Extending Mamba with Stochastic Dynamics: A Probabilistic State-Space Approach to Financial Time Series Forecasting

Candidate Number: NZQP0 [1]

MSc Machine Learning

Supervisors: Prof Philip Treleaven, UCL
Dr. Omer Gunes, Oxtractor

September 2025

[1]**Disclaimer:** This report is submitted as part requirement for the MSc Degree in Machine Learning. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged. The code used in this project can be found at `https://github.com/micyiucm/Prob-Mamba`

Abstract

Selective state-space models, such as **Mamba**, have emerged as formidable alternatives to Transformers due to their capability of capturing long-range dependencies with linear computational complexity. This is achieved by simulating latent states via an input-dependent ordinary differential equation (ODE) and utilising a hardware-aware kernel fusion. We ask whether this paradigm can be made probabilistic without sacrificing computational efficiency. This project introduces a novel framework that extends the deterministic dynamics of Mamba into a probabilistic model, which we call **Prob-Mamba**. We augment Mamba's underlying input-dependent ODE with a Gaussian diffusion term, transforming its dynamics into an input-dependent Stochastic Differential Equation (SDE). Under zero-order-hold (ZOH), a discretisation rule consistent with the original Mamba implementation, the continuous-time SDE corresponds precisely to a discrete-time, time-varying Linear Gaussian State Space Model (LGSSM), which allows exact inference via Kalman filtering. Prob-Mamba thus bridges a state-of-the-art deep learning architecture with a cornerstone of Bayesian filtering theory, offering a principled and tractable framework for uncertainty-aware time series modeling.

We evaluate on two daily equity indices (NYSE Composite, NASDAQ Composite) and 5-minute Bitcoin returns, against ARIMA+GARCH, vanilla Mamba, and a simple RNN. On equities, Prob-Mamba achieves stronger prediction accuracy volatility scoring over econometric baseline ARIMA+GARCH as well as vanilla Mamba, while the RNN remains the best deterministic forecaster. On high-frequency Bitcoin data, vanilla Mamba attains the strongest deterministic RMSE; a partial Prob-Mamba run suggests promise but was curtailed for compute.

The main trade-off is computational: the probabilistic head is two to three orders of magnitude slower than deterministic baselines due to backpropagation through per-step Kalman updates with input-dependent LGSSM parameters. Overall, results show that principled uncertainty can be effectively integrated into Mamba-style models, yielding better mean accuracy than vanilla Mamba and improved volatility scoring on NYSE, with clear avenues (approximate inference and custom kernels) to make the approach practical at scale.

Acknowledgements

I would like to thank Dr. Omer Gunes at Oxtractor for supervising this project and for providing the datasets used in the experiments. I am also grateful to Prof. Philip Treleaven, my internal supervisor at UCL, for his guidance and feedback on the structure and layout of the thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The field of sequence modelling has largely been dominated by the Transformer architecture ([35]), which has demonstrated remarkable success across domains, from natural language processing (e.g. Generative Pre-Trained Transformers, [30]) to image generation (e.g. Diffusion Transformers, [27]). Its success is primarily attributed to the self-attention mechanism ([35]), which captures complex, long-range dependencies. However, this mechanism scales quadratically with sequence length L ([14]), creating a practical barrier for long sequences.

In response to this limitation, Structured State Space Models (S4 [10]), inspired by classical State Space Models (SSMs), introduced architectural constraints that enable sub-quadratic computation. This innovation unlocked their potential for large-scale sequence modeling, making them a highly efficient and effective alternative to Transformers. Furthering this work, Mamba ([9]), another modern SSM based on S4, integrated a selective parameterisation and a hardware-aware scan. This design allows it to match the expressiveness of transformers based on the self-attention mechanism while scaling in $O(L)$ time and memory.

While powerful, Mamba — like many modern deep learning models — is deterministic and is largely a black box: It fits the data well but provides little insight into the underlying dynamics and offers no principled quantification of its uncertainty, making its internal workings difficult to interpret. This limitation is often acceptable in fields like natural language processing or speech recognition, where the primary goal is to generate a single, coherent output, such as a translated sentence. However, in risk-sensitive fields where understanding and quantifying uncertainty is a central requirement for reliable decision-making, this approach is insufficient. By contrast, probabilistic models explicitly quantify uncertainty, providing a principled way to capture the randomness inherent in data. Unlike purely black-box methods, they can offer greater transparency by modelling not only expected trajectories but also the variability (or ideally full predictive distributions) around them.

This project develops a probabilistic counterpart of Mamba, referred to Prob-Mamba. We augment the continuous-time Mamba state equation with input-dependent diffusion and add heteroskedastic observation noise, turning the state equation into an input-dependent stochastic differential equation (SDE). We then show that under zero-order hold (ZOH), the same discretisation scheme used in Mamba, the model reduces to time-varying Linear Gaussian State Space Model (LGSSM) whose parameters change with the input via Mamba's selection maps. This yields a tractable, linear-time architecture in sequence length while delivering full predictive distributions and enabling exact Kalman filtering. This explicitly quantifies the model's own uncertainty, offering a principled way to capture the randomness inherent in financial data as well as enhancing the model's transparency by revealing its predictive confidence via the variance.

We aim to answer the following questions:

1. What are the mathematical and modelling implications of introducing input-dependent diffusion into the Mamba state dynamics?

2. How does Probabilistic Mamba perform compared to vanilla Mamba, econometric baselines (e.g., ARIMA, GARCH), and deep learning models, both in terms of point-forecast accuracy and the quality of its predictive uncertainty?

3. What is the computational cost of injecting stochastic dynamics directly into Mamba's state equation?

The main contribution of this project is to show that it is both possible and mathematically well-founded to extend Mamba with stochastic dynamics by formulating the latent state equation as an input-dependent SDE. We prove that under mild conditions this SDE admits a unique strong solution, and that it can be discretised into a time-varying LGSSM, enabling exact Kalman filtering. A second contribution, however, is more cautionary: although principled, the computational cost of per-step Kalman updates makes Prob-Mamba one to two orders of magnitude slower than deterministic Mamba in practice, undermining its efficiency. Finally, we provide an empirical evaluation against econometric and deep learning baselines, which highlights both the advantages of principled uncertainty quantification and the severe computational trade-off of this approach.

Thesis Structure

- **Chapter 2** reviews background on time-series modelling, State Space Models/Mamba, and stochastic differential equations.

- **Chapter 3** gives a concise overview on previous work related to the project.

- **Chapter 4** introduces Probabilistic Mamba, proves existence and uniqueness of solutions, derives the ZOH-discretised LGSSM and gives closed form inference.

- **Chapter 5** formulates the financial time-series forecasting task, describes the inference pipeline, and details dataset cleaning and pre-processing pipeline.

- **Chapter 6** outlines the experimental setup, baseline models, and evaluation metrics.

- **Chapter 7** discusses experimental results, limitations, and directions for future work.

Chapter 2

Background

This chapter presents background information on the key concepts for this project. The first part of the chapter starts by introducing terminology used in time series analysis as well as two popular models – ARIMA and GARCH. The chapter then looks specifically into Structured State Space Models as well as Mamba. The chapter ends with explaining and discussing Stochastic Differential Equations, a central modelling tool used in this project.

2.1 Classical Time Series Models

Time series models, originated from statistics and econometrics, aim to describe dependencies typically found in data collected serially in time. They are fundamental in financial modelling due to the sequential nature of financial data.

We assume time series values $y_1, \dots, y_T$ are realisations (observed data) of random variables $Y_1, \dots, Y_T$, which are part of a larger stochastic process $\{Y_t : t \in \mathbb{Z}\}$. We start by introducing some basic notions and terminology, all taken from [8]:

Definition 2.1.1 (Mean and covariance functions)**.** Consider a square-integrable stochastic process $(X_t)_{t\in\mathbb{Z}}$, i.e. $\mathbb{E}(|X_t^2|) < \infty$ for all $t \in \mathbb{Z}$. We define its **mean** and **covariance functions** as

$$\mu(t) := \mathbb{E}(X_t), \quad \gamma(s,t) := \mathrm{Cov}(X_s, X_t).$$

In time series analysis, we usually only observe a single realisation of the process, and statistical inference is only possible if certain properties remain stable over time. This is because without such invariance, past data carry little information about the future, making statistical inference meaningless. The property that formalises this stability is *stationarity*:

Definition 2.1.2 (Strict stationarity)**.** A stochastic process $(X_t)_{t\in\mathbb{Z}}$ is said to be **strictly stationary** if

$$(X_{t_1}, \dots, X_{t_n}) \overset{d}{=} (X_{t_1+k}, \dots, X_{t_n+k}),$$

i.e. $(X_{t_1}, \dots, X_{t_n})$ and $(X_{t_1+k}, \dots, X_{t_n+k})$ have the same joint distribution for any $t_1, \dots, t_n \in \mathbb{Z}, k \in \mathbb{Z}$ and $n \in \mathbb{N}$.

A less-restrictive notion of stationarity is weak stationarity:

Definition 2.1.3 (Weak stationarity)**.** A stochastic process $(X_t)_{t\in\mathbb{Z}}$ is said to be **weakly stationary** if it is square-integrable and

$$\mu(t) = \mu(t+k), \quad \gamma(s,t) = \gamma(s+k, t+k)$$

for any $s, t \in \mathbb{Z}$ and $k \in \mathbb{Z}$.

For any square-integrable process, strict stationarity implies weak stationarity, as the mean and covariance are entirely derived from the distribution. But the converse is only true if the process is Gaussian, since Gaussian processes are completely characterised by its mean and covariance functions.

In practice, we mainly focus on time series that are weakly stationary, since it is a more relaxed and practical assumption: Verifying that a time series would require one to prove that the joint distribution remains unchanged over time, which requires understanding of the underlying data-generating process. Weak stationarity, on the other hand, only requires checking the first two moments of the distribution. These can be reasonably estimated and tested from a single observed time series, making it a more practical condition to work with.

For a weakly stationary process $(X_t)_{t\in\mathbb{Z}}$, note that

$$\begin{aligned}\gamma(s,t) &= \gamma(s-t, t-t) \\ &= \gamma(s-t, 0)\end{aligned}$$

and

$$\begin{aligned}\gamma(s,t) &= \gamma(t,s) \quad \text{(by symmetry of covariance)} \\ &= \gamma(t-s, s-s) \\ &= \gamma(t-s, 0)\end{aligned}$$

which gives $\gamma(s,t) = \gamma(|s-t|, 0)$. This can be reduced to a one-parameter function $\gamma(k) = \gamma(k, 0) = \mathrm{Cov}(X_k, X_0)$, which we call the **autocovariance function**.

Definition 2.1.4 (Autocorrelation function)**.** For a weakly stationary process $(X_t)_{t\in\mathbb{Z}}$, we define the **autocorrelation function** (ACF)

$$\rho(k) := \frac{\gamma(k)}{\gamma(0)} = \mathrm{Cor}(X_k, X_0), \quad k \in \{0, 1 \ldots, \}$$

Definition 2.1.5 (White noise)**.** A stochastic process $(X_t)_{t\in\mathbb{Z}}$ is a **white noise** if it is weakly stationary, has mean $\mu(t) = 0$ and $\gamma(0) = \sigma^2$. We denote white noise by $\mathrm{WN}(0, \sigma^2)$.

Before moving to examples of classical time series models, we introduce the **differencing operator** ∇ and **lag operator** L for notational purposes. They are defined as

$$\nabla Y_t := Y_t - Y_{t-1}, \quad LY_t := Y_{t-1}.$$

2.1.1 Autoregressive Integrated Moving Average Models (ARIMA)

One of the most widely used classical time series models is the Autoregressive Integrated Moving Average (ARIMA), introduced by statisticians George Box and Gwilym Jenkins ([3]). ARIMA models are designed to capture linear dependencies in stationary or differenced time series data, making them a standard baseline for short-term forecasting of non-stationary financial and economic series, such as GDP or returns. We first introduce the building blocks of ARIMA, autoregressive (AR) models and moving average (MA) models.

Definition 2.1.6 (Autoregressive Model, AR(p))**.** A discrete-time stochastic process $(Y_t)_{t\in\mathbb{Z}}$ is said to be an **autoregressive model** with order p, denoted by AR(p), if it satisfies

$$Y_t = \alpha_1 Y_{t-1} + \alpha_2 Y_{t-2} + \ldots + \alpha_p Y_{t-p} + \varepsilon_t,$$

where (ε_t) is white noise.

In an AR(p) model, the current value is a weighted sum of its own past observations, where each coefficient α_i measures the influence of the value i time steps ago.

Definition 2.1.7 (Moving Average Model, MA(q))**.** A discrete-time stochastic process $(Y_t)_{t\in\mathbb{Z}}$ is said to be a **moving average model** with order q, denoted by MA(q), if it satisfies

$$Y_t = c + \beta_0\varepsilon_t + \beta_1\varepsilon_{t-1} + \ldots + \beta_q\varepsilon_{t-q},$$

where $\varepsilon_t \sim \mathcal{N}(0, \sigma_\varepsilon^2)$ and $c, \beta_0, \ldots, \beta_q$ are constant parameters.

In an MA(q) model, the current value is a weighted sum of the current and past shock terms, where each coefficient β_j captures how a shock from j time steps ago propagates into the present.

Definition 2.1.8 (Autoregressive Moving Average models, ARMA(p, q))**.** An **autoregressive moving average** model, denoted by ARMA(p, q), combines both AR(p) and MA(q):

$$Y_t = \alpha_0 + \sum_{i=1}^{p} \alpha_i Y_{t-i} + \sum_{j=1}^{q} \beta_j \varepsilon_{t-j} + \varepsilon_t.$$

Alternatively, the ARMA(p, q) model can be compactly written as

$$\phi(L)\,(Y_t - \mu) \;=\; \theta(L)\,\varepsilon_t,$$

where the autoregressive and moving-average polynomials are

$$\phi(L) = 1 - \phi_1 L - \phi_2 L^2 - \cdots - \phi_p L^p, \quad \theta(L) = 1 + \theta_1 L + \cdots + \theta_q L^q.$$

An ARMA(p, q) model allows both persistence from past values (AR part) and short-term shock effects (MA part), producing flexible autocorrelation patterns up to $\max(p, q)$ prior periods.

Definition 2.1.9. Let $\phi(L) = 1 - \alpha_1 L - \alpha_2 L^2 - \cdots \alpha_p L^p$. An **autoregressive integrated moving average** process, denoted by ARIMA(p, d, q), satisfies

$$\phi(L)\,\nabla^d (Y_t - \mu) \;=\; \theta(L)\,\varepsilon_t, \qquad \varepsilon_t \overset{\text{i.i.d.}}{\sim} (0, \sigma_\varepsilon^2).$$

An ARIMA(p, d, q) is simply obtained by applying the differencing operator $\nabla^d = (1 - L)^d$ (applying ∇ d-times) to the original series to remove non-stationarity, and the resulting differenced series $\nabla^d Y_t$ is then modelled as an ARMA(p, q).

In practice, ARIMA models are fitted using methods like maximum likelihood estimation after identifying p, d, q via Akaike Information Criterion (AIC) or Bayesian Information Criterion (BIC). They assume linear dependence on past values and homoskedasticity (constant variance).

2.1.2 Generalised Autoregressive Conditional Heteroskedasticity Model (GARCH)

Financial time series often exhibit volatility clustering([5]) – periods of high volatility followed by periods of low volatility. This violates the homoskedasticity assumption of ARIMA models. The Autoregressive Conditional Heteroskedasticity (ARCH) model introduced by Engle ([7]) and its generalisation, the Generalised ARCH (GARCH) model of Bollerslev ([2]), address this by modelling time-varying variance conditional on past observations and shocks.

