EDM

(c) Ambient (tuned)

(d) Ambient (formula)

Figure 5.8: Sample paths generated by different models for the Heston dataset. The top-left plot shows reference Monte Carlo simulations from the Heston SDE. All generative models were initialized with the same starting conditions.



(a) EDM (b) Ambient (tuned) (c) Ambient (formula)

Figure 5.9: Realized volatility estimation across generated paths for the Heston dataset. The Ambient model with formula-based σ_{t_n} better captures the heavy-tailed volatility dynamics.

Volatility Process Recovery

Figure 5.9 demonstrates the models' ability to capture the time-varying volatility structure of the Heston process. The plots show the evolution of realized volatility over time, computed as the cross-sectional volatility estimate at each time step across all generated paths (Definition 4.4.2). The blue line represents the training data, while the orange line shows the generated samples.

The formula-based Ambient model (c) most accurately reproduces both the increasing volatility trajectory and the magnitude of volatility levels, closely tracking the training data throughout the time horizon. The tuned Ambient model (b) shows some deviation from the training data pattern. EDM (a) shows even more significant overestimation of volatility compared to the true Heston process, failing to match the expected volatility behavior.

Option Pricing Performance

Table 5.10 highlights the practical implications of these trade-offs. While the formula-based Ambient model excels in distributional metrics, its drift distortion leads to significantly worse option pricing accuracy. In contrast, the tuned Ambient model delivers much more accurate prices, outperforming EDM across most strikes. The only exception is for deep in-the-money put options, where EDM achieves slightly better results. However, the error difference is less than 1%, which can likely be attributed to sampling variance or discretization artifacts.

Table 5.10: Relative error in European option pricing compared to theoretical prices on the Heston dataset. Moneyness is defined relative to $S_0 = 100$.

Strike Price K	**Error (EDM)**	**Error Ambient (tuned)**	**Error Ambient (formula)**	**Moneyness**
Call Options				
70	-3.05%	**-2.84%**	-5.92%	ITM
80	-3.91%	**-3.33%**	-6.54%	
90	-4.91%	**-3.85%**	-7.14%	
100	-6.07%	**-4.41%**	-7.72%	ATM
110	-7.35%	**-4.99%**	-8.28%	OTM
120	-8.68%	**-5.54%**	-8.77%	
130	-10.05%	**-6.04%**	-9.16%	
Put Options				
70	-5.97%	**1.23%**	11.05%	OTM
80	-4.32%	**1.29%**	9.32%	
90	-3.04%	**1.37%**	8.04%	
100	-2.09%	**1.41%**	7.00%	ATM
110	**-1.38%**	1.41%	6.17%	ITM
120	**-0.83%**	1.42%	5.49%	
130	**-0.41%**	1.41%	4.94%	

Table 5.11: Simulation Setup for the Merton Jump Diffusion Model

Parameter	**Value**
Drift (μ)	0.05
Volatility (ν)	0.3
Mean Jump Size (μ_J)	-0.3
Jump Volatility (σ_J)	0.3
Jump Intensity (λ)	1.0
Initial Price (S_0)	100
Number of Time Steps (N)	256
Number of Paths (M)	100,000
Time Horizon (T)	1.0

Summary

The Heston experiments highlight the complexity of modeling stochastic volatility processes and demonstrate important trade-offs between different performance metrics. While the formula-based Ambient approach achieves better distributional similarity and volatility process recovery, it suffers from drift distortion that severely impacts option pricing accuracy. In contrast, the manually tuned Ambient model provides a practical compromise, maintaining better statistical fidelity than EDM while delivering more accurate option prices across most strikes.

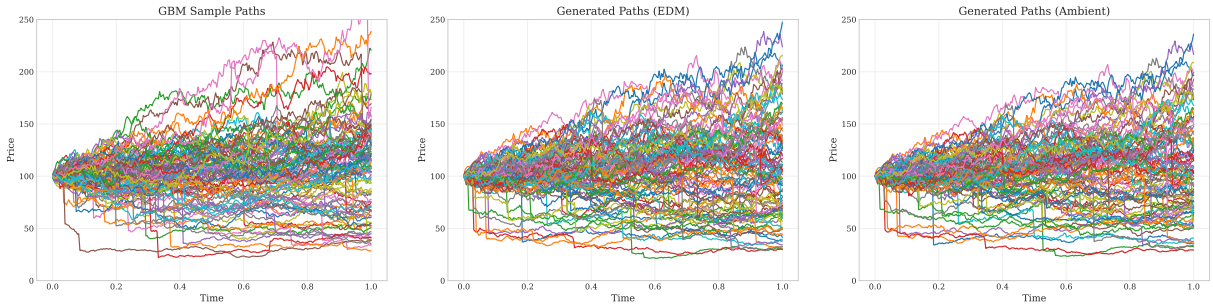
5.3.4 Merton Jump Diffusion Model

Setup

We conclude the synthetic experiments with the Merton Jump Diffusion (MJD) model, which incorporates discontinuous jumps into the standard GBM framework. This model tests the ability of diffusion models to capture sudden, large market changes that are typical of financial crises and extreme market events.

The simulation parameters are presented in Table 5.11. The configuration features negative mean jumps ($\mu_J = -0.3$) with moderate volatility ($\sigma_J = 0.3$) and relatively high jump intensity ($\lambda = 1.0$), corresponding to frequent downward market shocks.

Figure 5.10 shows representative sample paths for the MJD dataset. The reference



(a) Monte Carlo (b) EDM (c) Ambient

Figure 5.10: Sample paths generated for the Merton Jump Diffusion (MJD) dataset. The reference paths (left) are simulated using the true MJD SDE with jumps. The EDM (middle) and Ambient (right) models are trained to replicate these dynamics.

Table 5.12: Distributional similarity metrics for the MJD dataset. Lower values indicate better alignment between the generated and training log return distributions.

Model	**K-S Statistic**	**Wasserstein Distance**
EDM	0.01301	0.00093
Ambient	**0.00488**	**0.00044**

simulation (a) exhibits characteristic sudden downward jumps, while both EDM (b) and Ambient (c) models successfully generate paths with similar jump-diffusion behavior.

Distributional Similarity

Table 5.12 shows that Ambient significantly outperforms EDM in distributional alignment for the MJD dataset. The superior performance across both K-S statistic and Wasserstein distance demonstrates that Ambient maintains its variance correction advantages even in the presence of jump discontinuities, effectively capturing the heavy-tailed distribution of jump-diffusion processes.

Parameter Estimation

We assess the models' ability to recover the underlying MJD parameters through maximum likelihood estimation (Definition 4.4.3). Figure 5.11 shows the distributions of estimated

drift and volatility parameters. Both models successfully recover the drift parameter ($\mu = 0.05$). For volatility estimation, Ambient again demonstrates better performance, closely matching the training data distribution around the true value ($\nu = 0.3$), while EDM exhibits the overestimation observed in previous experiments.

Figure 5.12 presents the more challenging task of jump parameter recovery. Both models show very close alignment with training data estimates for both jump mean and jump volatility parameters. This demonstrates that both approaches can successfully capture the discontinuous jump dynamics.

Option Pricing Results

Table 5.13 demonstrates Ambient's superior performance in option pricing for the MJD dataset. Ambient achieves remarkably accurate pricing across all strikes and option types, with errors typically below 1% for calls and under 4% for puts. In contrast, EDM shows systematic overpricing with errors exceeding 10% for deep OTM calls and puts, reflecting its inability to capture the dynamics that significantly impact option valuations.

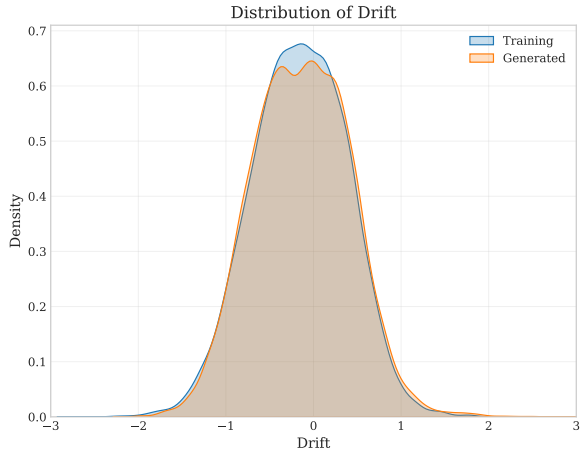
Summary

The MJD experiments demonstrate successful modeling of jump-diffusion processes, with Ambient consistently outperforming EDM across all metrics. Ambient achieves better distributional similarity, better parameter recovery, and remarkably accurate option pricing, highlighting the effectiveness of variance correction even for complex discontinuous dynamics.

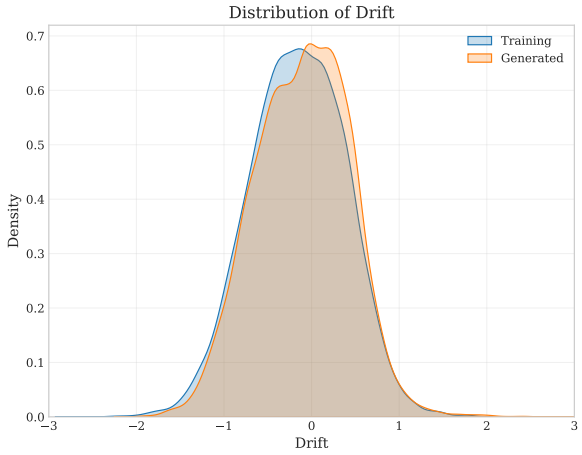
5.4 Experiments on Real-World Market Data

In this section, we evaluate model performance on real-world market data obtained through the Yahoo Finance API and processed as described in Section 3.2. We first present results for single-asset datasets consisting of adjusted closing prices, followed by multivariate OHLC (Open-High-Low-Close) data.

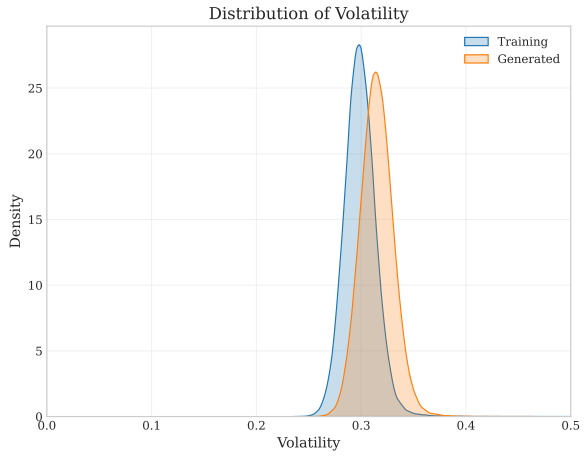
Our experiments use four assets representing diverse market dynamics: S&P 500 ETF (SPY) as a low-risk investment vehicle; Apple (AAPL) and Nvidia (NVDA) stocks representing medium and high volatility equities; and Bitcoin (BTC-USD) as a high-risk



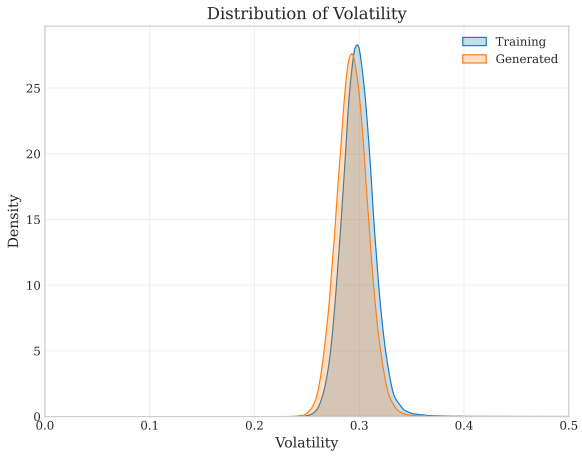
(a) Drift (EDM)



(b) Drift (Ambient)

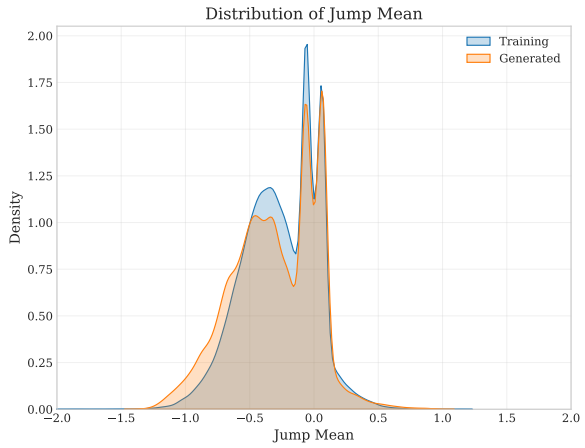


(c) Volatility (EDM)

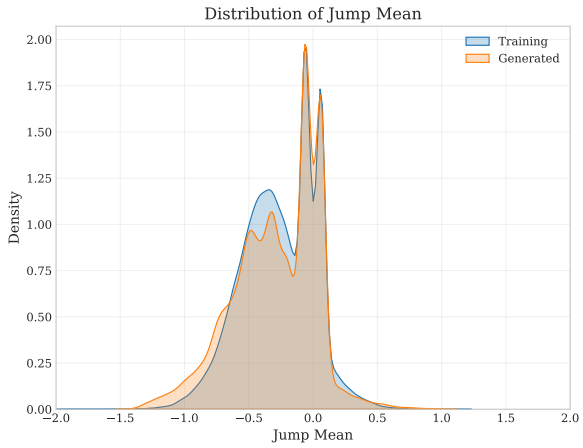


(d) Volatility (Ambient)

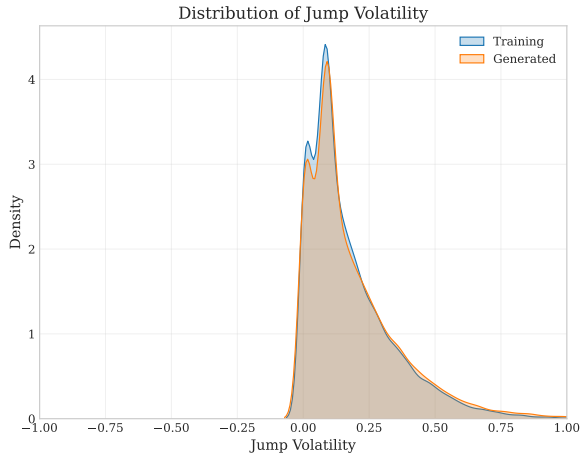
Figure 5.11: Estimated drift and volatility distributions for the MJD dataset. Left column: EDM model; right column: Ambient model.



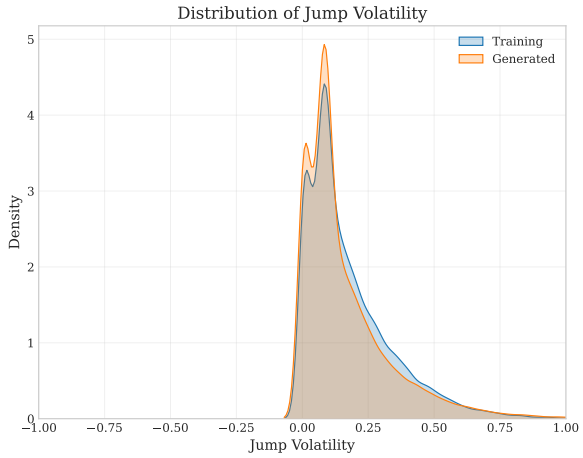
(a) Jump Mean (EDM)



(b) Jump Mean (Ambient)



(c) Jump Volatility (EDM)



(d) Jump Volatility (Ambient)

Figure 5.12: Estimated distributions of MJD jump parameters for EDM (left) and Ambient (right). Includes jump mean and jump volatility.

Table 5.13: European option prices and relative error compared to Monte Carlo prices on the MJD dataset. Moneyness is defined relative to $S_0 = 100$.

Strike Price K	**MC Price**	**EDM Price**	**Ambient Price**	**Error (EDM)**	**Error Ambient**	**Moneyness**
Call Options						
70	41.278	42.069	41.156	1.92%	**-0.30%**	
80	34.372	35.350	34.307	2.85%	**-0.19%**	ITM
90	28.209	29.322	28.172	3.95%	**-0.13%**	
100	22.816	24.008	22.791	5.23%	**-0.11%**	ATM
110	18.201	19.421	18.182	6.70%	**-0.11%**	
120	14.344	15.541	14.312	8.34%	**-0.23%**	OTM
130	11.182	12.308	11.133	10.07%	**-0.44%**	
Put Options						
70	4.610	5.153	4.779	11.77%	**3.66%**	
80	7.216	7.946	7.443	10.11%	**3.14%**	OTM
90	10.566	11.431	10.820	8.18%	**2.41%**	
100	14.686	15.629	14.951	6.42%	**1.81%**	ATM
110	19.583	20.554	19.854	4.96%	**1.39%**	
120	25.238	26.186	25.496	3.76%	**1.02%**	ITM
130	31.588	32.465	31.830	2.78%	**0.77%**	

Table 5.14: Summary of real asset datasets used for training and evaluation. Each path corresponds to a non-overlapping year of daily data (252 steps).

Asset	Date Range	Number of Paths	Steps per Path
AAPL	1980-12-12 to 2025-07-30	44	252
SPY	1993-01-29 to 2025-07-30	32	252
NVDA	1999-01-22 to 2025-07-30	26	252
BTC-USD	2014-09-17 to 2025-07-30	15	252

Table 5.15: Distributional similarity metrics for real asset datasets. Lower values indicate better alignment between generated and training log return distributions.

Asset	Model	K-S Statistic	Wasserstein Distance
SPY	EDM	0.04067	0.00090
	Ambient	**0.01153**	**0.00051**
AAPL	EDM	**0.01723**	0.00080
	Ambient	0.02002	**0.00059**
NVDA	EDM	0.05800	0.00720
	Ambient	**0.02411**	**0.00316**
BTC	EDM	0.01905	0.00147
	Ambient	**0.01799**	**0.00102**

cryptocurrency with atypical dynamics. Models are trained on historical price data segmented into non-overlapping sequences of 252 timesteps (approximately one trading year). Dataset details are summarized in Table 5.14.

5.4.1 Single-Asset Data

Univariate single-asset data consists of daily adjusted closing prices for each asset. Table 5.15 presents distributional similarity metrics comparing EDM and Ambient models across all four assets.

Ambient outperforms EDM for most assets, with particularly strong improvements for SPY and NVDA. AAPL shows mixed results with EDM achieving better K-S statistic but Ambient providing superior Wasserstein distance. Based on these results, we present

detailed analysis for SPY and BTC in this chapter, with additional results for AAPL and NVDA provided in Appendix A.

S&P 500 and Bitcoin datasets results

We begin with visual assessment of generated sample paths in Figure 5.13. All paths are normalized to start at an initial price of 100 for comparison purposes. The generated data successfully captures the distinct dynamics and volatility regimes of both SPY and BTC datasets. SPY exhibits relatively stable growth patterns with occasional downturns, while BTC demonstrates the high volatility and dramatic price swings typical for cryptocurrency markets.

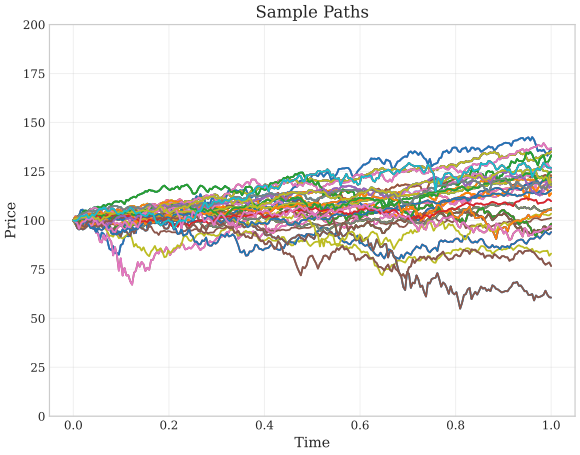
To better assess the dynamics of the generated data, we estimate annualized drift and volatility parameter distributions and compare them to the training dataset, following the approach used for synthetic experiments. Figure 5.14 presents the results for both assets. The model successfully captures volatility distributions for both datasets, with generated samples closely matching the training data. For drift estimation, SPY shows good alignment between generated and training distributions. BTC exhibits a different distributional shape for drift estimates, though the mean and spread are well captured. This effect is likely due to the limited number of training samples in the BTC dataset (15 paths versus 32 for SPY).