Definition 2.1.10 (Autoregressive Conditional Heteroskedasticity (ARCH) model)**.** An **autoregressive conditional heteroskedasticity** model of order q, denoted ARCH(q), specifies the conditional variance as

$$Y_t = \sigma_t \varepsilon_t, \qquad \sigma_t^2 = \alpha_0 + \alpha_1 Y_{t-1}^2 + \cdots + \alpha_q Y_{t-q}^2,$$

where $\varepsilon_t \sim \mathrm{WN}(0, 1)$, $\alpha_0 > 0$, and $\alpha_1, \ldots, \alpha_q \geq 0$.

Intuitively, in an ARCH(q) model, today's volatility depends on the size of past squared returns – large shocks in the past make today's returns more volatile.

Definition 2.1.11 (Generalised Autoregressive Conditional Heteroskedasticity (GARCH) model)**.** A **generalised autoregressive conditional heteroskedasticity** model of order (p, q), denoted GARCH(p, q), extends ARCH by incorporating lagged conditional variances:

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{q} \alpha_i Y_{t-i}^2 + \sum_{j=1}^{p} \beta_j \sigma_{t-j}^2,$$

with $\alpha_0 > 0$, $\alpha_i \geq 0$, $\beta_j \geq 0$, and $\sum_{i=1}^{q} \alpha_i + \sum_{j=1}^{p} \beta_j < 1$ for weak stationarity.

Intuitively, a GARCH(p, q) model states that today's volatility depends both on recent shocks and on past volatility itself, so periods of high volatility tend to carry over into the near future. The coefficients α_i capture the immediate impact of past squared shocks, while the β_j capture the persistence of volatility from previous periods. This ability to model heteroskedastic, time-varying volatility makes GARCH well-suited for describing heteroskedastic, time-varying asset returns (e.g. for value-at-Risk and volatility forecasting).

In empirical finance, the GARCH$(1, 1)$ specification is the dominant choice among GARCH-type models because it balances parsimony (explaining data well with as few parameters as possible) and explanatory power. With only one ARCH coefficient α (measuring the immediate impact of past squared shocks) and one GARCH coefficient β (capturing the persistence of conditional variance), it reproduces the volatility clustering typically observed in daily and intraday return series. The model is straightforward to estimate by quasi-maximum likelihood, numerically stable under mild conditions, and its parameters admit a clear interpretation in terms of short-run shock impact and long-run persistence ([6]).

2.2 State Space Models and Mamba

State Space Models (SSMs), originated from control theory ([24]), are fundamentally defined as mathematical models of continuous-time dynamical systems. They describe the evolution of latent states in response to an input via first-order differential equations or difference equations. A specific type of State Space Models of interest in this project is the **Structured State Space Models** (S4) ([10]), a sequence model designed to model long sequences both accurately and efficiently. This particular SSM describes the evolution of a latent state vector $h(t) \in \mathbb{R}^n$ in response to a one-dimensional input $x(t)$. The one-dimensional output $y(t)$ is then defined to be a linear projection of $h(t)$. Mathematically, S4 is defined by 4 parameters Δ, A, B, C and consists of a pair of equations:

- **State equation**:$\dfrac{\mathrm{d}h(t)}{\mathrm{d}t} = Ah(t) + Bx(t)$;
- **Output equation**: $y(t) = Ch(t)$,

where $A \in \mathbb{R}^{n\times n}$ is the state transition matrix governing the internal dynamics, $B \in \mathbb{R}^{n\times 1}$ is the input matrix that controls how the input signal $x(t)$ influences the latent state, and $C \in \mathbb{R}^{1\times n}$ is the output matrix. Such SSMs are named as *structured* because computing them efficiently also requires imposing structure on A, with a diagonal matrix with negative entries being the most popular choice. For practical computation with discrete data sequences, one discretise this first order ODE into a first order difference equation of the form

$$h_t = \bar{A}h_{t-1} + \bar{B}x_t, \quad h_0 = 0$$

with discretization step size $\Delta > 0$, and the corresponding output equation $y_t = Ch_t$. This is a linear recurrence, but S4 models can also be interpreted as a global convolution:

$$\begin{aligned}
y_t &= Ch_t \\
&= C(\bar{A}h_{t-1} + \bar{B}x_t) \\
&= C(\bar{A}((\bar{A}h_{t-2} + \bar{B}x_{t-1})) + \bar{B}x_t) \\
&= C(\bar{A}^2 h_{t-2} + \bar{A}\bar{B}x_{t-1} + \bar{B}x_t) \\
&\vdots \\
&= C\left(\sum_{j=0}^{t} \bar{A}^j \bar{B} x_{t-j}\right) \\
y &= x * \bar{K}
\end{aligned}$$

where the convolution kernel $\bar{K} = (C\bar{B}, C\bar{A}\bar{B}, C\bar{A}^2\bar{B}, \ldots)$ and $*$ is the discrete convolution operator. This merges ideas from control theory, recurrent neural networks and convolutional neural networks into one unified framework.

The original S4 model uses the bilinear (Tustin) transform, motivated by stability considerations in HiPPO (High-order Polynomial Projection Operator)-based state matrices. Later architectures, such as Mamba, adopt the zero-order hold (ZOH) rule, which yields simpler and more efficient closed-form updates when A is diagonal. We first define the notion of a matrix exponential:

Definition 2.2.1 (Matrix exponential)**.** For $A \in \mathbb{R}^{n\times n}$ and $t \in \mathbb{R}$, the **matrix exponential** is defined as

$$\exp(tA) := \sum_{k=0}^{\infty} \frac{t^k A^k}{k!}.$$

The matrix exponential satisfies the following:

(i) $\exp(tA)\exp(sA) = \exp((t+s)A)$;

(ii) $\dfrac{\mathrm{d}}{\mathrm{d}t}\exp(tA) = A\exp(tA)$.

Proposition 2.2.1 (ZOH discretisation)**.** The zero-order hold discretisation of the S4 state equation assumes the input $x(s)$ is held constant over the sampling interval $[t, t+\Delta)$, i.e. $x(s) = x_t$ for $s \in [t, t+\Delta)$. This yields

$$\bar{A} = \exp(\Delta A), \quad \bar{B} = \left(\int_0^{\Delta} \exp(\tau A)\,\mathrm{d}\tau\right) B.$$

(i) If A is diagonal, i.e. $A = \mathrm{diag}(a_1, \dots, a_n)$, then letting $B = (b_1, \dots, b_n)^T$ gives

$$(\bar{A})_{ii} = \exp(a_i\Delta), \quad \bar{B}_i = \begin{cases} \frac{\exp(a_i\Delta)-1}{a_i} & \text{if } a_i \neq 0; \\ \Delta & \text{if } a_i = 0. \end{cases}$$

(ii) If A is invertible, then

$$\bar{B} = (\Delta A)^{-1}(\exp(\Delta A) - I)\Delta B$$

A proof can be found in Chapter 8.1.

ZOH assumes that $x(t)$ is constant within each discretisation interval $[t, t+\Delta)$, reflecting the fact that the model only observes inputs at discrete sampling instants and applies them for the duration of the step. This assumption matches the way inputs are processed in neural networks, where each time step or sample is applied for one time step.

In practice, the implementation uses batched, multi-dimensional inputs $x \in \mathbb{R}^{B\times L\times D}$, where B is the batch size, L is the sequence length and D is the feature dimension per time step. The scalar formulation above is for notational simplicity – the extension to the D-dimensional case is obtained by applying the same recurrence independently per channel $d = 1, \dots, D$, as shown in Algorithm 1.

One fundamental property of S4 models is that they are **linear time-invariant**: the governing differential, difference and output equations are linear, and the model parameters $\Delta, A, B, C, \bar{A}, \bar{B}$

Algorithm 1 Structured State Space Models (S4)

```
1:  Input: $x \in \mathbb{R}^{B \times L \times D}$
2:  For each $b, t$, let $x_{b,t,d} \in \mathbb{R}$ be the $d$-th channel value.
3:  Parameters: $A \in \mathbb{R}^{D \times N \times N}$, $B \in \mathbb{R}^{D \times N}, C \in \mathbb{R}^{D \times N}, \Delta \in \mathbb{R}^{D}$
4:  for each channel $d = 1, \dots, D$ do
5:      $\bar{A}_d = \mathrm{ZOH}(\Delta_d, A_d)$
6:      $\bar{B}_d = \mathrm{ZOH}(\Delta_d, B_d)$
7:  end for
8:  Initialise hidden state $h_{0,d}^{(b)} \leftarrow 0 \in \mathbb{R}^N$ for all batched $b = 1, \dots, B$ and channels $d = 1, \dots, D$
9:  for each time step $t = 1, \dots, L$, each batch $b = 1, \dots, B$ and each channel $d = 1, \dots, D$ do
10:     Update hidden state
            $h_{t,d}^{(b)} \leftarrow \bar{A}_d h_{t-1,d}^{(b)} + \bar{B}_d x_{b,t,d}$
11:     Update output equation
            $y_{b,t,d} \leftarrow C_d^T h_{t,d}^{(b)}$
12: end for
13: Return: $y \in \mathbb{R}^{B \times L \times D}$
```

are constant. This means the model applies the same dynamics across the entire input sequence, limiting its ability to adapt to changing contexts within the data. The Mamba architecture overcomes this limitation by making the parameters of the SSM input-dependent. This allows the model to selectively propagate or forget information along the sequence length dimension depending on the current time step. Mathematically, the system parametrises B, C and Δ as functions of the input $x(t)$, while A remaining constant:

$$\frac{\mathrm{d}h(t)}{\mathrm{d}t} = Ah(t) + B(x(t))x(t)$$
$$y(t) = C(x(t))h(t)$$

This parametrisation makes the system linear time-varying: the state dynamics remain linear in $h(t)$, but $B(x(t))x(t)$ is a non-linear function of the input $x(t)$. Hence, the overall model is linear in the hidden state but exhibits non-linear input dependence. The ZOH discretisation for Mamba follows exactly the same argument as in Proposition 2.2.1 for S4 models. The core idea is unchanged: under ZOH, the input $x(t)$ is held constant on the sampling interval $[t_k, t_{k+1})$, so the input-dependent maps become constant on that interval: $B(x(t)) = B(x_{t_k})$ and $C(x(t)) = C(x_{t_k})$. The key difference from S4 is that the selection step $\Delta(x_t)$ is input-dependent and distinct from the fixed data step $\Delta t = t_{k+1} - t_k$. Intuitively, with diagonal A having negative entries, a large $\Delta(x_t)$ strongly contracts the state ($\bar{A}_k \to 0$) and emphasises the current input ($\bar{B}_k \to -A^{-1}B(x_k)$), while a small $\Delta(x_t)$ largely preserves the state ($\bar{A}_k \approx I$) and adds only a small input contribution ($\bar{B}_k \approx \Delta_k B(x_k)$). This yields the discrete-time recurrence, with matrices recomputed at every step:

$$\bar{A}_k = \exp(\Delta_k A), \quad \bar{B}_k = \left(\int_0^{\Delta_k} e^{A\tau} \, d\tau \right) B(x_k), \quad \Delta_k := \Delta(x_{t_k}).$$

In practice, the maps chosen for the input-dependent parameters of Mamba are as follows:

$$\begin{aligned}
B(x) &= W_B x + b_B \\
C(x) &= W_C x + b_C \\
\Delta(x) &= \text{softplus}(w_\Delta^T x + b_\Delta) = \log(1 + \exp(w_\Delta^T x + b_\Delta))
\end{aligned}$$

where $W_B, W_C \in \mathbb{R}^{N \times D}$ are learnable projection matrices, $w_\Delta \in \mathbb{R}^D$ is a learnable weight vector, $b_B, b_C \in \mathbb{R}^N$ are learnable bias vectors and $b_\Delta \in \mathbb{R}$ is a learnable bias. These selection maps are not arbitrary. By construction, the step size $\Delta(x_t)$ induces an update gate $z_{t,i} = 1 - \exp(\Delta(x_t)a_i)$ for diagonal $A = \text{diag}(a_i)$, analogous to the gating in gated recurrent units (GRUs). Theorem 1 in [9] makes this connection explicit in the one-dimensional case, showing that the selective SSM recurrence reduces exactly to a gated RNN update. The maps $s_B(x_t)$ and $s_C(x_t)$ further act as input and output gates respectively, so that Mamba's selection mechanism can be mathematically understood as a structured gating mechanism derived from discretised state-space dynamics.

Mamba initially gained attention for showing that structured state space models could match the expressiveness of deep sequence models while scaling linearly in sequence length. Unlike Transformers, whose self-attention incurs $\mathcal{O}(L^2)$ cost in sequence length, Mamba leverages an efficient hardware-aware scan algorithm to achieve $\mathcal{O}(L)$ time and memory complexity. It also achieved 20 to 40 times faster scans and could process up to five times as many tokens per second as comparably sized Transformers during inference.

Theoretically, both A and $B(x)$ are discritised using the ZOH rule. However, in the actual implementation, Mamba does not utilise full ZOH discretisation – instead it uses ZOH to discretise A while discretising $B(x)$ using a first-order Euler approximation $\bar{B} = \Delta(x)B(x)$ for computational simplicity. Algorithm 2 shows the pseudocode for Mamba.

Algorithm 2 Mamba (Selective State Space Model)

```
1: **Input:** $x \in \mathbb{R}^{B\times L\times D}$
2: For each batch $b = 1, \dots, B$ and time step $t = 1, \dots, L$ let $x_{b,t} \in \mathbb{R}^D$ denote the feature vector
   at that time step and $x_{b,t,d}$ be its $d$-th component.
3: **Parameters:** $A \in \mathbb{R}^{D\times N\times N}$, $W_B, W_C \in \mathbb{R}^{N\times D}, b_B, b_C \in \mathbb{R}^N, w_\Delta \in \mathbb{R}^D, b_\Delta \in \mathbb{R}$
4: **Selection maps (per time step):**
       $B(x) = W_B x + b_B \in \mathbb{R}^N$
       $C(x) = W_C x + b_C \in \mathbb{R}^N$
       $\Delta(x) = \text{softplus}(w_\Delta^T x + b_\Delta) \in \mathbb{R}^+$
5: Initialise $h_{0,d}^{(b)} \leftarrow 0 \in \mathbb{R}^N$ for all batches $b = 1 \dots, B$ and channels $d = 1, \dots, D$
6: **for** $t = 1 \dots, L$ and $b = 1, \dots, B$ **do**
7:     Compute selections:
           $B_{b,t} \leftarrow B(x_{b,t}), \quad C_{b,t} \leftarrow C(x_{b,t}), \quad \Delta_{b,t} \leftarrow \Delta(x_{b,t})$
8:     **for** each channel $d = 1, \dots, D$ **do**
9:         $\bar{A}_{b,t,d} \leftarrow \text{ZOH}(\Delta_{b,t}, A_d)$
10:    **end for**
11:    $\bar{B}_{b,t} \leftarrow \text{Euler}(\Delta_{b,t} B_{b,t})$
12:    **for** each channel $d = 1, \dots, D$ **do**
13:        Update hidden state
               $h_{t,d}^{(b)} \leftarrow \bar{A}_{b,t,d} h_{t-1,d}^{(b)} + \bar{B}_{b,t} x_{b,t,d}$
14:        Update output equation
               $y_{b,t,d} \leftarrow C_{b,t}^T h_{t,d}^{(b)}$
15:    **end for**
16: **end for**
17: **Return:** $y \in \mathbb{R}^{B\times L\times D}$
```

2.3 Stochastic Differential Equations

This section is heavily based on Chapter 5.2 of [13]. For an introduction to measure-theoretic probability, see Chapter 8.3.1.

Let $(X_t)_{t\in[0,T]}$ be an n-dimensional stochastic process and $(W_t)_{t\in[0,T]}$ be a standard m-dimensional Brownian motion defined on a filtered probability space $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\in[0,T]}, \mathbb{P})$, where $(\mathcal{F}_t)_{t\in[0,T]}$ is the augmented natural filtration generated by $(W_t)_{t\in[0,T]}$ and $T > 0$.