Finally, we evaluate tail behavior using Value at Risk (VaR) and Conditional Value at Risk (CVaR) metrics (Definition 4.5.1). Figure 5.15 shows that the model captures risk characteristics reasonably well for both assets. SPY demonstrates close alignment between generated and training distributions for both risk measures. BTC shows slightly more deviation, which is likely due to both the smaller training set size and the greater volatility and randomness of cryptocurrency returns.

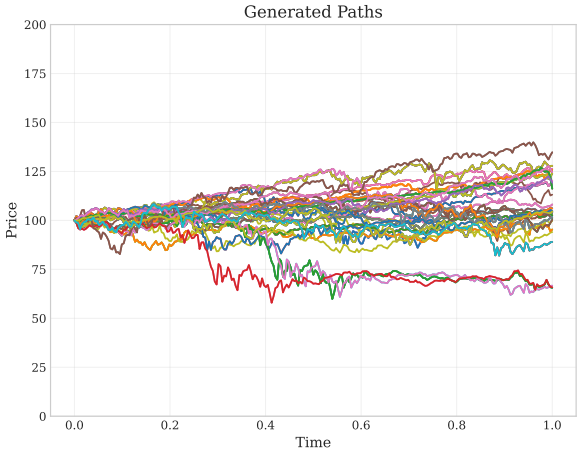
5.4.2 Open-High-Low-Close Structures Data

For multivariate OHLC analysis, we follow a similar setup to the univariate case but use all four daily price components (open, high, low, close) instead of only adjusted closing prices. This increases the complexity by capturing intraday price relationships and correlations between the different price levels.

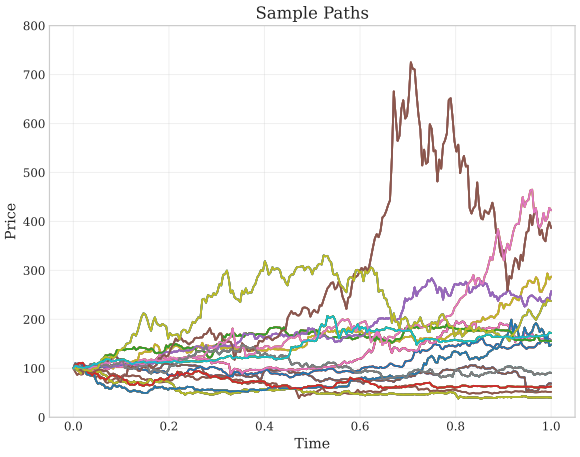
Table 5.16 presents distributional similarity metrics for OHLC log return distributions. The results show mixed performance between EDM and Ambient models across different



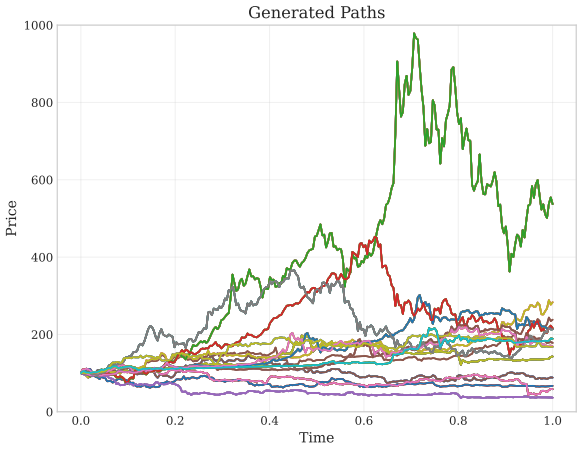
(a) Training Paths (SPY)



(b) Generated Paths (Ambient)

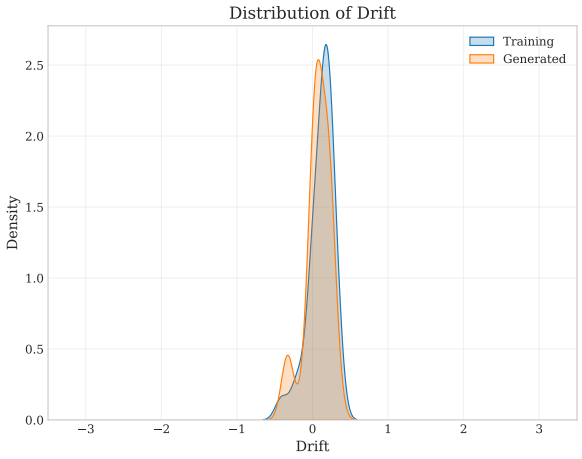


(c) Training Paths (BTC)

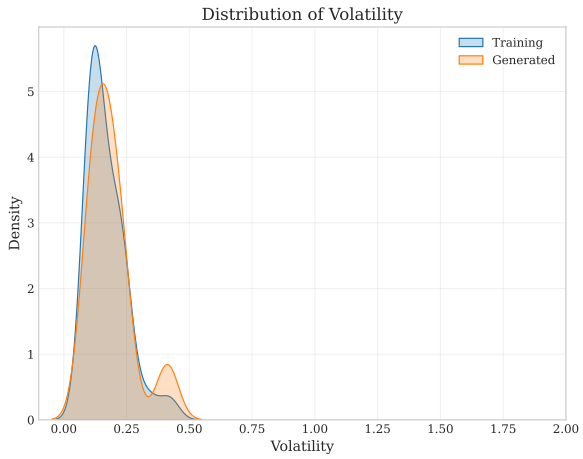


(d) Generated Paths (Ambient)

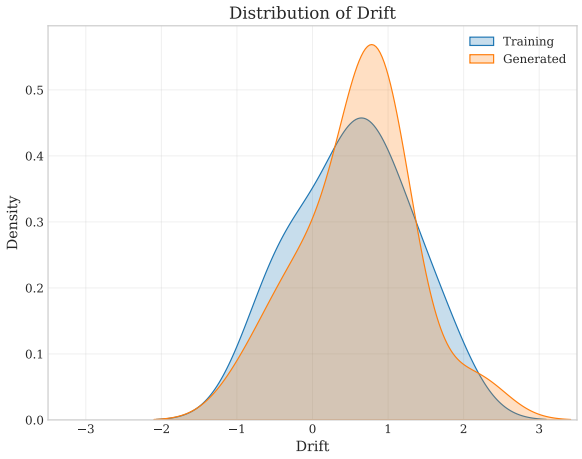
Figure 5.13: Sample paths of prices for SPY (top row) and BTC (bottom row). Left: original dataset. Right: paths generated by the Ambient model.



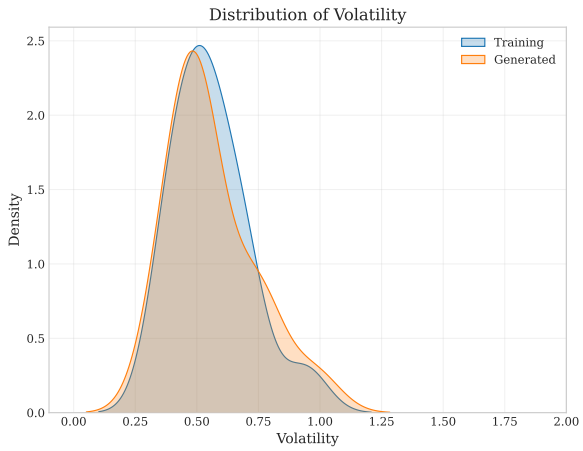
(a) Drift Estimate (SPY)



(b) Volatility Estimate (SPY)

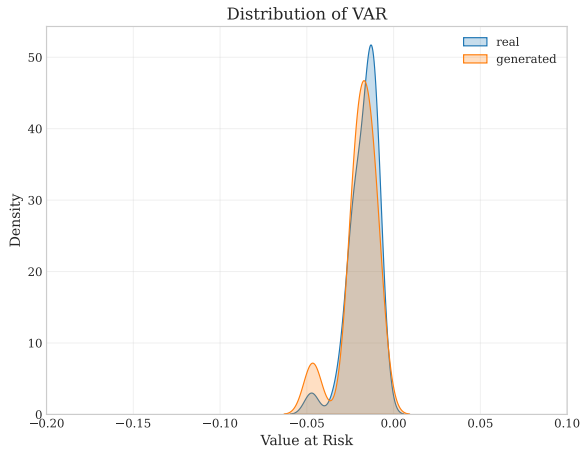


(c) Drift Estimate (BTC)

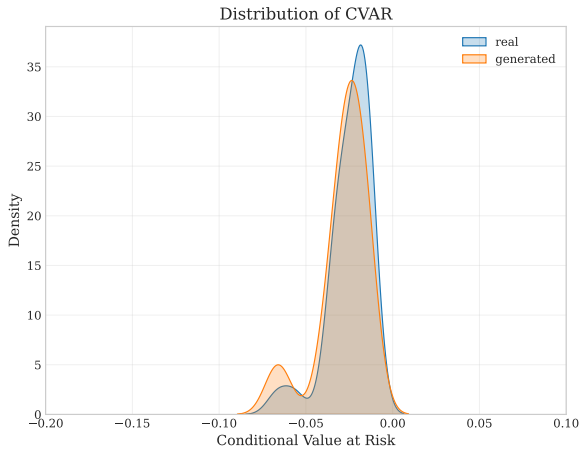


(d) Volatility Estimate (BTC)

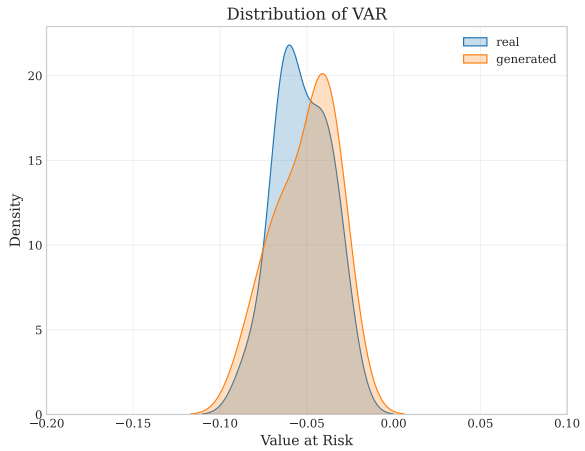
Figure 5.14: Estimated drift and volatility distributions using the Ambient model. Top row: SPY. Bottom row: BTC.



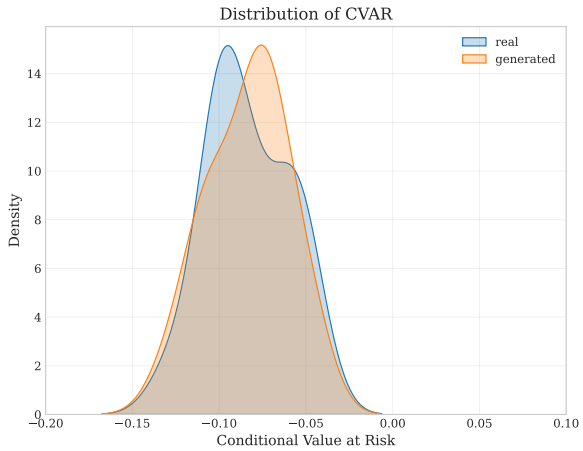
(a) SPY: VaR Estimate



(b) SPY: CVaR Estimate



(c) BTC: VaR Estimate



(d) BTC: CVaR Estimate

Figure 5.15: Estimated Value at Risk (VaR) and Conditional Value at Risk (CVaR) distributions using the Ambient model. Top: SPY. Bottom: BTC.

Table 5.16: Distributional similarity metrics for OHLC log return distributions across real assets. Lower values indicate better alignment between generated and training data.

Asset	Model	K-S Statistic	Wasserstein Distance
SPY	EDM	**0.01039**	**0.00027**
	Ambient	0.01150	0.00034
AAPL	EDM	0.03100	**0.00084**
	Ambient	**0.01986**	0.00090
NVDA	EDM	0.03659	0.00505
	Ambient	**0.00950**	**0.00118**
BTC	EDM	0.02738	0.00294
	Ambient	**0.02136**	**0.00126**

assets. SPY favors EDM slightly, while AAPL, NVDA, and BTC show better performance with Ambient. Notably, NVDA demonstrates the largest improvement with Ambient, consistent with the univariate results where this high-volatility stock benefited most from the variance correction mechanism.

Figure 5.16 demonstrates the model's ability to generate realistic OHLC data through candlestick chart visualization. Each candlestick represents one trading day, with the body showing the opening and closing prices (green for positive days, red for negative days) and the wicks indicating the daily high and low prices.

The generated candlestick chart (b) successfully reproduces key characteristics of real Bitcoin trading patterns (a), including realistic intraday price ranges, appropriate proportions of positive and negative trading days, and believable volatility clustering. This visual assessment confirms that the Ambient model can generate coherent OHLC sequences that maintain the internal consistency required for realistic market data.

To assess the model's ability to capture the dependencies between OHLC components, we analyze pairwise correlation distributions and mean correlation matrices. Figure 5.17 shows the distributions of pathwise correlation coefficients for all six OHLC pairs. The model successfully reproduces the correlation structure across most pairs, with particularly strong alignment for Open-High, Open-Low, and High-Close correlations. Some deviations are observed for Open-Close, High-Low, and Low-Close pairs, where the generated distributions show slightly different shapes while maintaining similar means.

Figure 5.18 presents the mean correlation matrices, providing a summary view of the

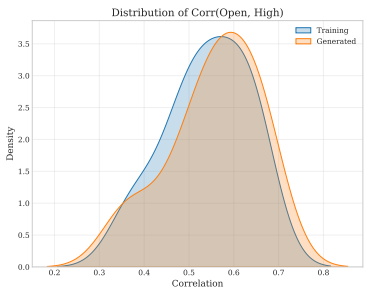


(a) Real candlestick chart

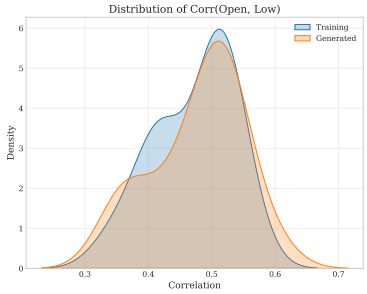


(b) Generated candlestick chart (Ambient)

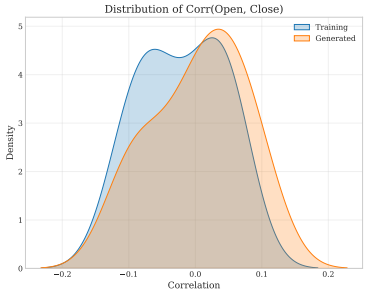
Figure 5.16: Comparison of real and generated OHLC data using candlestick plots for BTC.



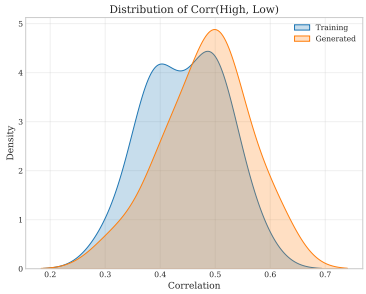
(a) Corr(Open, High)



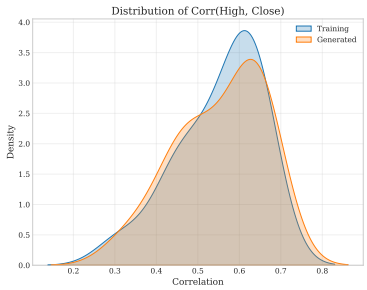
(b) Corr(Open, Low)



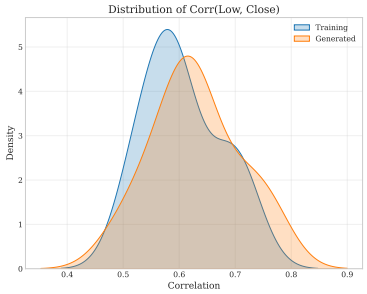
(c) Corr(Open, Close)



(d) Corr(High, Low)

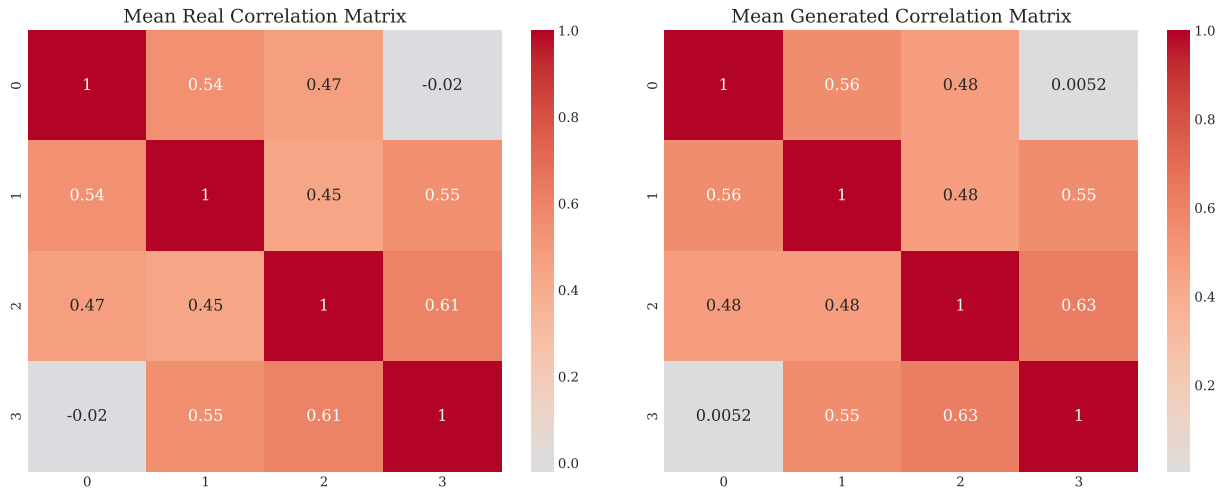


(e) Corr(High, Close)



(f) Corr(Low, Close)

Figure 5.17: Pairwise pathwise correlation coefficient distributions for BTC OHLC data (training vs generated).



(a) Mean correlation matrix (Training) (b) Mean correlation matrix (Generated)

Figure 5.18: Mean correlation matrices of OHLC data for BTC: training vs generated data.

overall correlation structure. The generated data closely matches the training data correlation pattern. This demonstrates that the model captures the essential feature that opening and closing prices within the same day are largely independent, while intraday extremes (high and low) maintain strong correlations with the opening price.

5.5 Summary

In this chapter, we systematically evaluated EDM and Ambient diffusion models on both synthetic and real-world financial time series. Ambient diffusion consistently improved distributional similarity, parameter estimation, and option pricing accuracy, especially for challenging settings like high-volatility assets and processes with jumps. On real-world market data, both models captured key statistical and risk features, with Ambient generally showing stronger performance.

These results highlight the strengths of diffusion-based generative modeling in finance, and provide a basis for the discussion and future work in the following sections.

Chapter 6

Conclusion

In this thesis, we proposed a data-driven approach for generating financial time series that, in contrast to traditional models, does not rely on explicit model selection or parameter calibration. We adapted a leading diffusion architecture [37, 71] originally developed for image synthesis to multivariate temporal data and introduced a variance correction mechanism through the Ambient Denoising Score Matching loss [18]. Building on this framework, we derived an analytical solution for selecting the ambient noise level. Furthermore, we developed a comprehensive evaluation methodology for generated time series, combining distributional similarity metrics with finance-specific measures and risk-based criteria.

We validated the proposed approach through extensive experiments on both synthetic and real-world datasets. Synthetic experiments covered well-established financial models, including Geometric Brownian Motion, the Heston stochastic volatility process, and the Merton jump-diffusion model. Real-world experiments included datasets such as the S&P 500 ETF (SPY), volatile equities (NVDA), and cryptocurrency markets (BTC-USD). The results consistently demonstrated that the Ambient Diffusion model with variance correction improves distributional alignment, volatility recovery, and option pricing accuracy compared to the EDM baseline. The model was able to reproduce key features of synthetic processes, such as jumps and stochastic volatility, while also capturing the distributional properties and dynamics of real-world market data. Our approach is sensitive to the choice of ambient noise level: while the proposed rule is effective in many settings, manual tuning may be required to balance drift accuracy and variance correction. In addition, limited sample sizes (e.g., BTC) remain a challenge, a difficulty common to many deep learning applications.

This work has focused on unconditional generation of financial time series. However,

the broader application of diffusion models to financial modeling remains largely unexplored. Two natural extensions are conditional tasks such as time series forecasting and imputation. Beyond these, diffusion models have shown promise in other domains for anomaly detection [82, 81], suggesting a potential direction for detecting rare events in financial markets.