Definition 2.3.1 (Stochastic Differential Equations)**.** A **stochastic differential equation** (SDE) is an equation of the form

$$\mathrm{d}X_t = b(t, X_t)\,\mathrm{d}t + \sigma(t, X_t)\,\mathrm{d}W_t, \tag{2.1}$$

where $b : \mathbb{R}^n \times [0,T] \to \mathbb{R}^n$ is the **drift** and $\sigma : \mathbb{R}^n \times [0,T] \to \mathbb{R}^{n\times m}$ is the **diffusion**. Such an equation usually comes with an initial condition $X_0 = x$, $x \in \mathbb{R}^n$ or $X_0 = \xi$, where ξ is a $\mathcal{F}_0$-measurable (i.e. known at time 0) random vector in $\mathbb{R}^n$ with distribution μ.

One can write the above system of SDEs in component-wise form:

$$\mathrm{d}X_t^{(i)} = b_i(t, X_t)\,\mathrm{d}t + \sum_{j=1}^{m} \sigma_{ij}(t, X_t)\,\mathrm{d}W_t^{(j)}, \quad 1 \le i \le n, 1 \le j \le m$$

where

- b_i is the i-th entry of the drift vector b;
- σ_{ij} is the (i,j)-th entry of the diffusion matrix σ; and
- $W_t^{(j)}$ is the j-th entry of the m-dimensional Brownian motion.

Unlike ordinary differential equations, where one simply speaks of a solution, stochastic differential equations admit two distinct notions of solutions: strong and weak.

Definition 2.3.2 (Strong solution)**.** A **strong solution** to the stochastic differential equation

$$\mathrm{d}X_t = b(t, X_t)\,\mathrm{d}t + \sigma(t, X_t)\,\mathrm{d}W_t, \quad X_0 = \xi,$$

is a stochastic process $(X_t)_{t\in[0,T]}$ with continuous sample paths and satisfies the following properties:

(i) $(X_t)_{t\in[0,T]}$ is adapted to the filtration $(\mathcal{F}_t)_{t\in[0,T]}$;

(ii) $\mathbb{P}(X_0 = \xi) = 1$;

(iii) $\mathbb{P}\left[\int_0^t |b_i(s, X_s)| + \sigma_{ij}^2(s, X_s)\,\mathrm{d}s < \infty\right] = 1$ holds for every $1 \le i \le n$,$1 \le j \le m$ and $0 \le t \le T$;

(iv) The integral form of the SDE

$$X_t = X_0 + \int_0^t b(s, X_s)\,\mathrm{d}t + \int_0^t \sigma(s, X_s)\,\mathrm{d}W_s; \quad 0 \le t \le T$$

or equivalently,

$$X_t^{(i)} = X_0^{(i)} + \int_0^t b_i(s, X_s)\,\mathrm{d}s + \sum_{j=1}^{m} \int_0^t \sigma_{ij}(s, X_s)\,\mathrm{d}W_s^{(j)}; \quad 0 \le t \le T, 1 \le i \le n.$$

holds almost surely (i.e. with probability 1).

Definition 2.3.3 (Weak solution)**.** A **weak solution** to the SDE (2.1) consists of

- a continuous, adapted $\mathbb{R}^n$-valued stochastic process $((\tilde{X}_t)_{t\in[0,T]]}$,
- an m-dimensional Brownian motion $(\tilde{W}_t)t \in [0,T])$, and
- a filtered probability space $(\tilde{\Omega}, \tilde{\mathcal{F}}, (\tilde{\mathcal{F}}_t)_{t\in[0,T]}, \tilde{\mathbb{P}})$

such that

(i) $\tilde{X}_0$ has law/distribution μ;

(ii) Conditions (iii) and (iv) in Definition 2.3.2 are satisfied with $(\tilde{X}_t)_{t\in[0,T]]}$ and $(\tilde{W}_t)_{t\in[0,T]}$.

The probability measure μ on $(\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n))$ is called the *initial distribution* of the solution.

In a strong solution, the filtered probability space and Brownian motion are fixed in advance, and one requires an adapted process $(X_t)_{t\in[0,T]}$ that satisfies the SDE almost surely. The initial condition is given by a random variable ξ that is known at time $t = 0$ ($\mathcal{F}_0$-measurable) with prescribed distribution μ. In contrast, a weak solution does not fix the probability space or Brownian motion a priori: instead, one only requires the existence of a filtered probability space $(\tilde{\Omega}, \tilde{\mathcal{F}}, (\tilde{\mathcal{F}}_t)_{t\in[0,T]}, \tilde{\mathbb{P}})$, a Brownian motion $(\tilde{W}_t)_{t\in[0,T]}$, and an $\tilde{\mathcal{F}}_t$-adapted process $(\tilde{X}_t)_{t\in[0,T]}$ such that the SDE holds in integral form and the initial state has law/distribution μ. Intuitively, this distinction can be understood in terms of how much of the randomness is fixed in advance: Given a particular Brownian motion, can you construct a process $(X_t)_{t\in[0,T]}$ adapted to it that solves the SDE? If the answer is yes, then $(X_t)_{t\in[0,T]}$ is a strong solution. In contrast, a weak solution relaxes this requirement: it only asks for some filtered probability space and some Brownian motion on which such a process exists with the correct distribution. Put differently, a strong solution provides the *exact* path a process will take for a given specific source of randomness, while a weak solution only guarantees that a process with the correct distribution exists, without tying it to a particular random source.

Similar to the theory of ODEs, we also have the notion of uniqueness for both strong and weak solutions. We will only define the notion of a unique strong solution since it will be used in this project.

Definition 2.3.4 (Unique strong solution)**.** Let $(X_t)_{t\in[0,T]}$ and $(\tilde{X}_t)_{t\in[0,T]}$ be two strong solutions to the SDE (2.1) with initial condition $X_0 = \xi$. We say that the SDE has a **unique strong solution** if $\mathbb{P}(X_t = \tilde{X}_t \text{ for all } t \in [0,T]) = 1$.

We also define the notion of a Frobenius norm, which is the "matrix equivalent" of Euclidean norm:

Definition 2.3.5 (Frobenius norm)**.** Let $A \in \mathbb{R}^{n \times n}$. Then we define the **Frobenius norm** of A by

$$||A||_F := \left[\sum_{i=1}^{n} \sum_{j=1}^{n} |a_{ij}|^2 \right]^{1/2},$$

where $|| \cdot ||$ is the Euclidean norm for vectors.

The following theorem gives the necessary and sufficient conditions on the drift b : and diffusion $\sigma(t, X_t)$ for an SDE to have a unique strong solution:

Theorem 2.3.1 (Existence and Uniqueness of strong solution)**.** Consider the SDE (2.1). Suppose that the drift b and diffusion σ satisfy the following conditions:

(i) (Global Lipschitz) $||b(t,x) - b(t,y)|| + ||\sigma(t,x) - \sigma(t,y)||_F \leq K||x - y||$;

(ii) (Linear growth) $||b(t,x)||^2 + ||\sigma(t,x)||_F^2 \leq K^2(1 + ||x||^2)$

for every $0 \leq t \leq T$, $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^n$, and $K \in \mathbb{R}^+$. Then there exists a unique continuous, adapted process $(X_t)_{t \geq 0}$ which is a strong solution to the SDE (2.1) with initial condition $X_0 = \xi$ that is independent of W_t and has finite second moment ($\mathbb{E}(|X|^2) < \infty$). Moreover, this process is square-integrable: For every $T > 0$, there exists a constant C that depends only on K and T such that

$$\mathbb{E}(||X_t||^2) \leq C(1 + \mathbb{E}(||\xi||^2))e^{Ct}, \quad 0 \leq t \leq T.$$

A detailed proof of Theorem (2.3.1) can be found in Chapter 5.2.9 of [13].

Chapter 3

Related Work

This chapter reviews related work for this project. It first examines probabilistic extensions of Mamba and selective state-space models, then surveys probabilistic forecasting methods in finance. The chapter also situates our framework within the literature on neural differential equations, reviews applications of Mamba in financial modelling, and concludes with its broader use across AI and machine learning domains.

Probabilistic extensions of Mamba

A growing line of work explores how to endow selective state-space models with uncertainty while preserving their scan-friendly efficiency. KalMamba ([1]) uses Mamba layers to learn the parameters of a latent linear Gaussian state space model (LGSSM) and performs inference via Kalman filtering/smoothing with parallel scans to support efficient, uncertainty-aware reinforcement learning (RL) under partial observability and noisy environments. For time series forecasting, Mamba-ProbTSF ([28]) utilises a dual-network framework based on the Mamba architecture. This approach addresses limitations in traditional Mamba forecasts by proposing a method where one neural network generates point forecasts while another estimates predictive uncertainty by modeling variance. These demonstrate that probabilistic reasoning and scan-parallel SSMs are compatible, yet current approaches either (i) constrain dynamics to linear-Gaussian forms, or (ii) treat uncertainty at the output head. In contrast, this project introduces stochasticity directly into the selective dynamics: by injecting Gaussian diffusion into the Mamba state update, the system discretises into a time-varying LGSSM (see Chapter 4.3) whose parameters depend on the input, allowing uncertainty to propagate temporally through hidden states while retaining nonlinear selective structure.

Probabilistic forecasting methods in finance

Probabilistic forecasting has been increasingly applied to financial returns to capture heteroskedasticity, non-Gaussian features, and regime shifts that point forecasts miss. [19] evaluate neural models that directly parameterise heavy-tailed distributions and show improvements over classical GARCH in value-at-risk and tail risk assessment. [36] study volatility clustering and propose a scale-mixture framework that improves calibration compared to Deep Evidential Regression. These methods refine likelihood design to improve predictive distributions at the output stage. In contrast, our Prob-Mamba introduces stochasticity inside the state dynamics, ensuring that uncertainty is carried through time rather than appended at prediction.

Neural Differential Equations and Stochastic Processes

The Prob-Mamba framework of formulating an SDE with learnt drift and diffusion is situated within the rapidly developing field of neural differential equations. Neural ODEs [4] parametrise the vector field of an ODE with a neural network, thereby learning continuous-time dynamics directly from the data. Our proposed framework can be seen as an extension of this concept into the stochastic domain, also known as Neural SDEs ([17]). Neural SDEs use neural networks to learn the drift and diffusion terms of an SDE. This class of models is exceptionally powerful, capable of representing highly complex, non-linear stochastic dynamics. However, this generality often comes at a steep price. For a general Neural SDE, the transition density is intractable, and inference typically requires sophisticated and computationally intensive approximate methods. Prob-Mamba, on the other hand, balances model expressivity and computational tractability by making specific structural choices to the drift and diffusion terms – the drift term is linear with respect to the latent state and that the diffusion process is driven by Gaussian noise (see Chapter 4.1 for details). This specific structure allows closed-form discretisation into a time-varying LGSSM and enables the use of Kalman filter for inference (see Chapter 4.4).

Mamba in financial modelling

The first dedicated application of selective SSMs to equity prediction is MambaStock ([32]). By leveraging Mamba's input-dependent parameters and scan-parallel updates, the model processes raw historical prices without handcrafted features and learns to selectively propagate informative signals while down-weighting transient fluctuations. Empirically, it improves over traditional econometric and deep learning baselines, establishing that Mamba can be effectively applied for financial time series forecasting. Subsequent studies extend Mamba to hybrid or multimodel settings that fuse prices with exogenous signals: a notable example includes FinMamba [37]) and SAMBA ([18]) which combine Mamba with graph neural networks to capture inter-stock dependencies. A key limitation of these models, which we address in this project, is that they are deterministic and treat forecasting as a point-prediction problem, providing no direct quantification of risk.

Mamba across Artificial Intelligence and Machine Learning domains

Lastly, we present several notable works in Mamba across other domains in Artificial Intelligence (AI) and Machine Learning (ML):

- **Large Language Models:** Recent work suggests that Mamba can be effectively combined with attention mechanisms without sacrificing efficiency. Samba ([31]) presents a simple layer-wise hybrid that integrates Mamba blocks with sliding-window attention, leveraging Mamba's recurrent stats compression for long memory while using attention for precise local context. This design demonstrates that scan-parallel recurrence and attention can be harmonised to achieve both efficiency and capacity.

- **Computer vision:** In computer vision, Mamba has shown that it can be integrated effectively with other modules and to scale deep neural networks. Mambavision ([11]) combines Mamba blocks and Transformer-style attention into a hybrid vision model. This design shows that Mamba modules can be interleaved with attention layers in deep networks while retaining efficiency, yielding competitive performance across classification and transfer tasks such as detection and segmentation. SiMBA ([26]) proposes a stabilized variant of Mamba that improves training robustness when scaling to large vision models. These works highlight Mamba's composability and scalability and motivate its adaptation to other sequential domains.

Chapter 4

Main Theoretical Result

This chapter presents the main theoretical result of the project. We propose to extend the deterministic Mamba dynamics into a probabilistic model by introducing a Gaussian diffusion term to the underlying ordinary differential equation, yielding an input-dependent stochastic differential equation (SDE). We establish that this SDE admits a unique strong solution under global Lipschitz and linear growth conditions. Finally, we show that, under zero-order hold discretisation, the resulting process corresponds to a time-varying Linear Gaussian State Space Model (LGSSM).

4.1 The Probabilistic Mamba (Prob-Mamba)

Let $W = (W_t)_{t\in[0,T]}$ be a standard m-dimensional Brownian motion defined on a filtered probability space $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\in[0,T]}, \mathbb{P})$, where $(\mathcal{F}_t)_{t\in[0,T]}$ is the augmented natural filtration generated by W and $T > 0$. We consider multivariate input data $x_t \in \mathbb{R}^{d_x}$, a latent state $h_t \in \mathbb{R}^n$ and an output $y_t \in \mathbb{R}^{d_y}$, where d_x, d_y and n are the dimensions of input data, output and latent space respectively.

Prob-Mamba is derived by augmenting the Mamba ODE with input-dependent diffusion $\Sigma(x_t)$:

$$\mathrm{d}h_t = (Ah_t + B(x_t)x_t)\,\mathrm{d}t + \Sigma(x_t)\,\mathrm{d}W_t, \quad h_0 = \xi \tag{4.1}$$

and adding observation noise ε_t to the output equation:

$$y_t = C(x_t)h_t + \varepsilon_t,$$

where $B(x_t) \in \mathbb{R}^{n\times d_x}, C(x_t) \in \mathbb{R}^{d_y\times n}, \Sigma(x_t) \in \mathbb{R}^{n\times m}$ and $\Delta(x_t)$ are input-dependent parameters and $A \in \mathbb{R}^{n\times n}$ is a diagonal matrix with negative entries. When it comes to inference, we condition on the observed inputs $x_1, \ldots, x_T$, therefore $B(x_t)$, $C(x_t)$, $\Sigma(x_t)$ and $\Delta(x_t)$ are all deterministic time-dependent coefficients. Hence, the only randomness of the model comes from the Brownian motion W and the observation noise ε_t.

To be faithful to the original Mamba implementation, we choose the same input-selective maps for Prob-Mamba:

$$\begin{aligned} \mathrm{vec}(B(x_t)) &= W_B x_t + b_B; \\ \mathrm{vec}(C(x_t)) &= W_C x_t + b_C; \\ \Delta(x_t) &= \mathrm{softplus}(w_\Delta^T x_t + b_\Delta), \end{aligned}$$

where $W_B \in \mathbb{R}^{nd_x \times d_x}, W_C \in \mathbb{R}^{(d_y n \times d_x)}$ are learnable projection matrices, $b_B \in \mathbb{R}^{nd_x}, b_C \in \mathbb{R}^{d_y n}$ are learnable bias vectors, $w_\Delta \in \mathbb{R}^{d_x}$ is a learnable weight vector and $b_\Delta \in \mathbb{R}$ is a learnable bias. Here vec : $\mathbb{R}^{n \times d_x} \to \mathbb{R}^{nd_x}$ is the vectorisation operator, which stacks the columns of a matrix into a single column vector: For any $A = (a_{ij}) \in \mathbb{R}^{n \times d_x}$,

$$\mathrm{vec}(A) = (a_{1,1}, a_{2,1}, \ldots, a_{n,1}, a_{1,2}, a_{2,2,\ldots}, a_{n,2}, \ldots, a_{1,d_x}, a_{2,d_x}, \ldots, a_{n,d_x})^T.$$

For the diffusion term $\Sigma(x_t)$ which controls conditional covariance of latent states h_t, we choose

$$\Sigma(x_t) = \mathrm{diag}(\mathrm{softplus}(W_\Sigma x_t + b_\Sigma)), \quad W_\Sigma \in \mathbb{R}^{n \times d_x}, b_\Sigma \in \mathbb{R}^n,$$

i.e. an input-dependent diagonal matrix with positive entries enforced by the softplus activation function (since volatilities are non-negative). This implicitly assumes that the random shocks driving each latent state dimension are uncorrelated, which is a simplifying assumption but much more computationally efficient than letting $\Sigma(x_t)$ to be a full matrix.