Finally, our variance correction mechanism was inspired by the Ambient Diffusion framework [18], part of a growing research effort on signal recovery and training with noisy data [17]. These methods are increasingly applied to inverse problems such as denoising [1], and further extending them to the financial domain represents an exciting opportunity for future research.

In summary, this thesis has demonstrated that diffusion-based generative models provide a flexible and powerful framework for modeling financial time series, with clear advantages over traditional approaches. Our approach opens new directions for the use of diffusion models in finance, where their ability to learn directly from data and capture complex stochastic dynamics can contribute to improvements in quantitative modeling.

References

[1] Asad Aali, Giannis Daras, Brett Levac, Sidharth Kumar, Alexandros G. Dimakis, and Jonathan I. Tamir. Ambient Diffusion Posterior Sampling: Solving Inverse Problems with Diffusion Models Trained on Corrupted Data, April 2025.

[2] Brian D. O. Anderson. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 12(3):313–326, May 1982.

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN, December 2017.

[4] Paul A. Bereuter, Benjamin Stahl, Mark D. Plumbley, and Alois Sontacchi. Towards Reliable Objective Evaluation Metrics for Generative Singing Voice Separation Models, July 2025.

[5] Fischer Black and Myron Scholes. The Pricing of Options and Corporate Liabilities. *The Journal of Political Economy*, 81(3):637–654, 1973.

[6] Ashish Bora, Eric Price, and Alexandros G. Dimakis. AmbientGAN: Generative models from lossy measurements. In *International Conference on Learning Representations*, February 2018.

[7] Peter J. Brockwell. *Time Series: Theory and Methods*. Springer Series in Statistics. Springer New York, New York, NY, 2nd ed. 1991. edition, 1991.

[8] Tong Che, Ruixiang Zhang, Jascha Sohl-Dickstein, Hugo Larochelle, Liam Paull, Yuan Cao, and Yoshua Bengio. Your GAN is Secretly an Energy-based Model and You Should use Discriminator Driven Latent Sampling, July 2021.

[9] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating Long Sequences with Sparse Transformers, April 2019.

[10] Yoonjin Chung, Pilsun Eu, Junwon Lee, Keunwoo Choi, Juhan Nam, and Ben Sangbae Chon. KAD: No More FAD! An Effective and Efficient Evaluation Metric for Audio Generation, March 2025.

[11] R. Cont. Empirical properties of asset returns: Stylized facts and statistical issues. *Quantitative Finance*, 1(2):223–236, February 2001.

[12] Rama Cont and Peter Tankov. *Financial Modelling with Jump Processes*. Chapman & Hall/CRC Financial Mathematics Series. Chapman & Hall/CRC, Boca Raton, Fla, 2004.

[13] John C. Cox, Jonathan E. Ingersoll, and Stephen A. Ross. A Theory of the Term Structure of Interest Rates. *Econometrica*, 53(2):385–407, 1985.

[14] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. Generative Adversarial Networks: An Overview. *IEEE Signal Processing Magazine*, 35(1):53–65, January 2018.

[15] Giannis Daras, Yeshwanth Cherapanamjeri, and Constantinos Daskalakis. How much is a noisy image worth? Data Scaling Laws for Ambient Diffusion, November 2024.

[16] Giannis Daras, Yuval Dagan, Alexandros G. Dimakis, and Constantinos Daskalakis. Consistent Diffusion Models: Mitigating Sampling Drift by Learning to be Consistent, February 2023.

[17] Giannis Daras, Adrian Rodriguez-Munoz, Adam Klivans, Antonio Torralba, and Constantinos Daskalakis. Ambient Diffusion Omni: Training Good Models with Bad Data, June 2025.

[18] Giannis Daras, Kulin Shah, Yuval Dagan, Aravind Gollakota, Alexandros G. Dimakis, and Adam Klivans. Ambient Diffusion: Learning Clean Distributions from Corrupted Data, May 2023.

[19] Prafulla Dhariwal and Alex Nichol. Diffusion Models Beat GANs on Image Synthesis, June 2021.

[20] Carl Doersch. Tutorial on Variational Autoencoders, January 2021.

[21] Mihai Dogariu, Liviu-Daniel Ştefan, Bogdan Andrei Boteanu, Claudiu Lamba, Bomi Kim, and Bogdan Ionescu. Generation of Realistic Synthetic Financial Time-series. *ACM Trans. Multimedia Comput. Commun. Appl.*, 18(4):96:1–96:27, March 2022.

[22] Bradley Efron. Tweedie's Formula and Selection Bias. *Journal of the American Statistical Association*, 106(496):1602–1614, December 2011.

[23] Christian Francq and Jean-Michel Zakoian. *GARCH Models: Structure, Statistical Inference and Financial Applications*. Wiley, 1 edition, May 2019.

[24] Samir Yitzhak Gadre, Gabriel Ilharco, Alex Fang, Jonathan Hayase, Georgios Smyrnis, Thao Nguyen, Ryan Marten, Mitchell Wortsman, Dhruba Ghosh, Jieyu Zhang, Eyal Orgad, Rahim Entezari, Giannis Daras, Sarah Pratt, Vivek Ramanujan, Yonatan Bitton, Kalyani Marathe, Stephen Mussmann, Richard Vencu, Mehdi Cherti, Ranjay Krishna, Pang Wei Koh, Olga Saukh, Alexander Ratner, Shuran Song, Hannaneh Hajishirzi, Ali Farhadi, Romain Beaumont, Sewoong Oh, Alex Dimakis, Jenia Jitsev, Yair Carmon, Vaishaal Shankar, and Ludwig Schmidt. DataComp: In search of the next generation of multimodal datasets, October 2023.

[25] Hamed Ghoddusi, Germán G. Creamer, and Nima Rafizadeh. Machine learning in energy economics and finance: A review. *Energy Economics*, 81:709–727, June 2019.

[26] Paul Glasserman. *Monte Carlo Methods in Financial Engineering*. Number 53 in Applications of Mathematics : Stochastic Modelling and Applied Probability. Springer, New York, NY, softcover version of original hardcover ed. 2003 edition, 2010.

[27] John W. Goodell, Satish Kumar, Weng Marc Lim, and Debidutta Pattnaik. Artificial intelligence and machine learning in finance: Identifying foundations, themes, and research clusters from bibliometric analysis. *Journal of Behavioral and Experimental Finance*, 32:100577, December 2021.

[28] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks, June 2014.

[29] Bruno Miranda Henrique, Vinicius Amorim Sobreiro, and Herbert Kimura. Literature review: Machine learning techniques applied to financial market prediction. *Expert Systems with Applications*, 124:226–251, June 2019.

[30] Steven L. Heston. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *The Review of Financial Studies*, 6(2):327–343, April 2015.

[31] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs Trained by a Two Time-Scale Update Rule Converge to a Local

Nash Equilibrium. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[32] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models, December 2020.

[33] John Hull. *Options, Futures, and Other Derivatives*. Pearson Education, 2010.

[34] Aapo Hyvärinen. Estimation of Non-Normalized Statistical Models by Score Matching. *Journal of Machine Learning Research*, 6(24):695–709, 2005.

[35] IDC and Statista. Estimated value of the banking sector's artificial intelligence (ai) and generative artificial intelligence (gen ai) spending worldwide in 2023 and 2024, with forecasts until 2028 (in billion u.s. dollars). Statista, August 2024. [Graph]. Retrieved August 22, 2025.

[36] Ioannis Karatzas and Steven E. Shreve. *Brownian Motion and Stochastic Calculus*, volume 113 of *Graduate Texts in Mathematics*. Springer, New York, NY, 1998.

[37] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the Design Space of Diffusion-Based Generative Models, October 2022.

[38] Bahjat Kawar, Noam Elata, Tomer Michaeli, and Michael Elad. GSURE-Based Diffusion Model Training with Corrupted Data, June 2024.

[39] Onur Keleş, M. Akın Yılmaz, A. Murat Tekalp, Cansu Korkmaz, and Zafer Dogan. On the Computation of PSNR for a Set of Images or Video, April 2021.

[40] Kevin Kilgour, Mauricio Zuluaga, Dominik Roblek, and Matthew Sharifi. Fréchet Audio Distance: A Reference-Free Metric for Evaluating Music Enhancement Algorithms. In *Interspeech 2019*, ISCA, September 2019. ISCA.

[41] Kwanyoung Kim and Jong Chul Ye. Noise2Score: Tweedie's Approach to Self-Supervised Image Denoising without Clean Images, October 2021.

[42] Taesup Kim and Yoshua Bengio. Deep Directed Generative Models with Energy-Based Probability Estimation, June 2016.

[43] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017.

[44] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes, December 2022.

[45] Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. DiffWave: A Versatile Diffusion Model for Audio Synthesis, March 2021.

[46] Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, Saurabh Garg, Rui Xin, Niklas Muennighoff, Reinhard Heckel, Jean Mercat, Mayee Chen, Suchin Gururangan, Mitchell Wortsman, Alon Albalak, Yonatan Bitton, Marianna Nezhurina, Amro Abbas, Cheng-Yu Hsieh, Dhruba Ghosh, Josh Gardner, Maciej Kilian, Hanlin Zhang, Rulin Shao, Sarah Pratt, Sunny Sanyal, Gabriel Ilharco, Giannis Daras, Kalyani Marathe, Aaron Gokaslan, Jieyu Zhang, Khyathi Chandu, Thao Nguyen, Igor Vasiljevic, Sham Kakade, Shuran Song, Sujay Sanghavi, Fartash Faghri, Sewoong Oh, Luke Zettlemoyer, Kyle Lo, Alaaeldin El-Nouby, Hadi Pouransari, Alexander Toshev, Stephanie Wang, Dirk Groeneveld, Luca Soldaini, Pang Wei Koh, Jenia Jitsev, Thomas Kollar, Alexandros G. Dimakis, Yair Carmon, Achal Dave, Ludwig Schmidt, and Vaishaal Shankar. DataComp-LM: In search of the next generation of training sets for language models, April 2025.

[47] Yinheng Li, Shaofei Wang, Han Ding, and Hang Chen. Large Language Models in Finance: A Survey. In *ICAIF - ACM Int. Conf. AI Financ.*, pages 374–382. Association for Computing Machinery, Inc, 2023.