The observation noise ε_t is modelled to be a mean-zero Gaussian random vector, independent across t and the initial state $h_0 = \xi$:

$$\varepsilon_t \sim \mathcal{N}(0, R(x_t)),$$

where $R(x_t)$ is an input-dependent variance. Again we choose the map

$$R(x_t) = \mathrm{diag}(\mathrm{softplus}(W_R x_t + b_R))$$

where $W_R \in \mathbb{R}^{d_y \times d_x}$ is a learnable weight matrix and $b_R \in \mathbb{R}^{d_y}$ is a learnable bias vector. By letting the variance to be input-dependent, the observation noise is heteroscedastic, i.e. the variance of ε_t is not constant, mirroring the real-life phenomenon that volatility of returns evolves over time. The choice of a Gaussian noise is as follows:

- **Theoretical guarantee**: Aggregated shocks from many small, independent influences are well-approximated by a Gaussian distribution by the Central Limit Theorem. While financial time series often show heavy tails and volatility clustering, we retain conditional Gaussianity with time-varying variances via input-dependent maps $\Sigma(x_t)$ and $R(x_t)$, so the unconditional behaviour can still be heavy-tailed.

- **Computational tractability**: Gaussian distributions are mathematically tractable. They are fully characterised by just their mean and variance, and linear transformations of Gaussian variables are Gaussian. By choosing ε_t to be Gaussian, the discretised probabilistic Mamba gives a closed form solution as we will see in Section 4.3.

- **Standard framework for Stochastic Calculus**: The standard model for continuous-time randomness is Brownian motion, which is defined by its independent and stationary Gaussian increments. By using Gaussian noise, the model is grounded in the well-understood framework of stochastic calculus.

4.2 Existence and uniqueness of strong solution

For Prob-Mamba to be a meaningful forecasting tool, the underlying SDE governing the evolution of the latent state h_t needs to have a strong, unique solution because of the following reasons:

- **Well-posedness**: A strong solution means that for a specific path of the underlying random noise W, there is one and only one possible outcome path for the latent state h_t. This guarantees there is no ambiguity in the model's output: For the same input data and the same random shocks, the model will always produce the exact same latent state trajectory.
- **Consistency with discretisation**: Many discretisation algorithms used in practice, such as Euler-Maruyama or ZOH, rely on approximating the continuous-time SDE by a discrete-time recursion. Strong uniqueness ensures that these algorithms converge to the correct underlying process, rather than drifting toward different candidate solutions. This guarantees the discretised model in Section 4.3 truly represents the same dynamics as the SDE driving the probabilistic Mamba.
- **Learning and reproducibility**: When training the model, we optimise the parameters by backpropagating through simulated or discretised trajectories. With a unique strong solution, these trajectories are reproducible functions of the noise W and inputs $x_1, \dots, x_T$. This makes gradient-based learning stable and ensures that two runs of the model with the same random seed produce identical latent states.

Before proceeding to the proof, we need the following results:

Lemma 4.2.1. For any $A = (a_{ij}) \in \mathbb{R}^{n\times n}$ and $x \in \mathbb{R}^n$, define the **operator norm**:

$$||A||_{\text{op}} := \sup_{x\neq 0} \frac{||Ax||}{||x||}.$$

Then the following holds:

(i) $||Ax|| \leq ||A||_{\text{op}} \cdot ||x||$;

(ii) $||A||_{\text{op}} \leq ||A||_F$.

Lemma 4.2.2. Let $x = (x_1, \dots, x_n)^T \in \mathbb{R}^n$. Then

$$\text{softplus}(x) \leq |x| + (\log 2)\mathbf{1},$$

where $|x| = (|x_1|, \dots, |x_n|)^T$, $\mathbf{1} = (1, \dots, 1) \in \mathbb{R}^n$ and the softplus function is applied elementwise.

Proofs of Lemmas 4.2.1 and 4.2.2 can be found in Chapter 8.1. We now formally prove that the Mamba SDE (4.1) admits a unique strong solution:

Theorem 4.2.1 (Existence and uniqueness of strong solution)**.** Let $W = (W_t)_{t\in[0,T]}$ be an m-dimensional Brownian motion on a filtered probability space $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\in[0,T]}, \mathbb{P})$, where the filtration $(\mathcal{F}_t)_{t\in[0,T]}$ is the augmented natural filtration generated by W. Consider the SDE

$$\mathrm{d}h_t = (Ah_t + B(x_t)x_t)\,\mathrm{d}t + \Sigma(x_t)\,\mathrm{d}W_t, \quad h_0 = \xi,$$

where $A \in \mathbb{R}^{n\times n}$, $(x_t)_{t\in[0,T]}$ is an $(\mathcal{F}_t)$-adapted, bounded input process (i.e. there exists $M < \infty$ such that $\sup_{t\in[0,T]} ||x_t|| < M$) and the input-dependent selective maps are

$$\mathrm{vec}(B(x_t)) = W_b x_t + b_B, \quad \Sigma(x_t) = \mathrm{diag}(\mathrm{softplus}(W_\Sigma x_t + b_\Sigma)).$$

If the initial condition $h_0 = \xi$ is independent of W and $\mathbb{E}[|\xi|^2] < \infty$, then the SDE admits a unique continuous, adapted strong solution $(h_t)_{t\in[0,T]}$. Moreover, the solution is square-integrable and satisfies a moment bound of the form

$$\mathbb{E}[||h_t||^2] \le C(1 + \mathbb{E}[||\xi||^2])e^{Ct}, \quad 0 \le t \le T$$

for some constant C depending only on K and T.

Proof. We prove this theorem by verifying conditions (i) and (ii) in Theorem 2.3.1:

(i) For $h, \tilde{h} \in \mathbb{R}^n$,

$$\begin{aligned} ||b(t,h) - b(t,\tilde{h})|| + ||\sigma(t,h) - \sigma(t,\tilde{h})||_F &= ||Ah + B(x_t)x_t - A\tilde{h} - B(x_t)x_t|| + ||\Sigma(x_t) - \Sigma(x_t)||_F \\ &= ||A(h - \tilde{h})|| \\ &\le ||A||_{\mathrm{op}} \cdot ||h - \tilde{h}|| \quad \text{(by Lemma 4.2.1 (i))} \end{aligned}$$

so choose $K = ||A||_{\mathrm{op}}$ gives the desired result.

(ii)

$$\begin{aligned} ||b(t,h)||^2 + ||\sigma(t,h)||_F^2 &= ||Ah + B(x_t)x_t||^2 + ||\Sigma(x_t)||_F^2 \\ &\le (||Ah|| + ||B(x_t)x_t)||)^2 + ||\Sigma(x_t)||_F^2 \quad \text{(by Triangle inequality)} \\ &\le 2||Ah||^2 + 2||B(x_t)x_t||^2 + ||\Sigma(x_t)||_F^2 \quad (||u+v||^2 \le 2||u||^2 + 2||v||^2) \\ &\le 2||A||_{\mathrm{op}}^2 \cdot ||h||^2 + 2||B(x_t)x_t||^2 + ||\Sigma(x_t)||_F^2 \quad \text{(by Lemma 4.2.1 (i))} \end{aligned}$$

Now we bound $||B(x_t)(x_t)||$ and $||\Sigma(x_t)||_F$:

$$\begin{aligned}
||B(x_t)x_t|| &\leq ||B(x_t)||_{\text{op}}||x_t|| \\
&\leq ||B(x_t)||_F||x_t|| \quad \text{(by Lemma 4.2.1 (ii))} \\
&= ||\text{vec}(B(x_t))|| \cdot ||x_t|| \\
&= ||W_B x_t + b_B|| \cdot ||x_t|| \\
&\leq (||W_B||_{\text{op}}||x_t|| + ||b_B||) \cdot ||x_t|| \quad \text{(Triangle inequality + Lemma 4.2.1 (i))} \\
&\leq (||W_B||_{\text{op}}M + ||b_B||) \cdot M \\
&= ||W_B||_{\text{op}}M^2 + ||b_B||M \\
&=: C_B \\
&< \infty
\end{aligned}$$

Let $v_t = \text{softplus}(W_\Sigma x_t + b_\Sigma) \in \mathbb{R}^n$, where the softplus function is applied elementwise. Then

$$\begin{aligned}
||\Sigma(x_t)||_F &= ||\text{diag}(v_t)||_F \\
&= ||v_t|| \\
&\leq ||\, |W_\Sigma x_t + b\Sigma| \, + (\log 2)\mathbf{1}|| \quad \text{(by Lemma 4.2.2)} \\
&\leq ||\, |W_\Sigma x_t + b\Sigma| \, || + ||(\log 2)\mathbf{1}|| \quad \text{(by Triangle inequality)} \\
&= ||\, |W_\Sigma x_t + b\Sigma| \, || + \sqrt{n}\log 2 \\
&\leq ||W_\Sigma x_t|| + ||b_\Sigma|| + \sqrt{n}\log 2 \quad \text{(by Triangle inequality)} \\
&\leq ||W_\Sigma||_{\text{op}} \cdot ||x_t|| + ||b_\Sigma|| + \sqrt{n}\log 2 \quad \text{(by Lemma 4.2.1)} \\
&\leq ||W_\Sigma||_{\text{op}} \cdot M + ||b_\Sigma|| + \sqrt{n}\log 2 \\
&=: C_\Sigma
\end{aligned}$$

Hence we have the inequality

$$||b(t,h)||^2 + ||\sigma(t,h)||_F^2 \leq 2||A||_{\text{op}}^2 \cdot ||h||^2 + 2C_B^2 + C_\Sigma^2$$

Choosing $K^2 = \max\{2||A||_{\text{op}}^2, 2C_B^2 + C_\Sigma^2\}$ gives the desired result: If $2||A||_{\text{op}}^2 \geq 2C_B^2 + C_\Sigma^2$, then

$$\begin{aligned}
||b(t,h)||^2 + ||\sigma(t,h)||_F^2 &\leq 2||A||_{\text{op}}^2 \cdot ||h||^2 + 2C_B^2 + C_\Sigma^2 \\
&\leq 2||A||_{\text{op}}^2 \cdot ||h||^2 + 2||A||_{\text{op}}^2 \\
&\leq K^2(1 + ||h||^2)
\end{aligned}$$

and the results also holds if $2C_B^2 + C_\Sigma^2 \geq 2||A||_{\text{op}}^2$ by symmetry.

□

Since the observed inputs $x_1, \ldots, x_T$ are assumed to be fixed, the input process is deterministic and hence trivially adapted to any filtration, because every constant function is measurable with respect to any σ-algebra. Also, it is bounded after standardisation during data pre-processing, hence the input sequence $\{x_1, \dots, x_T\}$ satisfies both requirements as an input process in Theorem 4.2.1.

4.3 Zero-order hold discretisation

Similar to Mamba, we use the zero-order hold (ZOH) method to discretise the parameters A, $B(x_t)$ and $\Sigma(x_t)$. We observe the input sequence $\{x_k\}_{k=1}^T$ at discrete time steps $t_k = k$. Define the **selection gate**

$$\Delta_k := \Delta(x_{t_k}) > 0$$

which is the model's learnt, input-dependent time step chosen at the beginning of time step k, $k \in \{1, \dots, T\}$ and controls the retention of input information. Note that this gate is not the physical data spacing $\Delta t_k = 1$, but rather a latent time step that determines how far the latent state h_t evolves in its continuous-time dynamics between successive data points.

On the sampling interval $[t_k, t_k + \Delta_k)$, under the ZOH assumption, the input and the input-dependent maps are held constant at their values from the start of the interval. We define, for notational simplicity,

$$B_k := B(x_{t_k}); \quad \Sigma_k := \Sigma(x_{t_k})$$

Now equation (4.1) reduces to a linear stochastic differential equation with constant coefficients

$$\mathrm{d}h_s = (Ah_s + B_k x_{t_k})\,\mathrm{d}s + \Sigma_k\,\mathrm{d}W_s, \quad h_0 = \xi \tag{4.2}$$

on some probability space $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\in[0,T]}, \mathbb{P})$, where $(\mathcal{F}_t)_{t\in[0,T]}$ is again the augmented natural filtration generated by the Brownian motion W. To discretise this SDE, we need the following lemma:

Lemma 4.3.1. Let $W = (W_t)_{t\in[0,T]}$ be an m-dimensional standard Brownian motion defined on $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\in[0,T]}, \mathbb{P})$ where the filtration is the augmented natural filtration generated by W. Fix t_k and define, for $\tau \in [0, \Delta_k]$,

$$\tilde{W}_\tau := W_{t_k+\Delta_k} - W_{t_k+\Delta_k-\tau}.$$

Then $(\tilde{W})_{\tau\in[0,\Delta_k]}$ is an m-dimensional Brownian motion and is independent of $\mathcal{F}_{t_k}$.

A proof can be found in Chapter 8.1.

Theorem 4.3.1 (ZOH discretisation of Mamba SDE)**.** Let

- $\Delta_k := \Delta(x_{t_k})$ be the selection gate chosen at time t_k;
- latent times $S_0 := 0$ and $S_{k+1} := S_k + \Delta_k$, and
- $\tilde{B} = (\tilde{B}_\tau)_{\tau\in[0,T]}$ be an m-dimensional Brownian motion that is independent of the physical filtration $(\mathcal{F}_t)_{t\in[0,T]}$ and with independent increments on disjoint latent intervals $[S_k, S_{k+1})$.

On the sampling interval $[t_k, t_k + \Delta_k)$, the solution of equation (4.2) induces the discrete-time,

time-varying linear Gaussian state space model (LGSSM)

$$\begin{aligned} h_{k+1} &= \bar{A}_k h_k + \bar{B}_k x_k + \eta_k, \quad \eta_k \sim \mathcal{N}(0, Q_k) \text{ and independent across } k, \\ y_k &= C(x_k) h_k + \varepsilon_k, \quad \varepsilon_k \sim \mathcal{N}(0, R(x_k)), \end{aligned}$$

where

$$\bar{A}_k = \exp(\Delta_k A), \quad \bar{B}_k = \left(\int_0^{\Delta_k} \exp(\tau A)\, \mathrm{d}\tau \right) B(x_{t_k}), \quad Q_k = \int_0^{\Delta_k} \exp(\tau A) \Sigma(x_{t_k}) \Sigma(x_{t_k})^T \exp(\tau A^T)\, \mathrm{d}\tau,$$

and ε_k and η_k are assumed to be independent. The initial condition $h_0 = \xi$ carries over from equation (4.2).

Proof. We organise this proof into 4 stages:

Stage 1: Deterministic part under ZOH:

This stage solves the SDE for a single step using an integrating factor to derive the form of the deterministic components $\bar{A}_k$ and $\bar{B}_k$.

Let $g(s) := \exp(-sA) h_s$. By Itô's product rule,

$$\begin{aligned} \mathrm{d}(\exp(-sA) h_s) &= (-A \exp(-sA) h_s)\, \mathrm{d}s + \exp(-sA)\, \mathrm{d}h_s \\ &= (-A \exp(-sA) h_s)\, \mathrm{d}s + \exp(-sA) [\, (A h_s + B_k x_{t_k})\, \mathrm{d}s + \Sigma_k\, \mathrm{d}W_s] \\ &= \exp(-sA)(B_k x_{t_k}\, \mathrm{d}s + \Sigma_k\, \mathrm{d}W_s). \end{aligned}$$

Integrating both sides from t_k to $t_k + \Delta_k$:

$$\exp(-(t_k + \Delta_k) A) h_{t_k + \Delta k} - \exp(-t_k A) h_{t_k} = \int_{t_k}^{t_k + \Delta_k} \exp(-sA) B_k x_{t_k}\, \mathrm{d}s + \int_{t_k}^{t_k + \Delta_k} \exp(-sA) \Sigma_k\, \mathrm{d}W_s$$

Multiplying both sides by $\exp((t_k + \Delta_k) A)$:

$$h_{t_k + \Delta_k} = \exp(A \Delta_k) h_{t_k} + \int_{t_k}^{t_k + \Delta_k} \exp((t_k + \Delta_k - s) A) B_k x_{t_k}\, \mathrm{d}s + \int_{t_k}^{t_k + \Delta_k} \exp((t_k + \Delta_k - s) A) \Sigma_k\, \mathrm{d}W_s$$

Let $\tau = t_k + \Delta_k - s$. Then $\tau \in [0, \Delta_k]$ and

$$h_{t_k + \Delta_k} = \exp(A \Delta_k) h_{t_k} + \int_0^{\Delta_k} \exp(\tau A) B_k x_{t_k}\, \mathrm{d}\tau + \int_0^{\Delta_k} \exp(\tau A) \Sigma_k\, \mathrm{d}\tilde{W}_\tau, \tag{4.3}$$

where $\tilde{W}_\tau := W_{t_k + \Delta_k} - W_{t_k + \Delta_k - \tau}$ is also a Brownian motion independent of $\mathcal{F}_{t_k}$ by Lemma 4.3.1. Now define

$$\bar{A}_k := \exp(\Delta_k A), \quad \bar{B}_k := \left(\int_0^{\Delta_k} \exp(\tau A)\, \mathrm{d}\tau \right) B_k,$$

and let η_k to be the stochastic term

$$\eta_k = \int_0^{\Delta_k} \exp(\tau A) \Sigma_k\, \mathrm{d}\tilde{W}_\tau.$$

Then the equation (4.3) becomes an affine recursion when setting $h_k := h_{t_k}$ and $h_{k+1} := h_{t_k+\Delta_k}$:

$$h_{k+1} = \bar{A}_k h_k + \bar{B}_k x_k + \eta_k.$$

This stage is called the deterministic part since we will not rely on the expression of η_k defined here – it is only included to make the stochastic term fully explicit.

Stage 2: Latent-time definition of the process noise η_k

This stage formally defines the process noise η_k on a latent time axis to ensure its mathematical properties are well-behaved. It then proves that η_k is a conditionally Gaussian random variable with zero mean and derives its covariance matrix Q_k.

A subtle issue arises if $\Delta_k > 1$, because the physical time intervals $[t_k, t_k + \Delta_k]$ would overlap, causing η_k to be correlated, while for an LGSSM the noise terms should be independent. This is because Brownian motion only exhibit independent increments over disjoint intervals. To address this issue without restricting Δ_k, the model is formally defined on the latent time axis S_k, driven by $\tilde{B}$. We set

$$\begin{aligned} \eta_k :&= \int_{S_k}^{S_{k+1}} \exp((\tau - S_k)A)\Sigma_k \,\mathrm{d}\tilde{B}\tau \\ &= \int_0^{\Delta_k} \exp(\tau A)\Sigma_k \mathrm{d}\hat{B}\tau \quad \text{(by Lemma 4.3.1)} \end{aligned}$$

where $\hat{B}_\tau := \tilde{B}_{S_k+\tau} - \tilde{B}_{S_k}$. Conditional on $\mathcal{F}_{t_k}$, the integrand $\exp(\tau A)\Sigma_k$ is deterministic and continuous on the interval $[0, \Delta_k]$, hence bounded and square-integrable. Since Itô integrals with deterministic (hence adapted) and square-integrable integrands are martingales with mean zero, we have

$$\mathbb{E}[\eta_k | \mathcal{F}_{t_k}] = 0.$$

This also means the covariance Q_k is given by $\mathbb{E}[\eta_k \eta_k^T | \mathcal{F}_{t_k}]$. Let $H_k(\tau) = \exp(\tau A)\Sigma_k \in \mathbb{R}^{n\times m}$. For any $v \in \mathbb{R}^n$,

$$v^T \eta_k = \int_0^{\Delta_k} v^T H_k(\tau)\,\mathrm{d}\hat{B}_\tau$$

is a scalar Itô integral. By Itô's isometry,

$$\begin{aligned} \mathbb{E}[(v^T\eta_k)^2 | \mathcal{F}_{t_k}] &= \int_0^{\Delta_k} ||v^T H_k(\tau)||^2 \,\mathrm{d}\tau \\ v^T \mathbb{E}[\eta_k \eta_k^T | \mathcal{F}_{t_k}] v &= v^T \left(\int_0^{\Delta_k} H_k(\tau) H_k(\tau)^T \,\mathrm{d}\tau \right) v. \end{aligned}$$

Let $M := \mathbb{E}[\eta_k \eta_k^T | \mathcal{F}_{t_k}]$ and $N := \int_0^{\Delta_k} H_k(\tau) H_k(\tau)^T \,\mathrm{d}\tau$ Since both M and N are symmetric positive semi-definite matrices and $v^T M v = v^T N v$ for all $v \in \mathbb{R}^n$, $M = N$. This implies that

$$Q_k := \mathbb{E}[\eta_k \eta_k^T | \mathcal{F}_{t_k}] = \int_0^{\Delta_k} \exp(\tau A)\Sigma_k \Sigma_k^T \exp(\tau A^T)\,\mathrm{d}\tau$$

Stage 3: Independence across steps

> This stage uses the latent-time construction to rigorously prove that the noise η_k are independent across discrete time steps, a key requirement for the LGSSM framework.

By construction, the latent intervals $[S_k, S_{k+1})$ are disjoint, and $\tilde{B}$ has independent increments over disjoint intervals. Hence the random vectors η_k are conditionally independent across k given the coefficients Δ_k, B_k, Σ_k (which are $\mathcal{F}_{t_k}$-measurable), and independent unconditionally when inputs are deterministic since these coefficients are fixed.

Stage 4: Output equation

> This final stage specifies the observation equation, which links the hidden state h_k to the observed output y_k at discrete measurement times.

Lastly, the output equation are taken at the physical time grid $t_k = k$ (not at $t_k + \Delta_k$). Under ZOH, the instantaneous output map is evaluated at t_k, yielding

$$y_k = C(x_k)h_k + \varepsilon_k, \quad \varepsilon_k \sim \mathcal{N}(0, R(x_k)).$$

□

The following corollary is important for computational purposes, since A is a diagonal matrix with negative entries:

Corollary 4.3.1.1. Suppose $A = \operatorname{diag}(a_1, \dots, a_n)$. Then under the ZOH discretisation in Theorem 4.3.1 at step k with gate $\Delta_k > 0$, we have

$$\begin{aligned}
\bar{A}_k &= \exp(\Delta_k A) = \operatorname{diag}(\exp(a_1\Delta_k), \dots, \exp(a_n\Delta_k));\\
\bar{B}_k &= \left(\int_0^{\Delta_k} \exp(\tau A)\, \mathrm{d}\tau\right) B(x_{t_k}) = \operatorname{diag}\left(\frac{\exp(a_1\Delta_k) - 1}{a_1}, \dots, \frac{\exp(a_n\Delta_k) - 1}{a_n}\right) B(x_{t_k}).
\end{aligned}$$

For the process noise covariance $Q_k = \displaystyle\int_0^{\Delta_k} \exp(\tau A)\Sigma_k\Sigma_k^T \exp(\tau A^T)\, \mathrm{d}\tau$, the entries of Q_k is given by

$$\begin{aligned}
(Q_k)_{ij} &= \left(\int_0^{\Delta_k} \exp(\tau(a_i + a_j))\, \mathrm{d}\tau\right) (\Sigma_k\Sigma_k^T)_{ij}\\
&= \begin{cases} \frac{\exp((a_i+a_j)\Delta_k)-1}{a_i+a_j}(\Sigma_k\Sigma_k^T)_{ij} & \text{if } a_i + a_j \neq 0;\\ \Delta_k(\Sigma_k\Sigma_k^T)_{ij} & \text{if } a_i + a_j = 0. \end{cases}
\end{aligned}$$

Moreover, if $\Sigma_k = \operatorname{diag}(\sigma_{k,1}, \dots, \sigma_{k,n})$, then Q_k is diagonal with

$$(Q_k)_{ii} = \begin{cases} \frac{\exp(2a_i\Delta_k)-1}{2a_i}\sigma_{k,i}^2 & \text{if } a_i \neq 0;\\ \Delta_k\sigma_{k,i}^2 & \text{if } a_i = 0. \end{cases}$$

A proof can be found in Chapter 8.1.

4.4 Exact inference via Kalman filtering

Consider the time-varying LGSSM in Theorem 4.3.1 with fixed inputs $\{x_t\}_{t=1}^T$ (so the matrices $\bar{A}_k, \bar{B}_k, Q_k, C(x_k)$ and $R(x_k)$ are deterministic time-varying). The fundamental task of inference is to compute the probability distribution of the current latent state h_k given all observations up to the point $y_1 \ldots, y_k$ (which we call the *filtering posterior*, i.e. $p(h_k|y_1, \ldots, y_k)$). To derive the closed-form solutions, we need the following facts on multivariate Gaussian distributions:

1. Marginal and conditional distribution of Gaussians is Gaussian ([29, Chapter 8.1.3]): Assume $X \sim \mathcal{N}(\mu, \Sigma)$ where

$$X = \begin{pmatrix} X_a \\ X_b \end{pmatrix}, \quad \mu = \begin{pmatrix} \mu_a \\ \mu_b \end{pmatrix}, \quad \Sigma = \begin{pmatrix} \Sigma_a & \Sigma_c \\ \Sigma_c^T & \Sigma_b \end{pmatrix},$$

   then

   - $p(X_a) \sim \mathcal{N}(\mu_a, \Sigma_a)$, $p(X_b) \sim \mathcal{N}(\mu_b, \Sigma_b)$; and
   - $p(X_a|X_b) \sim \mathcal{N}(\mu_a + \Sigma_c \Sigma_b^{-1}(X_b - \mu_b), \Sigma_a - \Sigma_c \Sigma_b^{-1} \Sigma_c^T)$.

2. Linear combination of Gaussian random vectors is Gaussian (([29, Chapter 8.1.4]): Assume $X \sim \mathcal{N}(\mu_x, \Sigma_x)$ and $Y \sim \mathcal{N}(\mu_Y, \Sigma_Y)$. Then

$$AX + BY + c \sim \mathcal{N}(A\mu_X + B\mu_Y + c, A\Sigma_X A^T + B\Sigma_Y B^T),$$

   where A, B are matrices and c is a vector.

3. Product of two Gaussians is Gaussian up to normalisation (([29, Chapter 8.1.8]): Let $\mathcal{N}(\mu, \Sigma)$ denote the probability density function of some random vector X. Then for some $\mu_1, \mu 2$ and Σ_1, Σ_2,

$$\mathcal{N}(\mu_1, \Sigma_1) \cdot \mathcal{N}(\mu_2, \Sigma_2) \propto \mathcal{N}(\mu_c, \Sigma_c),$$

   where

   - $\mu_c = (\Sigma_1^{-1} + \Sigma_2^{-1})^{-1}(\Sigma_1^{-1}\mu_1 + \Sigma_2^{-1}\mu_2)$; and
   - $\Sigma_c = (\Sigma_1^{-1} + \Sigma_2^{-1})^{-1}$.

Notation. In this section we use the following notations:

- $\hat{h}_{k|k}$ denotes the mean of the posterior distribution $p(h_k|y_1, \ldots, y_k)$;
- $P_{k|k}$ denotes the covariance of the posterior distribution $p(h_k|y_1, \ldots, y_k)$;
- $\hat{h}_{k+1|k}$ denotes the mean of the predictive distribution $p(h_{k+1}|y_1, \ldots, y_k)$. This is the "one-step-ahead" prediction of the state at time $k+1$ before seeing the observation at $k+1$;
- $P_{k+1|k}$ denotes the covariance of the predictive distribution $p(h_{k+1}|y_1, \ldots, y_k)$.

Theorem 4.4.1 (Exact inference for Prob-Mamba)**.** Assume the initial state belief $h_0 = \xi$ is Gaussian, with $h_0 \sim \mathcal{N}(\mu_{0|0}, P_{0|0})$. Then for all time steps $k \geq 0$,

(i) The filtering posterior $p(h_k \mid y_{1:k})$ is Gaussian;

(ii) The posterior parameters $(\hat{h}_{k|k}, P_{k|k})$ can be computed recursively; and

(iii) Given $(\hat{h}_{k|k}, P_{k|k})$, the parameters at step $k+1$ are obtained by the two–step prediction-update procedure:

- Prediction:
$$\hat{h}_{k+1|k} = \bar{A}_k \hat{h}_{k|k} + \bar{B}_k x_k, \quad P_{k+1|k} = \bar{A}_k P_{k|k} \bar{A}_k^T + Q_k$$

- Innovation:
$$e_{k+1} = y_{k+1} - C_{k+1}\hat{h}_{k+1|k}, \quad S_{k+1} = C_{k+1} P_{k+1|k} C_{k+1}^T + R_{k+1}$$

- Gain:
$$K_{k+1} = P_{k+1|k} C_{k+1}^T S_{k+1}^{-1}$$

- Update:
$$\begin{aligned}\hat{h}_{k+1|k+1} &= \hat{h}_{k+1|k} + K_{k+1} e_{k+1} \\ P_{k+1|k+1} &= (I - K_{k+1}C_{k+1}) P_{k+1|k} (I - K_{k+1}C_{k+1})^T + K_{k+1} R_{k+1} K_{k+1}^\top\end{aligned}$$

Moreover, conditional on past outputs and the inputs, the one-step predictive distribution is

$$y_{k+1} | y_1, \ldots, y_k, x_1, \ldots, x_k, \theta \ \sim \ \mathcal{N}(C_{k+1}\hat{h}_{k+1|k}, S_{k+1}),$$

so the conditional marginal log-likelihood of the outputs (by marginalising the latent states $h_1, \ldots, h_T$ out) is given by

$$\log[p(y_1, \ldots, y_T)|x_1, \ldots, x_T] = -\frac{1}{2}\sum_{k=0}^{T-1}[\log \det S_{k+1} + e_{k+1}^T S_{k+1}^{-1} e_{k+1} + d_y \log(2\pi)]$$

This inference approach on LGSSMs is called **Kalman filtering** ([12], [25]). A proof is provided in Chapter 8.1.

Chapter 5

Methods

This chapter formalises the one-step-ahead forecasting task, specifies the Prob-Mamba implementation, and details how the ZOH-discretised LGSSM parameters feed a Kalman filtering head for training and inference. It also documents the datasets used in this project as well as the cleaning and pre-processing pipeline.

5.1 Problem Setup

We forecast one-step-ahead log-returns y_{t+1} for a univariate target, optionally conditioned on multivariate exogenous features $x_t \in \mathbb{R}^{d_x}$ available at time t. Let p_t denote the observed (closing) price or index level, and define the **log return**

$$y_t := \log\left(\frac{p_t}{p_{t-1}}\right) = \log p_t - \log p_{t-1}.$$

If $p_t \in \mathbb{R}^{d_y}$, the return is taken elementwise so that $y_t \in \mathbb{R}^{d_y}$. In this project we set $d_y = 1$ (one series at a time). Econometric baselines are fit univariately to y_t, while deep models take x_t as input and output a scalar $\hat{y}_{t+1}$.