[48] James Lucas, George Tucker, Roger Grosse, and Mohammad Norouzi. Don't Blame the ELBO! A Linear VAE Perspective on Posterior Collapse, November 2019.

[49] Frank J. Massey Jr. The Kolmogorov-Smirnov Test for Goodness of Fit. *Journal of the American Statistical Association*, 46(253):68–78, March 1951.

[50] Caspar Meijer and Lydia Y. Chen. The Rise of Diffusion Models in Time-Series Forecasting, January 2024.

[51] Andrew Melnik, Michal Ljubljanac, Cong Lu, Qi Yan, Weiming Ren, and Helge Ritter. Video Diffusion Models: A Survey, November 2024.

[52] Chenlin Meng, Yang Song, Wenzhe Li, and Stefano Ermon. Estimating High Order Gradients of the Data Distribution by Denoising, November 2021.

[53] Robert C. Merton. Option pricing when underlying stock returns are discontinuous. *Journal of Financial Economics*, 3(1):125–144, January 1976.

[54] Carlo Milana and Arvind Ashta. Artificial intelligence techniques in finance and financial markets: A survey of the literature. *Strategic Change*, 30(3):189–209, 2021.

[55] Rihab Najem, Ayoub Bahnasse, Meryem Fakhouri Amr, and Mohamed Talea. Advanced AI and big data techniques in E-finance: A comprehensive survey. *Discover Artificial Intelligence*, 5(1), 2025.

[56] Alex Nichol and Prafulla Dhariwal. Improved Denoising Diffusion Probabilistic Models, February 2021.

[57] Bernt Øksendal. *Stochastic Differential Equations*. Universitext. Springer, Berlin, Heidelberg, 2003.

[58] Jinseong Park, Hyungjin Ko, and Jaewook Lee. Modeling Asset Price Process: An Approach for Imaging Price Chart with Generative Diffusion Models. *Computational Economics*, 66(1):349–375, July 2025.

[59] Danilo Jimenez Rezende and Shakir Mohamed. Variational Inference with Normalizing Flows, June 2016.

[60] Herbert E. Robbins. An Empirical Bayes Approach to Statistics. In Samuel Kotz and Norman L. Johnson, editors, *Breakthroughs in Statistics: Foundations and Basic Theory*, pages 388–394. Springer, New York, NY, 1992.

[61] R. Tyrrell Rockafellar and Stanislav Uryasev. Optimization of conditional value-at-risk. *The Journal of Risk*, 2(3):21–41, 2000.

[62] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-Resolution Image Synthesis with Latent Diffusion Models, April 2022.

[63] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation, May 2015.

[64] Ahmed Salih and Hani Hagras. Towards a Type-2 Fuzzy Logic Based System for Decision Support to Minimize Financial Default in Banking Sector. In *2018 10th Computer Science and Electronic Engineering (CEEC)*, pages 46–49, September 2018.

[65] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved Techniques for Training GANs, June 2016.

[66] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep Unsupervised Learning using Nonequilibrium Thermodynamics, November 2015.

[67] Michael Soloveitchik, Tzvi Diskin, Efrat Morin, and Ami Wiesel. Conditional Frechet Inception Distance, February 2022.

[68] Yang Song and Stefano Ermon. Generative Modeling by Estimating Gradients of the Data Distribution, October 2020.

[69] Yang Song and Stefano Ermon. Improved Techniques for Training Score-Based Generative Models, October 2020.

[70] Yang Song, Sahaj Garg, Jiaxin Shi, and Stefano Ermon. Sliced Score Matching: A Scalable Approach to Density and Score Estimation, June 2019.

[71] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-Based Generative Modeling through Stochastic Differential Equations, February 2021.

[72] Tomonori Takahashi and Takayuki Mizuno. Generation of synthetic financial time series by diffusion models. *Quantitative Finance*, 2025.

[73] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A Generative Model for Raw Audio, September 2016.

[74] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, August 2023.

[75] Pascal Vincent. A Connection Between Score Matching and Denoising Autoencoders. *Neural Computation*, 23(7):1661–1674, July 2011.

[76] Yifei Wang, Weimin Bai, Weijian Luo, Wenzheng Chen, and He Sun. Integrating Amortized Inference with Diffusion Models for Learning Clean Distribution from Corrupted Images, July 2024.

[77] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, April 2004.

[78] Magnus Wiese, Robert Knobloch, Ralf Korn, and Peter Kretschmer. Quant GANs: Deep Generation of Financial Time Series. *Quantitative Finance*, 20(9):1419–1440, September 2020.

[79] Paul Wilmott. *Paul Wilmott on Quantitative Finance*. Wiley, Hoboken, N.J., 2014.

[80] Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. BloombergGPT: A Large Language Model for Finance, December 2023.

[81] Julian Wyatt, Adam Leach, Sebastian M. Schmon, and Chris G. Willcocks. AnoDDPM: Anomaly Detection with Denoising Diffusion Probabilistic Models using Simplex Noise. In *IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recogn. Workshops*, volume 2022-June, pages 649–655. IEEE Computer Society, 2022.

[82] Chaocheng Yang, Tingyin Wang, and Xuanhui Yan. DDMT: Denoising Diffusion Mask Transformer Models for Multivariate Time Series Anomaly Detection, October 2023.

[83] Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. Diffusion Models: A Comprehensive Survey of Methods and Applications, December 2024.

[84] Peter Guangping Zhang. *Exotic Options: A Guide To Second Generation Options*. World Scientific, January 1997.

[85] You Zhu, Li Zhou, Chi Xie, Gang-Jin Wang, and Truong V. Nguyen. Forecasting SMEs' credit risk in supply chain finance with an enhanced hybrid ensemble machine learning approach. *International Journal of Production Economics*, 211:22–33, May 2019.

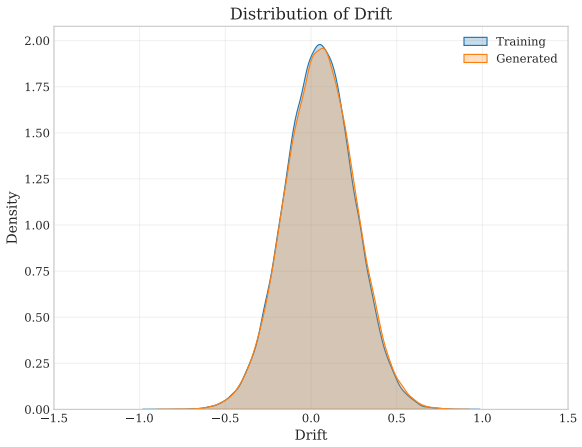
APPENDICES

Appendix A

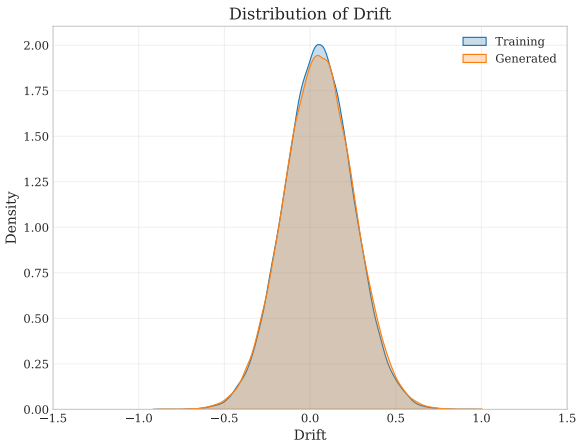
Additional Results

A.1 Additional Results for the CGBM Dataset

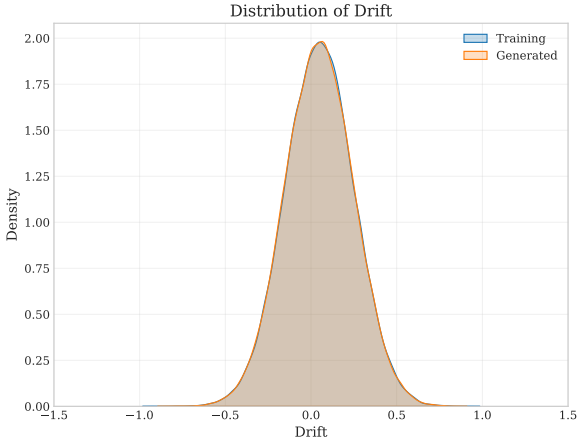
A.2 Additional Results for Single-Asset Datasets