Log-returns are preferred over simple returns $r_t = \frac{p_t - p_{t-1}}{p_{t-1}}$ because:

- **Unbounded support.** Simple returns satisfy $r_t \in [-1, \infty)$, creating a mismatch with Gaussian noise η_k and ε_k. Log-returns are unbounded and better aligned with Gaussian assumptions.
- **Additivity.** Log-returns are additive over time. The return over an interval is simply the sum of daily log returns:

$$\begin{aligned}\log\left(\frac{p_t}{p_{t-k}}\right) &= \log\left(\frac{p_t}{p_{t-1}} \cdot \frac{p_{t-1}}{p_{t-2}} \cdot \ldots \cdot \frac{p_{t-k+1}}{p_{t-k}}\right) \\ &= \log\left(\frac{p_t}{p_{t-1}}\right) + \log\left(\frac{p_{t-1}}{p_{t-2}}\right) + \ldots + \log\left(\frac{p_{t-k+1}}{p_{t-k}}\right) \\ &= y_t + y_{t-1} + \ldots + y_{t-k+1},\end{aligned}$$

  while simple returns must be geometrically compounded.

- **Reduced non-stationarity.** The log transform compresses large values and helps stabilise variance. While log-returns are not guaranteed weakly stationary, they are typically more stable and align with classical econometric assumptions, facilitating fair comparison with deep models.

We enforce adaptedness: each x_t contains only information available up to time t (i.e., x_t is $(\mathcal{F}_t)$-measurable), preventing look-ahead bias. The forecasting task is to learn the conditional distribution of y_{t+1} given $\mathcal{F}_t = \sigma(x_1, \ldots, x_t, y_1, \ldots, y_t)$. Prob-Mamba in Chapter 4 specifies an input-dependent SDE with Gaussian diffusion that, under zero-order-hold (ZOH) discretisation, yields a time-varying LGSSM.

5.2 Model implementation

We implement Prob-Mamba outlined in Chapter 4. The implementation involves four stages:

1. Defining the model's learnable parameters and the input-dependent maps they create;
2. Implementing the ZOH discretisation to compute the time-varying LGSSM matrices from the output of these maps;
3. Learning the LGSSM matrices for the output equation; and
4. Executing Kalman filter's forward pass to compute the marginal log-likelihood, which is used for training.

5.2.1 Model parameterisation

Let d_x, d_y denote input and output dimensions and n denote the latent state dimension. We choose the initial condition $h_0 = 0$ to align with Mamba's implementation ([9, state-spaces/mamba GitHub repository, 2023]). The learnable parameters θ consist of:

- A diagonal continuous-time state matrix $A = \mathrm{diag}(a_1, \ldots, a_n)$ with $a_i \leq 0$ for $i \in \{1, \ldots, n\}$;
- A set of linear layers that form the input-dependent maps defined in Chapter 4.1. These maps take an input x_k and generate the continuous-time LGSSM parameters as well as the output equation components C_k and R_k:
  - Selection gate $\Delta_k := \mathrm{softplus}(w_\Delta^T x_k + b_\Delta)$;
  - Input matrix B_k: A vector computed via $W_b x_k + b_B$ and then reshaped into the $n \times d_x$ matrix B_k;
  - Output matrix C_k: A vector from $W_c x_k + b_C$ is reshaped into $d_y \times n$ matrix C_k;
  - Process noise $\Sigma_k := \mathrm{diag}(\mathrm{softplus}(W_\Sigma x_k + b_\Sigma))$;
  - Observation noise $R_k := \mathrm{diag}(\mathrm{softplus}(W_R x_k + b_R))$.

  All weights and biases are learnt end-to-end: The model is trained by using the Kalman filter to calculate the marginal log-likelihood of the data. The loss is the negative of the value, and gradients are backpropagated through the entire sequence to update all learnable parameters simultaneously.

Since $h_0 = 0$ is deterministic, the state prior parameters reduce to $\mu_{0|0} = 0$, $P_{0|0} = 0$. In practice, we choose $P_{0|0} = \varepsilon I$ for some small $\varepsilon > 0$, because starting with zero covariance makes the filter fatally overconfident in its initial guess, causing it to ignore new data and preventing it from ever correcting its errors. Positivity of Δ_k as well as diagonal entries of R_k and Σ_k are enforced via a softplus activation function and then adding a small $\varepsilon > 0$. This gives strictly positive diagonal entries and ensures Σ_k and R_k are symmetric positive definite. In both cases ε are taken to be 1×10^{-6}.

5.2.2 ZOH discretisation

Define $z_{k,i} = a_i \Delta_k$. For numerical stability we use the scalar helpers

$$\gamma(z) = \frac{\exp(z) - 1}{z}, \quad \rho(z) = \frac{\exp(2z) - 1}{2z}$$

implemented using `expm1` from NumPy (for greater precision when calculating $\exp(z) - 1$ for small z) as documented in the NumPy manual ([23]) with a vectorised small argument branch: for $|z| <$ tol we evaluate $\gamma(z) \approx 1 + \frac{z}{2} + \frac{z^2}{6}$ and $\rho(z) \approx 1 + z + \frac{2z^2}{3}$ to avoid division by zero. The tolerance tol is chosen to be the square root of machine epsilon, which in this case is about $\sqrt{1.19 \times 10^{-7}} = 3.45 \times 10^{-4}$ for `float32`.

The quantities computed per time step k, elementwise for $i = 1, \ldots, n$ are:

- $\bar{A}_k = \mathrm{diag}(\exp(z_{k,1}), \ldots, \exp(z_{k,n}))$;
- $\bar{B}_k = \mathrm{diag}(\gamma(z_{k,1})\Delta_k, \ldots, \gamma(z_{k,n})\Delta_k)B_k$;
- $Q_k = \mathrm{diag}(q_{k,1}, \ldots, q_{k,n})$ where $q_{k,i} = \sigma_i(x_k)^2 \rho(z_{k,i})\Delta_k$.

5.2.3 Learning C_k and R_k

The ZOH step yields $\bar{A}_k, \bar{B}_k$ and Q_k. Now we need to learn C_k and R_k which are not part of the state equation in the LGSSM.

- Observation matrix C_k: We parameterise C_k as a linear projection of the input

$$C_k = \mathrm{reshape}(W_c x_k + b_C, d_y, n),$$

  which reshapes the vector $W_c x_k + b_C$ into a $d_y \times n$ matrix. This allows the output layer from latent state to observations to vary with input context.

- Observation noise R_k. We model heteroskedastic but diagonal noise

$$R_k = \mathrm{diag}(r(x_k)), \quad r(x_k) = \mathrm{softplus}(W_R x_k + b_R) + \varepsilon_R \quad \text{(elementwise)},$$

  where we add ε_R to ensure R_k is symmetric positive definite.

During the filter update, the innovation and its covariance matrix is given by

$$e_{k+1} = y_{k+1} - C_{k+1}\hat{h}_{k+1|k}, \quad S_{k+1} = C_{k+1}P_{k+1|k}C_{k+1}^\top + R_{k+1},$$

and the marginal log-likelihood is given by

$$\log p(y_1,\ldots,y_T|x_1,\ldots,x_T,\theta) = -\frac{1}{2}\sum_{k=0}^{T-1}[\log\det S_{k+1} + e_{k+1}^\top S_{k+1}^{-1} e_{k+1} + d_y \log(2\pi)].$$

by Theorem 4.4.

We minimise the negative marginal log-likelihood with Adam ([16]): Gradients are obtained by autodiff through e_{k+1}, S_{k+1}, $\hat{h}_{k+1|k}$, $P_{k+1|k}$ and flow to W_C, b_C, W_R, b_R and the ZOH parameters. To avoid computing S_k^{-1}, we use the **Cholesky decomposition**:

$$S_{k+1} = L_{k+1}L_{k+1}^\top, \quad \log\det S_{k+1} = 2\sum_i \log(L_{k+1})_{ii}, \quad e_{k+1}^\top S_{k+1}^{-1} e_{k+1} = \|L_{k+1}^{-1}e_{k+1}\|_2^2.$$

5.2.4 Executing the Kalman filter forward pass

Given $\bar{A}_k, \bar{B}_k, Q_k, C_k$ and R_k:

- Prediction:

$$\begin{aligned}\hat{h}_{k+1|k} &= \bar{A}_k\hat{h}_{k|k} + \bar{B}_k x_k;\\ P_{k+1|k} &= \bar{A}_k P_{k|k}\bar{A}_k^T + Q_k\end{aligned}$$

- Update:

$$\begin{aligned}e_{k+1} &= y_{k+1} - C_{k+1}\hat{h}_{k+1|k}, \quad S_{k+1} = C_{k+1}P_{k+1|k}C_{k+1}^T + R_{k+1};\\ K_{k+1} &= P_{k+1|k}C_{k+1}^T S_{k+1}^{-1};\\ \hat{h}_{k+1|k+1} &= \hat{h}_{k+1|k} + K_{k+1}e_{k+1};\\ P_{k+1|k+1} &= (I - K_{k+1}C_{k+1})P_{k+1|k}(I - K_{k+1}C_{k+1})^T + K_{k+1}R_{k+1}K_{k+1}^T\end{aligned}$$

5.3 Datasets and pre-processing

We evaluate our approach using 2 major U.S. stock market daily indices as well Bitcoin data at 5-minute intervals to represent different segments of financial markets:

(i) **NASDAQ Composite Index (IXIC)**: A market capitalization-weighted index that includes over 3,000 stocks listed on the NASDAQ stock exchange, heavily weighted towards technology companies [21].

(ii) **New York Stock Exchange Composite Index (NYSE)**: Represents all common stocks listed on the NYSE, providing a broader representation of the U.S. equity market including traditional industries [22].

(iii) **Bitcoin (BTC–USD)**: This decentralized digital currency ([20]) is the largest and most prominent cryptocurrency. It operates on a 24/7 basis, is known for its high volatility, and is included to represent this growing digital asset class, which behaves differently from traditional equity markets.

5.3.1 NYSE and IXIC data

The IXIC and NYSE datasets are daily, span from January 2010 to November 2023, providing 3,470 trading days of historical data. This 13-year period encompasses various market conditions including the post-2008 recovery, the COVID-19 pandemic and subsequent market volatility, ensuring the models are tested on diverse market regimes. Both datasets are publicly available in .csv files on `https://www.kaggle.com/datasets/ehoseinz/stock-market-prediction/data` and are used as datasets in the paper "*Mamba Meets Financial Markets: A Graph-Mamba Approach for Stock Price Prediction*" [18].

Both datasets were engineered to include 82 features, which we categorise into several groups:

- Technical indicators (22):
  - Momentum indicators: `mom`, `mom1`, `mom2`, `mom3`;
  - Rate of change: `ROC_5`, `ROC_10`, `ROC_15`, `ROC_20`;
  - Exponential Moving Averages: `EMA_10`, `EMA_20`, `EMA_50`, `EMA_200`
  - Temporal Encodings: `TE1`, `TE2`, `TE3`, `TE5`, `TE6`
  - Day Encodings: `DE1`, `DE2`, `DE4`, `DE5`, `DE6`
- Global equity market indices (19):
  - U.S. indices: `DJI`, `GSPC`, `RUT`, `NYSE` (IXIC dataset only), `IXIC` (NYSE dataset only)
  - International indices: `FTSE` (UK), `GDAXI` (Germany), `FCHI` (France), `HSI` (Hong Kong), `SSEC` (China)
  - Futures contracts: `DJI-F`, `S&P-F`, `NASDAQ-F`, `RUSSELL-F`, `Nikkei-F`, `KOSPI-F`, `CAC-F`, `DAX-F`, `FTSE-F`, `HSI-F`, `Dollar Index-F`
- Commodities (10):
  - Energy: `WTI-oil`, `Brent`, `oil`, `GAS-F`,
  - Precious metals: `Gold-F`, `XAU`, `silver-F`, `XAG`
  - Base metals: `copper-F`
  - Agricultural: `wheat-F`
- Foreign exchange rates (9): `EUR`, `GBP`, `JPY`, `CHF`, `AUD`, `NZD`, `CAD`, `CNY`, `Dollar Index`
- Fixed income (8): `DGS5`, `DGS10`, `CTB3M`, `CTB6M`, `CTB1Y`, `DTB3`, `DTB4WK`, `DTB6`
- Corporate bonds (2): `DAAA`, `DBAA`
- Individual stocks (8): `AAPL`, `MSFT`, `AMZN`, `XOM`, `JPM`, `WFC`, `JNJ`, `GE`
- Market microstructure (3): `weekday`, `Price`, `Vol.`

Data cleaning and pre-processing

The data cleaning and pre-processing pipeline for NYSE and IXIC data consists of the following steps:

1. **Preventing look-ahead bias and data leakage**: We construct the next-day log-return target

   $$y_{t+1} = \log p_{t+1} - \log p_t$$

   from the `Price` (for NYSE/IXIC datasets) or `close` (for BTC dataset) series and then remove it from the model inputs. All predictors are restricted to information available no later than the close of the day t. Rolling features such as Rate of change and exponential moving averages are typically computed only using data from the past up to current time step, and missing-values over weekends and holidays are only carried forward and never back-filled from time $t+1$ to time t. An empirical analysis was conducted on the momentum features (`mom`, `mom1`, `mom2`, `mom3`) by correlating them with future simple returns. The results revealed that `mom`, `mom1` and `mom2` contained significant look-ahead bias, with the `mom` column exhibiting a near-perfect correlation of -0.9991 with the 1-day future return. Consequently, all momentum columns were dropped from the feature set to be conservative and ensure model integrity. See Chapter 8.2 for correlation results.

   Features that end with `-F` indicate a futures contract. Because many futures contracts trade continuously and may reflect information that becomes available after the cash market close, all columns corresponding to futures contracts are shifted forward by one trading day. After shifting, the leading row(s) that become `NaN` are dropped. This removes observations that would otherwise require future values to be filled and ensures all retained rows contain only information that would have been available at or before the market close. This rule covers futures contracts such as `DJI-F`, `S&P-F`, `NASDAQ-F`, `RUSSELL-F`, `Nikkei-F`, `KOSPI-F`, `Gold-F`, `silver-F`, etc.

2. **Conversion from .csv to .feather**: The first step converts the cleaned datasets from `.csv` format to `.feather`, a portable file format for storing Apache Arrow tables or data frames. This is done because feather is a columnar memory format, which is significantly faster for data loading and processing compared to text-based, row-oriented CSV format. Arrow avoids the computational overhead of parsing text and converting data types every time the file is read, making subsequent operations much more efficient.

3. **Target and engineering**: The pipeline then creates the target variable by calculating the log-return of the index/asset price for the next time step and save it as `y_next`.

4. **Causal imputation**: Missing values are imputed using forward-fill only: backward-fill or interpolation are avoided to prevent look-ahead bias and data leakage. We forward-fill all columns except `timestamp/date`, `_log_price` (logarithm of the raw price) and `y_next`, drop numeric columns that are entirely `NaN`, then compute a completeness mask (boolean array marking rows with no `NaN`*s*) across numeric features and remove all leading rows before the first fully complete row, since these initial rows cannot be causally recovered.

5. **Chronological splitting**: The dataset is then split into three distinct sets – training, validation and test – roughly based on the 70/15/15 splitting rule:

   - Training set: 4th January, 2010 - 12th August, 2019 (2417 samples, 69.7%)
   - Validation set: 13 August, 2019 - 14th July, 2021 (484 samples, 14.0%)
   - Test set: 15th July, 2021 - 16th October, 2023 (567 samples, 16.3%)

   Chronological splitting is crucial for time-series forecasting to ensure the model is trained on past data and evaluated on its ability to predict future, unseen data. One cannot simply perform *k*-fold cross validation as this will introduce look-ahead bias and data leakage.