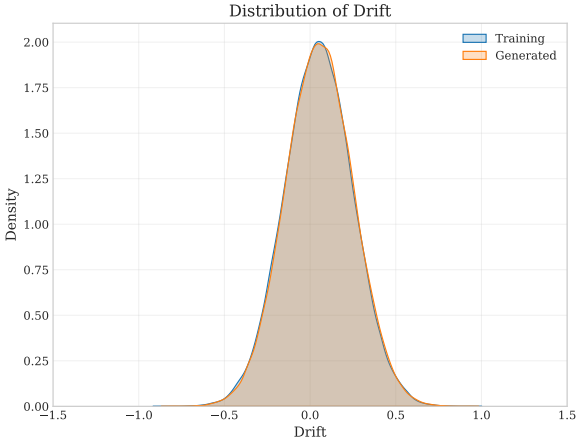
(a) EDM – Asset 1



(b) EDM – Asset 2

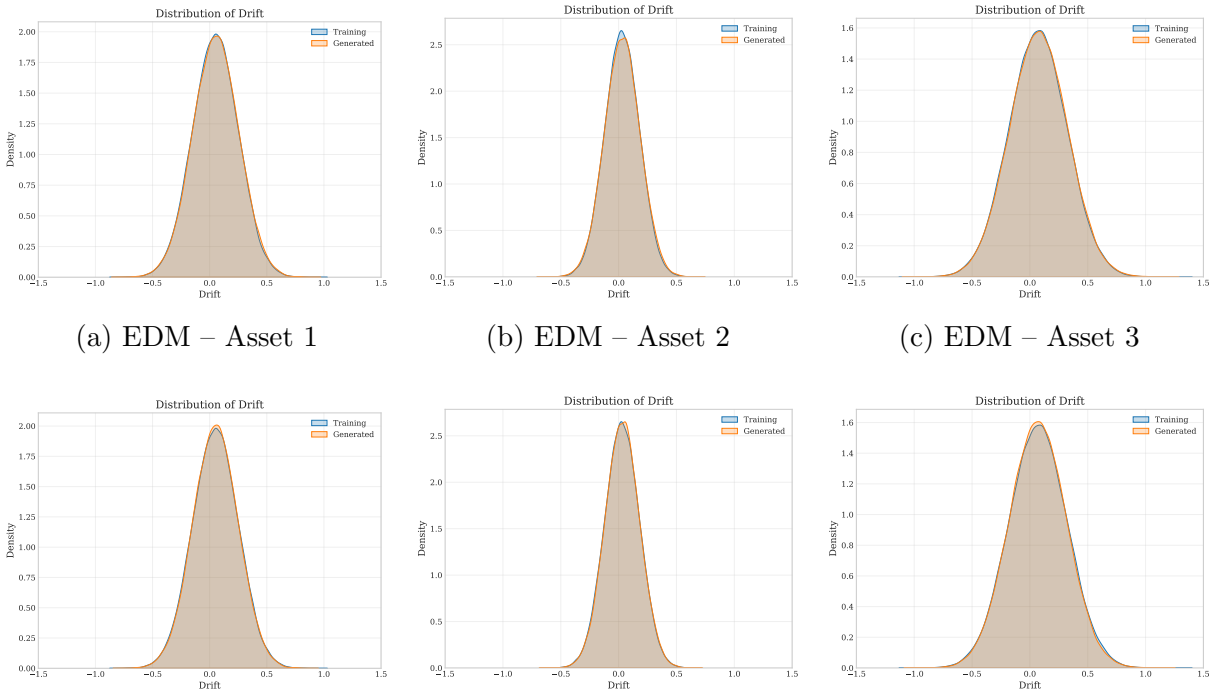


(c) Ambient – Asset 1



(d) Ambient – Asset 2

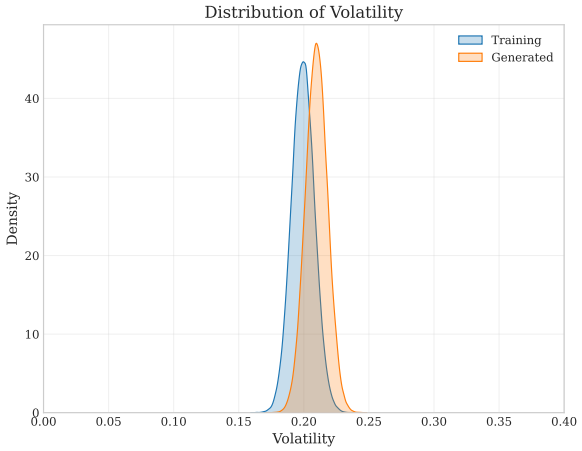
Figure A.1: Drift estimation distributions for the 2-asset CGBM dataset.



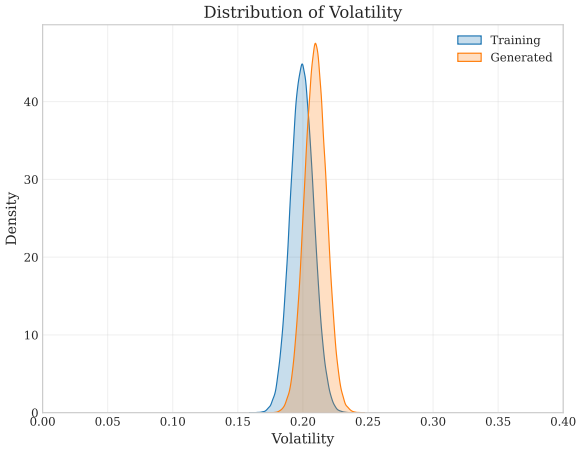
(a) EDM – Asset 1 (b) EDM – Asset 2 (c) EDM – Asset 3

(d) Ambient – Asset 1 (e) Ambient – Asset 2 (f) Ambient – Asset 3

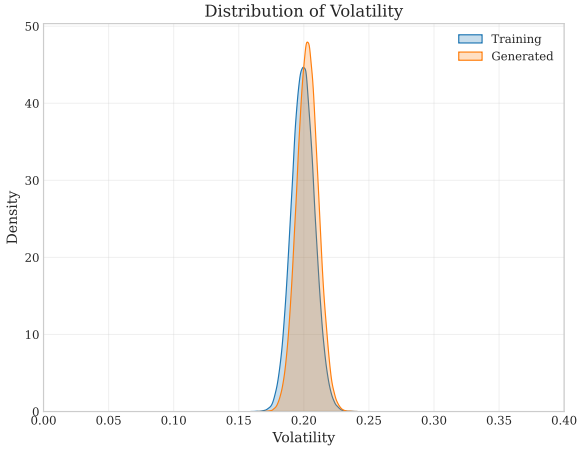
Figure A.2: Drift estimation distributions for the 3-asset CGBM dataset.



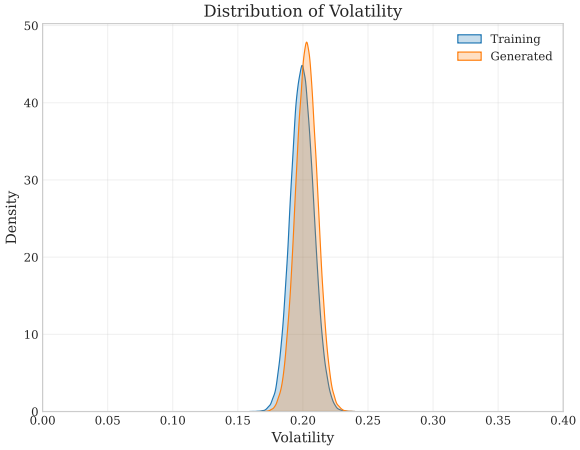
(a) EDM – Asset 1



(b) EDM – Asset 2

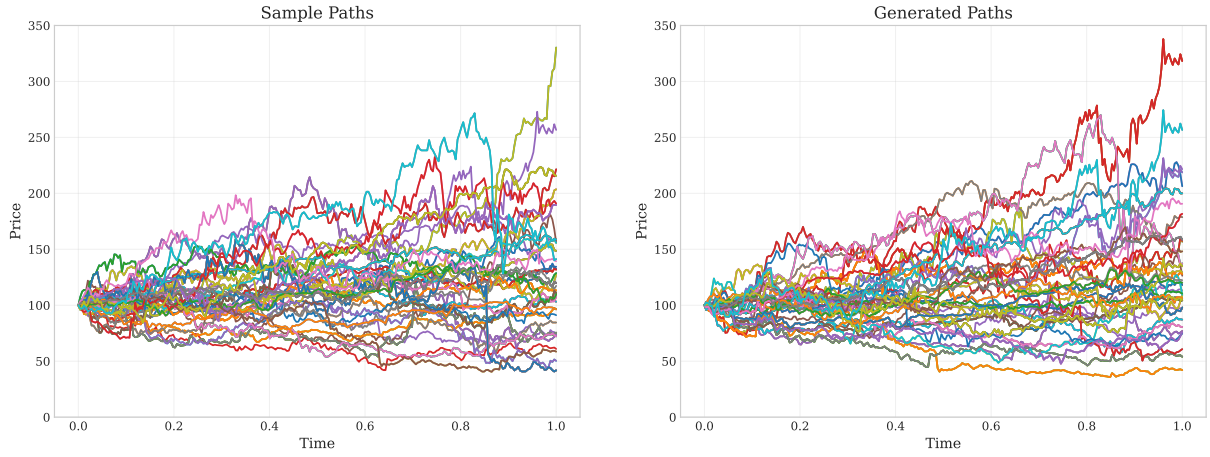


(c) Ambient – Asset 1



(d) Ambient – Asset 2

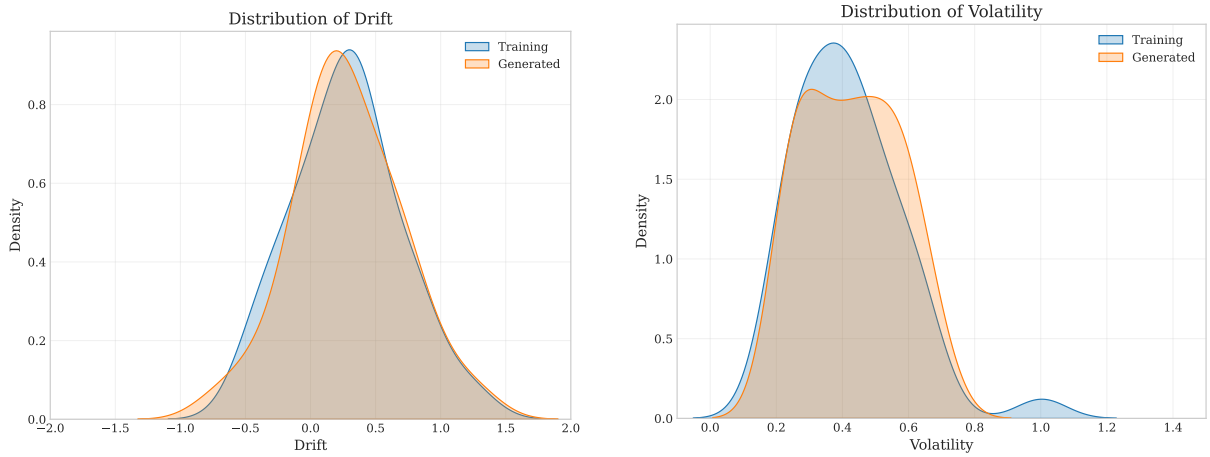
Figure A.3: Volatility estimation distributions for the 2-asset CGBM dataset.



(a) Training Paths (AAPL)

(b) Generated Paths (Ambient)

Figure A.4: Sample paths of prices for AAPL. Left: original dataset. Right: paths generated by Ambient model.



(a) Drift Estimate

(b) Volatility Estimate

Figure A.5: Estimated drift and volatility distributions for AAPL using the Ambient model.