6. **Feature selection to mitigate multicollinearity** Since the experiments involves econometric baselines (ARIMA, GARCH) which are prone to multicollinear features, we perform variance inflation factor (VIF)-based pruning ([33]) on the training set to remove redundant features. The procedure builds a causal feature matrix, compute VIFs on the training set, and iteratively eliminates the feature with the highest VIF value:

   (i) **VIF computation.** For each feature x_i in the training matrix X_{train}, we add an intercept and regress x_i on all other predictors to obtain R_i^2, then compute

   $$\text{VIF} = \frac{1}{1 - R_i^2},$$

   where R_i^2 is the coefficient of determination. High VIF_i indicates that x_i is well-explained by the remaining features (strong linear redundancy between feature i and the remaining features).

   (ii) **Greedy backward elimination.** We iteratively drop the feature with the largest VIF if it exceeds the threshold $\tau = 10$, recompute all VIFs on the reduced set, and repeat until all VIFs are less than 10 or fewer than two features remain (since VIF requires at least 2 features to be calculated). Each removal is logged for transparency.

   (iii) **Consistent projection to val/test.** The final retained column set learned on the training split is applied to the validation and test matrices to keep the feature space fixed across splits without peeking at future data.

   The VIF test results can be found in Chapter 8.2. 34 features were dropped from the NYSE dataset while 33 features were dropped for the IXIC dataset.

7. **Feature Normalisation**: Finally, the numeric features are normalised using `StandardScaler` from the `scikit-learn` package in Python. This process scales the features to have zero mean and a standard deviation of 1. Importantly, the scaler is fit only on the training set to learn the scaling parameters (mean and standard deviation). The same fitted scaler is then used to transform the training, validation, and test sets. This prevents information from the validation and test sets from leaking into the training process.

5.3.2 Bitcoin data

The Bitcoin (BTC) datasets are recorded on a 5-minute interval, spanning from 00:00 on 1st January, 2021 to 23:55 on 31st December, 2021. This dataset consists of 5 features only: `open`, `close`, `volume`, `high` and `low`, a standard "OHLCV" dataset common for cryptocurrencies. Since this dataset is engineered by my industrial supervisor Dr. Omer Gunes, no data cleaning is necessary.

Data Pre-processing

The pre-processing pipeline for the BTC dataset is much simpler than that of NYSE and IXIC datasets since it does not contain missing values and comes as a `.feather` file.

1. **Target and feature engineering**: Again we create the target variable by calculating the log-return of the asset price for the next time step and save it as `y_next`.
2. **Chronological splitting**: The dataset is then split into three distinct sets – training, validation and test as follows:
    - Training set: 00:00, 1st January 2021 - 23:55, 13th August 2021 (64626 samples, 61.6%)
    - Validation set: 00:00, 14th August 2021 - 23:55, 7th November 2021 (24744 samples, 23.6%)
    - Test set: 00:00, 8th November 2021 - 23:55, 31st December, 2021 (15551 samples, 14.8%)
3. **Feature Normalisation**: Finally, the numeric features are normalised using `StandardScaler` from the `scikit-learn` package in Python, similar to how the NYSE and IXIC datasets are normalised.

Chapter 6

Experiments

This chapter describes the experimental setup used to assess Prob-Mamba against deterministic and econometric baselines. We state training configurations and hardware, define evaluation metrics, and present the validation pipeline and parameter-budgeting for baselines to ensure fair comparisons.

6.1 Experimental Setup

Training Configuration

All models are trained with identical configurations to ensure that fair, controlled experiments are conducted:

- Optimizer: Adam ([15]) with learning rate 0.001
- Batch size: 64
- Training epochs: 100
- Loss function:
  - For deterministic models, we use **Mean Squared Error** (MSE);
  - For probabilistic models, we use **Negative Log-Likelihood** (NLL).
- Hardware: NVIDIA Tesla V100 GPU (32GB)
- Framework: PyTorch with CUDA acceleration

Furthermore, to ensure fairness between comparisons of deep learning models, each model was tuned to have a similar amount of parameters.

Implementation Details

- Sequence length: We chose $L = 270$ for NYSE and IXIC data, which is approximately one trading year. This gives each model roughly one year of history per window, which is long enough to capture medium-term effects. For high-frequency BTC data, we chose

$L = 288$ time steps (approximately one real-life day), allowing the model to capture intraday seasonalities.

- Input dimension: 50 features for NYSE dataset, 49 features for IXIC dataset and 5 features for Bitcoin dataset.
- Output dimension: 1 (one-step ahead log return)
- Data type: Float32 for all computations

6.2 Evaluation Metrics

1. **Root Mean Squared Error (RMSE)**:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

   where y_i is the actual log return and $\hat{y}_i$ is the predicted log return by the model.

   This metric is used for all models to test their prediction accuracy. A lower RMSE indicates a more accurate prediction.

2. **Quasi-likelihood**:

$$\text{QLIKE} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{\varepsilon_t^2}{\hat{\sigma_t^2}} + \log(\hat{\sigma_t}^2)\right),$$

   where

   - N is the total number of observations being evaluated;
   - ε_t is the squared residual at time t, i.e. the square of the difference between the actual log return y_i and the predicted log return $\hat{y}_i$;
   - $\hat{\sigma_t}^2$ is the forecasted variance of residual at time t.

   This metric is only used for probabilistic models to test their uncertainty quantification ability. A lower QLIKE score indicates a more accurate volatility forecast.

3. **Training time**: The wall-clock time for 100 training epochs (in seconds). These are only measured for deep learning models to evaluate their efficiency in training.

6.3 Baseline Models

We implement three baseline models to compare their performance to that of Prob-Mamba in both prediction accuracy and uncertainty quantification. For the two deep learning baselines (RNN and vanilla Mamba), we use a parameter-matched validation pipeline to pick hyperparameters under fixed budgets.

6.3.1 Validation Pipeline for Deep Learning Baselines

For each dataset, we select model specifications under budget parameters using multi-seed validation:

- Budgets: 100,000 and 300,000 total parameters.
- Candidates: Generated separately for RNN to be as close as possible to each budget.
- Seeds: 0, 1, 2
- Layer counts: 1, 2, 3
- Validation epochs: This is defined as `max(10, round(num_epochs/5))` = 20. We choose this to be lower than the training epochs just to get a rough performance estimate before picking the model.
- Criterion for best candidate model: Validation RMSE. For each candidate we train for 20 epochs under each seed, keep the best validation RMSE per seed, then rank by the mean across seeds.
- Selection & Training: For each budget and model family, take the candidate with the lowest mean validation RMSE, then retrain it for 100 epochs and evaluate its performance on the test set.

Deterministic Models

Recurrent Neural Network (RNN)

We choose simple RNNs with the tanh activation function. The RNN processes sequences using the recurrence relation:

$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h), \\ \hat{y}_t &= W_o h_t + b_o \end{aligned}$$

where

- $x_t \in \mathbb{R}^{d_x}$ is the input vector at time step t;
- $h_t \in \mathbb{R}^H$ is the hidden state at time step t with dimension H;
- $W_{xh} \in \mathbb{R}^{H \times d}$ is the weight matrix mapping input to the hidden state;
- $W_{hh} \in \mathbb{R}^{H \times H}$ is the recurrent weight matrix mapping previous hidden state to current hidden state;
- $b_h \in \mathbb{R}^H$ is the bias vector for the hidden state update; and
- $W_o \in \mathbb{R}^{1 \times H}$ and $b_o \in \mathbb{R}$ maps the hidden state to a one-dimensional log-return prediction.

For each parameter budget, we first solve the quadratic formula for the hidden size H that would exactly match the target number of parameters. Around this closed-form solution, we construct a small window of candidate hidden sizes ($H \pm 10$). For each candidate, we compute the exact parameter count by solving

$$\text{Number of parameters} = (2L-1)H^2 + (I + 2L + O)H + O,$$

where I is the input dimension and $O = 1$ is the output dimension (see Proposition 8.1.1 for a derivation). The candidates are then ranked by the absolute difference between their parameter count and the budget. Only the top 6 specifications that are closest to the budget are retained for validation. This ensures that each model evaluated lies within a narrow tolerance of the intended budget, while avoiding redundant or overly mismatched configurations. The model with the lowest mean validation RMSE is then fully trained.

The model configurations chosen via validation is as follows:

- NYSE dataset:
  - RNN with budget 100,000 : $H = 136$, $L = 3$, number of parameters = 100,097;
  - RNN with budget 300,000: $H = 522, L = 1$, number of parameters = 299,629.
- IXIC dataset:
  - RNN with budget 100,000 : $H = 136$, $L = 3$, number of parameters = 100,233;
  - RNN with budget 300,000: $H = 521, L = 1$, number of parameters = 299,055.
- BTC dataset:
  - RNN with budget 100,000 : $H = 140$, $L = 3$, number of parameters = 99,681;
  - RNN with budget 300,000: $H = 314, L = 2$, number of parameters = 298,929.

Vanilla Mamba

We choose vanilla Mamba models [9], where each Mamba block is a selective state-space model (SSM) with input-dependent parameters. For a multi-block Mamba, we first project the dataset features $x_t \in \mathbb{R}^{d_{\text{in}}}$ to $\tilde{x}_t \in \mathbb{R}^{d_{\text{model}}}$ (identity if $d_{\text{in}} = d_{\text{model}}$). At each time step, a Mamba block maintains a latent state $h_t \in \mathbb{R}^{d_{\text{state}}}$ and processes the input $\tilde{x}_t \in \mathbb{R}^{d_{\text{model}}}$ through the recurrence

$$\begin{aligned} \frac{\mathrm{d}h(t)}{\mathrm{d}t} &= Ah(t) + B(\tilde{x}(t))\,\tilde{x}(t), \\ z_t &= C(\tilde{x}(t))\,h(t), \end{aligned}$$

as outlined in Chapter 2.2. In practice, each block also includes a depthwise convolution of kernel size d_{conv} applied before the SSM update to capture short-range dependencies, and an expansion step that widens the intermediate channel dimension to expand d_{model} times before projecting back to d_{model}. A final linear head $W_o \in \mathbb{R}^{1 \times d_{\text{model}}}$, $b_o \in \mathbb{R}$ maps z_t to a scalar prediction $\hat{y}_t \in \mathbb{R}$ corresponding to the one-step-ahead log return.

The hyperparameters thus have the following interpretations:

- d_{state}: latent state dimension, i.e. the number of decay modes in the continuous-time SSM;
- d_{model}: feature dimension of the sequence representation passed between blocks (block input/output dimension);
- d_{conv}: convolution kernel size providing local context before the SSM update (no direct analogue in the continuous-time theory);
- expand: expansion factor determining the width of the intermediate projections used to parameterise $B(x), C(x), \Delta(x)$.

For each parameter budget, we construct candidate Mamba models by varying the feature dimension d_{model} over the grid $\{64, 72, 80, \ldots, 512\}$ and stacking $L \in \{1, 2, 3\}$ blocks with fixed $d_{\text{state}} = 64$, $d_{\text{conv}} = 4$, and expand $= 2$. For each configuration we compute the exact parameter count and rank candidates by their distance to the target budget. Only the top 6 specifications closest to the budget are retained for validation. Each candidate is trained for 20 epochs under three random seeds, and the model with the lowest mean validation RMSE is then retrained for the full 100 epochs and evaluated on the test set.

The model configurations chosen via validation is as follows:

- NYSE dataset:
  - Mamba with budget 100,000 : $d_{\text{model}} = 136$, $L = 3$, number of parameters = 100,097;
  - Mamba with budget 300,000: $d_{\text{model}} = 136$, $L = 1$, number of parameters = 299,629.
- IXIC dataset:
  - Mamba with budget 100,000 : $d_{\text{model}} = 136$, $L = 3$, number of parameters = 100,097;
  - Mamba with budget 300,000: $d_{\text{model}} = 136$, $L = 1$, number of parameters = 299,629.
- BTC dataset:
  - Mamba with budget 100,000 : $d_{\text{model}} = 112$, $L = 3$, number of parameters = 91,729;
  - Mamba with budget 300,000: $d_{\text{model}} = 200$, $L = 1$, number of parameters = 274,201.

Probabilistic Models

ARMA–GARCH

We use a univariate ARMA mean with a GARCH(1,1) conditional variance and Gaussian standardized shocks. Since log returns are approximately stationary due to the log transformation, we chose $d = 0$ as a modelling decision based on the principle of parsimony (an ARIMA model with $d = 0$ corresponds to an ARMA model). Let y_t denote the (scaled) log return and $\varepsilon_t = y_t - \mu_t$ the mean-corrected residual. The model is

$$\begin{aligned} \text{Mean (ARMA}(p,q)\text{)}: \quad & y_t = \mu \; + \; \sum_{i=1}^{p} \phi_i \, y_{t-i} \; + \; \sum_{j=1}^{q} \theta_j \, \varepsilon_{t-j} \; + \; \varepsilon_t, \\ \text{Shocks}: \quad & \varepsilon_t = \sigma_t z_t, \qquad z_t \mid \mathcal{F}_{t-1} \sim \mathcal{N}(0,1), \\ \text{Variance (GARCH(1,1))}: \quad & \sigma_t^2 = \omega \; + \; \alpha \, \varepsilon_{t-1}^2 \; + \; \beta \, \sigma_{t-1}^2, \end{aligned}$$

with $\sigma_t^2 > 0$ almost surely. In statistics and econometrics, the innovation or shock at time t is often written as $\varepsilon_t = \sigma_t z_t$, where (z_t) are i.i.d. standard normal variables. More formally, $z_t \mid \mathcal{F}_{t-1} \sim \mathcal{N}(0,1)$, where $\mathcal{F}_{t-1}$ denotes the information set available up to time $t-1$. The one-step-ahead forecast delivers both a mean prediction $\hat{y}_{t+1}$ and a variance forecast $\hat{\sigma}_{t+1}^2$.

On the training set, we select the ARMA orders (p,q) by Bayesian Information Criterion (BIC) over $p \in \{0,1,2,3\}$, $q \in \{0,1,2,3\}$ (skipping $(0,0)$), fitting ARIMA(order=$(p,0,q)$, trend=`'n'`) without enforcing stationarity/invertibility. The chosen order is then refit on a concatenate dataset of training and validation set and used to forecast the full test horizon. Next, we fit a GARCH(1,1) to the ARMA residuals using `arch_model` with `mean='Zero'`, `vol='GARCH'`, `p=1`, `q=1`, and `dist='normal'`. For numerical stability, residuals are multiplied by a dataset-specific scale during estimation (100 for equities data, 1000 for BTC data). The resulting variance forecasts are then unscaled by dividing by the square of the same factor. Forecasts of the ARMA mean and the GARCH variance are combined to evaluate out-of-sample performance on the test set.

The models chosen by BIC are:

- NYSE data: AR(1);
- IXIC data: AR(1);
- BTC data: ARMA(3,3).

6.4 Prob-Mamba

Seq2seq targets and loaders. For each dataset, we build sequence-to-sequence loaders with sequence length L equal to the input window used by the deep baselines. Given a dataframe with the one-step-ahead target column y_{next}, we form

$$Y_i = (y_{\text{next},i}, y_{\text{next},i+1}, \dots, y_{\text{next},i+L-1}) \in \mathbb{R}^L,$$

and package inputs/targets as tensors of shape (B, L, d_{in}) and $(B, L, 1)$, respectively. This seq2seq target allows the probabilistic head to compute per-step innovations across the whole window during training and validation.

Model construction under a parameter budget. Given an input dimension d_{in}, we instantiate a Prob-Mamba model with one Mamba block and an LGSSM Kalman head. We denote the feature width by d_{model} and fix

$$n_{\text{state}} = 16, \quad d_{\text{state}} = 16, \quad d_{\text{conv}} = 4, \quad \text{expand} = 2,$$

and search the feature width $d_{\text{model}} \in \{32, 40, \dots, 192\}$ to match a target parameter budget, in which we chose to be 100,000. We select the first configuration whose total parameter count is within a tolerance of 3% of the budget; otherwise we return the closest match found.

Training objective and loss function The forward pass expects both inputs and targets (X, Y) so the Kalman head can compute exact innovations. We optimise the Gaussian negative log-likelihood over the sequence:

$$NLL = \frac{1}{2}\sum_{t=1}^{L}\left[\log(2\pi) + \log S_t + \frac{v_t^2}{S_t}\right], \quad v_t = y_t - \hat{y}_{t|t-1}, \;\; S_t = \mathrm{Var}(y_t \mid y_1, \ldots, y_{t-1}),$$

averaged per sample in the loader.