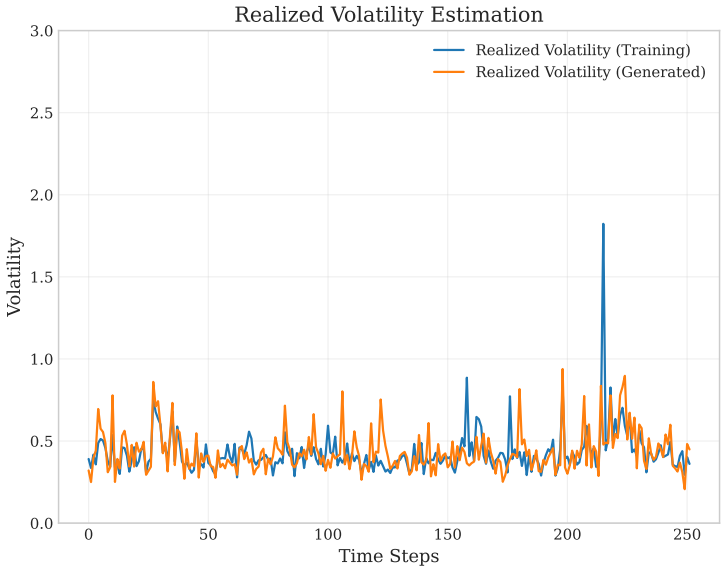
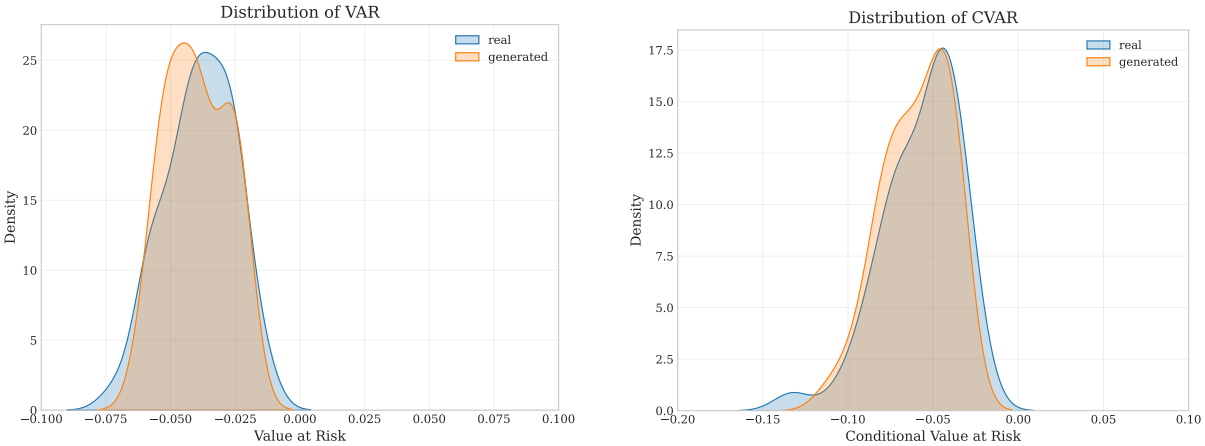


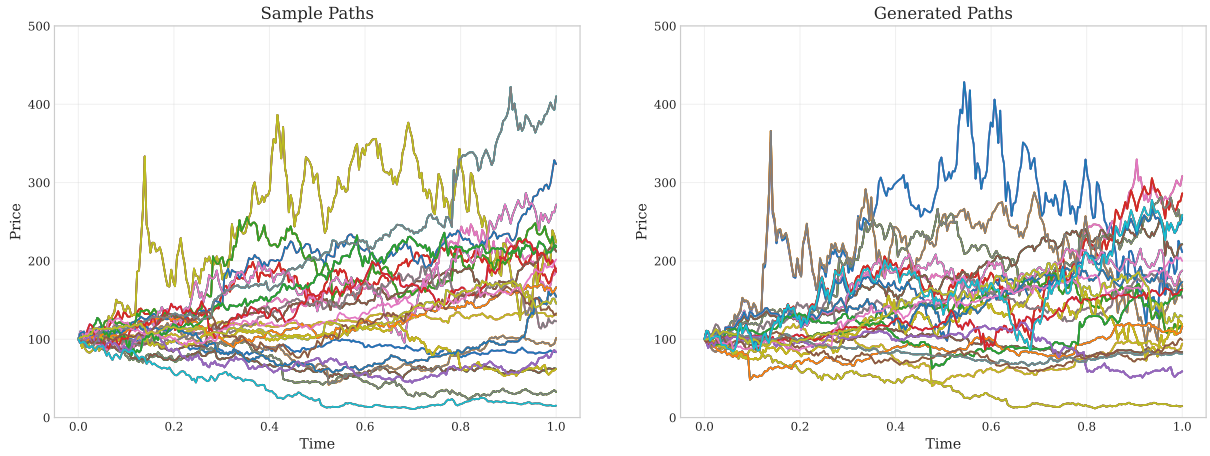
Figure A.6: Estimated realized volatility for AAPL: generated vs. training data.



(a) Value at Risk (VaR)

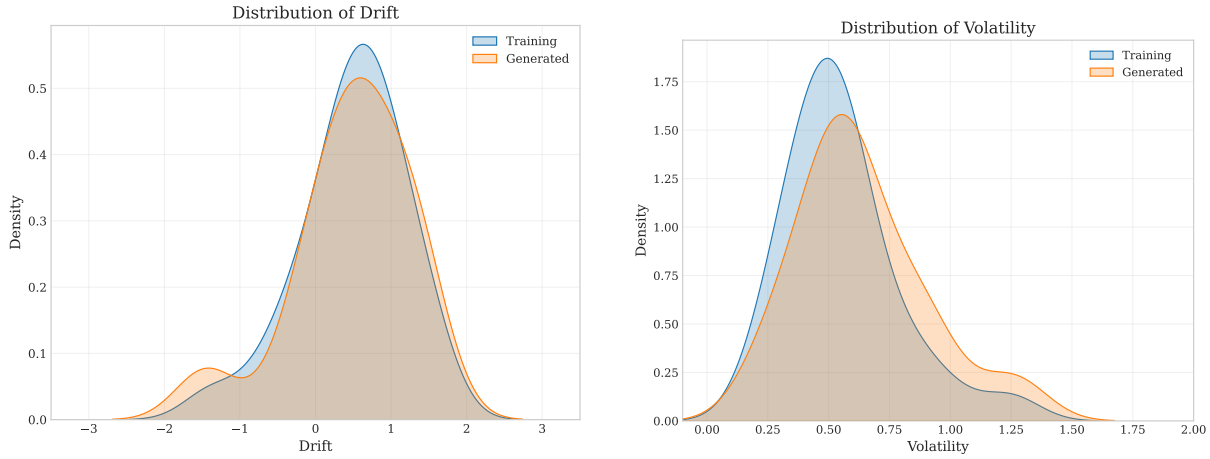
(b) Conditional Value at Risk (CVaR)

Figure A.7: Risk estimates for AAPL: VaR and CVaR distributions under the Ambient model.



(a) Training Paths (NVDA) (b) Generated Paths (Ambient)

Figure A.8: Sample paths of prices for NVDA. Left: original dataset. Right: paths generated by Ambient model.



(a) Drift Estimate (b) Volatility Estimate

Figure A.9: Estimated drift and volatility distributions for NVDA using the Ambient model.

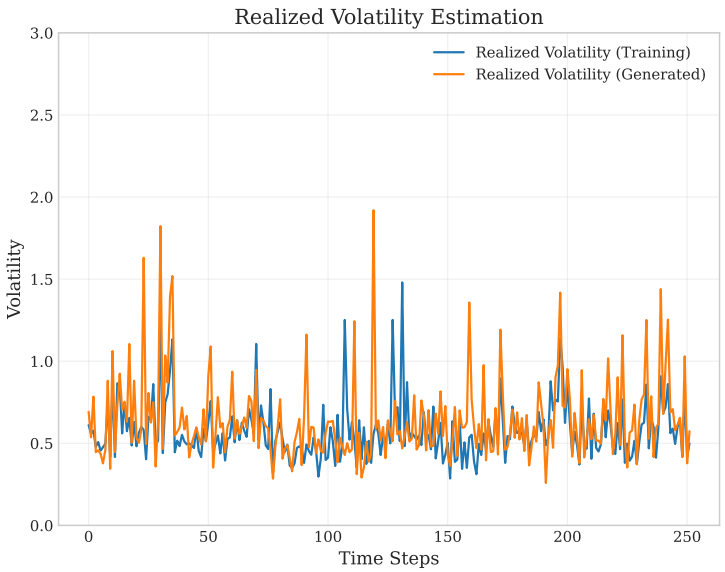
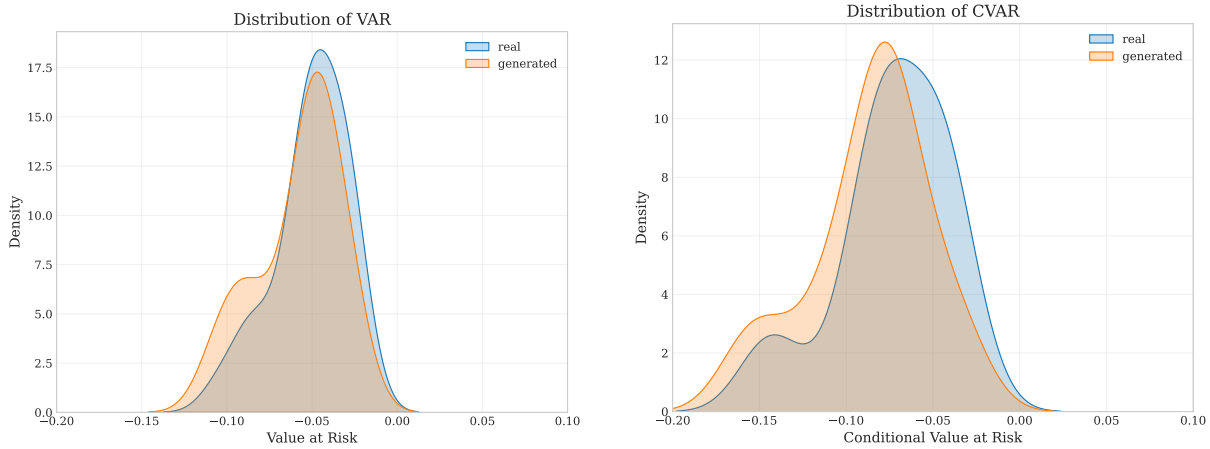


Figure A.10: Estimated realized volatility for NVDA: generated vs. training data.



(a) Value at Risk (VaR)

(b) Conditional Value at Risk (CVaR)

Figure A.11: Risk estimates for NVDA: VaR and CVaR distributions under the Ambient model.