Evaluation (last-step metrics). During evaluation, we report

(i) the average NLL per sample returned by the model, and

(ii) last-step point and volatility metrics.

Let L be the window length and let $i = 1, \ldots, N$ index test windows. Denote the model's predictive mean/variance sequences by $\{\hat{\mu}_t^{(i)}\}_{t=1}^{L}$ and $\{\hat{\sigma}_t^{2,(i)}\}_{t=1}^{L}$, with targets $\{y_t^{(i)}\}_{t=1}^{L}$. We evaluate only the last step:

$$\hat{\mu}_i := \hat{\mu}_L^{(i)}, \qquad \hat{\sigma}_i^2 := \hat{\sigma}_L^{2,(i)}, \qquad y_i := y_L^{(i)}.$$

For numerical stability we clamp $\tilde{\sigma}_i^2 := \max(\hat{\sigma}_i^2, \varepsilon)$ with $\varepsilon = 10^{-12}$, and compute

$$\mathrm{RMSE} \;=\; \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{\mu}_i - y_i)^2}, \qquad \mathrm{QLIKE} \;=\; \frac{1}{N}\sum_{i=1}^{N}\left(\frac{(y_i - \hat{\mu}_i)^2}{\tilde{\sigma}_i^2} + \log \tilde{\sigma}_i^2\right).$$

Using only the last element of each window mirrors a true one-step-ahead forecast within each sliding window.

Validation and model selection. Unlike the deep baselines, we do not run a candidate/seed validation sweep for Prob-Mamba. This choice is pragmatic: the Kalman head makes each epoch substantially more expensive than a deterministic head. From , we can see that for 100,000 parameter models, the training time of Prob-Mamba is about 200 times slower than that of vanilla Mamba. A full validation regime would therefore multiply compute by orders of magnitude and is very computationally expensive, hence not realistic given the time-frame of this project. We therefore train the budget-matched model for 100 epochs and report its validation/test NLL, RMSE and QLIKE.

The chosen model configurations are:

- NYSE dataset: $d_{\text{model}} = 32$, number of parameters = 92,802 (72,068 parameters from Kalman head);
- IXIC dataset: $d_{\text{model}} = 32$, number of parameters = 92,834 (72,068 parameters from Kalman head);
- BTC dataset: $d_{\text{model}} = 64$, number of parameters = 101,954 (68,820 parameters from Kalman head).

Chapter 7

Results and Discussion

This chapter presents the empirical results from our experiments on the Probabilistic Mamba (Prob-Mamba) model, compared against baselines including ARIMA/GARCH, vanilla Mamba, and simple RNN architectures. We evaluate performance on three datasets: daily NYSE Composite (NYSE), NASDAQ Composite (IXIC), and 5-minute Bitcoin (BTC) data. The analysis focuses on point forecasting accuracy (RMSE), volatility forecasting (QLIKE), and probabilistic metrics (NLL). We also interpret the training dynamics via NLL curves, discuss computational efficiency, and outline limitations with suggestions for future work.

7.1 Experimental Results

We denote

- Simple RNN with parameter budget 100,000 and 300,000 to be RNN-100k and RNN-300k respectively; and
- Simple RNN with parameter budget 100,000 and 300,000 to be RNN-100k and RNN-300k respectively.

NYSE Dataset Performance

Table 7.1 presents the comprehensive evaluation results on the NYSE dataset:

Table 7.1: Performance comparison on NYSE dataset

Model	Training Time (s)	Test RMSE	Test QLIKE
RNN-100k	13.60	0.002160	N/A
Mamba-100k	20.27	0.017704	N/A
RNN-300k	37.94	0.002015	N/A
Mamba-300k	32.11	0.010637	N/A
ARMA+GARCH	N/A	0.010569	-8.109088
Prob-Mamba	5764.7	0.007359	-8.649899

On the NYSE dataset, Prob-Mamba achieves a competitive RMSE of 0.0074, outperforming both vanilla Mamba variants and the ARMA component of the ARMA+GARCH baseline, though it remains behind the RNN models in predictive accuracy. In volatility scoring, Prob-Mamba attains a lower QLIKE (-8.65 vs -8.11), surpassing the GARCH component of the same baseline, which indicates better-calibrated volatility forecasts.

IXIC Dataset Performance

Table 7.2 presents the comprehensive evaluation results on the IXIC dataset:

Table 7.2: Performance comparison on IXIC dataset

Model	Training Time (s)	Test RMSE	Test QLIKE
RNN-100k	13.66	0.002772	N/A
Mamba-100k	18.82	0.023741	N/A
RNN-300k	38.37	0.003914	N/A
Mamba-300k	31.37	0.019491	N/A
ARMA+GARCH	N/A	0.015694	-7.102800
Prob-Mamba	5766.52	0.009855	-8.243943

For the IXIC dataset, Prob-Mamba again outperforms both vanilla Mamba models in terms of predictive accuracy while falling behind both RNN models. It also outperforms the econometric baseline ARMA-GARCH in both predictive accuracy as well as volatility calibration.

BTC Dataset Performance

Table 7.3 presents the comprehensive evaluation results on the Bitcoin (5-minute) dataset:

Table 7.3: Performance comparison on BTC dataset

Model	Training Time (s)	Test RMSE	Test QLIKE
RNN-100k	398.78	0.003375	N/A
Mamba-100k	533.76	0.002199	N/A
RNN-300k	5238.42	0.022428	N/A
Mamba-300k	912.96	0.002348	N/A
ARIMA+GARCH	Negligible	0.002145	-10.969096
Prob-Mamba	14702.4	0.003995	-9.044412

On high-frequency data, vanilla Mamba is now the best deterministic point forecaster. Prob-Mamba was partially trained but halted after 20 epochs due to prohibitive runtime, as shown in the amount of training time. At that point it had achieved RMSE = 0.00399 and QLIKE = -9.04, showing promise but not directly comparable to the fully trained baselines. This underlines the computational burden of the probabilistic head on high-frequency data, where the long sequences exacerbate the Kalman filtering cost.

7.2 Analysis

7.2.1 Performance Analysis

On daily equities (NYSE, IXIC), the RNN attains the lowest RMSE among deterministic baselines, while vanilla Mamba trails despite comparable sequence lengths (equities windows $L = 270$). In contrast, on BTC (5-minute), where the window length is similar ($L = 288$) but the sampling frequency and number of training windows are much higher, vanilla Mamba surpasses RNN in RMSE. This reflects richer intra-day dynamics (short-horizon momentum, microstructure noise) and larger data volume, conditions under which Mamba's state-space recurrence and linear-time scan can better capture fast, local temporal variation.

For probabilistic scoring, Prob-Mamba consistently improves QLIKE over GARCH on equities, indicating better-calibrated conditional variance (lower/more negative QLIKE is better). Simultaneously, it beats vanilla Mamba in RMSE on equities, showing that introducing stochastic dynamics and exact inference can improve both mean accuracy (vs. deterministic Mamba) and volatility calibration (vs. GARCH). The results highlight Prob-Mamba's strengths in probabilistic forecasting and showed the possibility of capturing heteroskedasticity of financial returns through the input-dependent diffusion term in the SDE. On BTC, the Prob-Mamba run was halted early for compute reasons, so its numbers are not directly comparable to fully trained baselines; we therefore refrain from ranking it there.

7.2.2 Interpreting the NLL learning curves

Here we show the NLL vs Epoch graphs for Prob-Mamba for both NYSE and IXIC datasets:

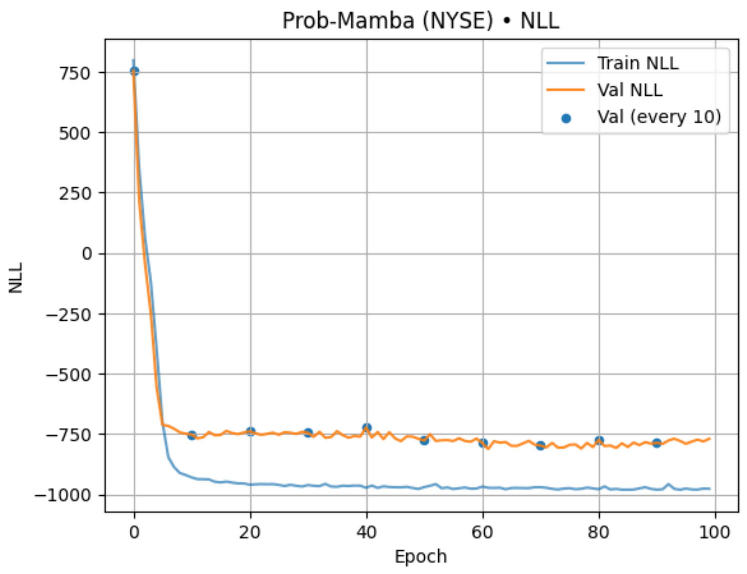

Figure 7.1: NLL vs Epoch for Prob-Mamba on NYSE dataset

From Figure 7.1, we can observe that the training NLL decreases sharply from around 750 to around -900 within the first 20 epochs and then gradually converges with minor fluctuations. The

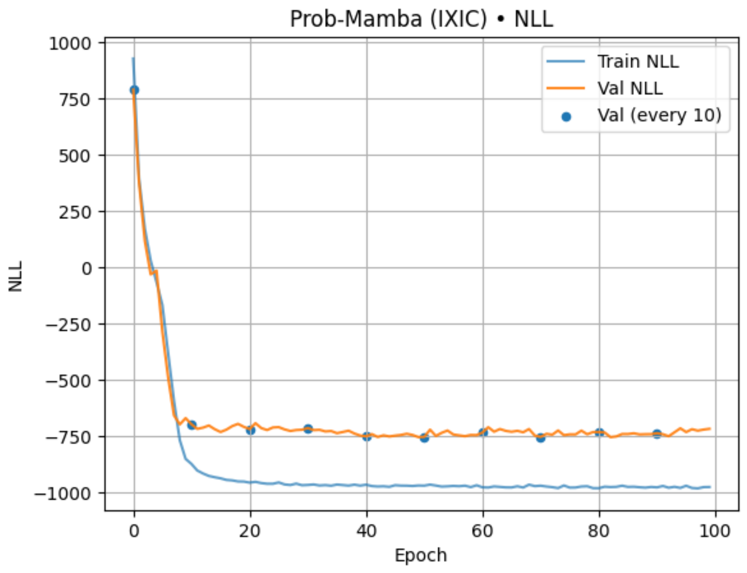

Figure 7.2: NLL vs Epoch for Prob-Mamba on IXIC dataset

same can also be said about the validation NLL, which rapidly drops from 750 to -750 within the first 10 epochs. A very similar trend can also be found in Figure 7.2.

Both figures s show a sharp initial drop followed by a gradual, steady decline with mild oscillations. Early on, the model corrects substantial miscalibration by simultaneously reducing the normalised innovation v_t^2/S_t and the predicted log-variance $\log S_t$ (here S_t is the innovation variance). Subsequently, improvements are incremental as the Kalman filter trades off mean accuracy (smaller innovations) against variance calibration (avoiding unnecessarily large S_t). The small train–validation gap indicates limited overfitting. It is worth noting that training is performed by minimising the NLL, while evaluation is based on last-step RMSE and QLIKE. This mismatch partly explains the brief validation bumps, which often coincide with volatility spikes: in those periods the model must prioritise calibrating variance forecasts for NLL optimisation, even if this temporarily worsens last-step predictive accuracy.

7.2.3 Computational Efficiency: Why is Prob-Mamba so slow and inefficient?

As we can see, Prob-Mamba is much slower than vanilla Mamba (at least 2 magnitudes slower and gets worse with higher-frequency data). This is mainly due to the following:

(i) **Per-step time-varying LGSSM work**. At every time step Prob-Mamba must

  (a) discretise the input-dependent, continuous SSM under ZOH to obtain $(\bar{A}_k, \bar{B}_k, Q_k)$, and

  (b) perform a full Kalman predict-update.

  Because these parameters change with k and the Kalman filter gain depends on the previous covariance, the recursion is inherently sequential and cannot be fused into Mamba's highly optimised parallel scan kernels.

(ii) **Backpropagation through time-varying inference**: During training, autodiff must traverse all L Kalman updates with input-dependent parameters, inflating compute and memory relative to a linear readout head.

(iii) **Parameter placement**: Observe that, in Chapter 6.4, a large fraction of parameters sits in the probabilistic head, increasing per-step tensor traffic and making the forward/backward passes compute-bound.

Together, these factors explain the order-of-magnitude wall-time gap observed between Prob-Mamba and vanilla Mamba, and why the BTC run was curtailed despite the sequence length L being similar to equities: the high frequency yields far more windows, so the per-step cost compounds across many more sequences.

7.3 Limitations and Future Work

Compute and scalability: Prob-Mamba's probabilistic head requires per-step ZOH discretisation and a full Kalman predict–update with input-dependent parameters. Backpropagating through this time-varying recursion is inherently sequential and compute-bound, making the method two to three orders of magnitude slower than a budget-matched deterministic Mamba. Despite its theoretical appeal, Prob-Mamba faces practical limitations: the computational overhead limits scalability to longer sequences or larger latent dimensions n, making it impractical under standard training budgets. This constrained the breadth of hyperparameter/seed sweeps and forced early stopping on the BTC experiment.

Training/evaluation scope: For equities we report single (or few) runs per model and focus on last-step RMSE and QLIKE. We did not evaluate calibration beyond QLIKE, for example measuring coverage using prediction intervals, and did not report full learning-curve variability across seeds. The BTC Prob-Mamba run was truncated at 20 epochs, so those numbers are not directly comparable to fully trained baselines.

Modelling assumptions: The current head assumes Gaussian noise with diagonal covariances and univariate outputs. This excludes heavy tails and correlated latent diffusion, which is possible in financial time series.

Ablations and diagnostics: We did not systematically vary window length L, feature sets, or probabilistic head parameterisation to investigate when vanilla Mamba overtakes RNN on daily data, or how much of Prob-Mamba's QLIKE gain comes from $R(x_k)$ vs. $Q(x_k)$ learning. We also did not compare against probabilistic deep learning baselines.

Future Work

Custom kernels for efficiency: Vanilla Mamba achieves its speed through custom CUDA kernels that fuse the recurrent scan into hardware-friendly operations. In contrast, Prob-Mamba currently uses unfused PyTorch primitives for discretisation and Kalman updates, leading to large overhead. Developing dedicated GPU kernels that integrate ZOH discretisation, innovation computation, and covariance updates could reduce this gap by orders of magnitude, making Prob-Mamba scalable to longer sequences and higher-dimensional latent states.

Non-Gaussian likelihood: Train Prob-Mamba using student t-distributions or skewed distribution to model heavy-tail characteristics in financial time series.

Approximate Inference instead of exact inference: Approximate inference for speed. Replace the exact per-step Kalman recursion with fast, approximate procedures that reduce sequential dependencies and heavy linear-algebra.

7.4 Conclusion

This dissertation introduced Prob-Mamba, a probabilistic extension of the Mamba state-space architecture that augments selective SSM dynamics with SDE-based stochasticity and performs exact inference via a Kalman filtering head. The framework yields time-varying LGSSM parameters from inputs and trains end-to-end on sequence negative log-likelihood.

Empirically, on daily equities (NYSE, IXIC), Prob-Mamba improves RMSE over vanilla Mamba and outperforms GARCH on QLIKE, indicating more accurate means than the deterministic Mamba and better-calibrated volatility than a classical volatility model. We also observe a frequency-dependent pattern among deterministic baselines: RNNs excel on low-frequency equities, while vanilla Mamba performs better on high-frequency BTC despite similar window lengths, consistent with richer intra-day dynamics and larger data volume favouring Mamba's state-space recurrence. The principal trade-off is computational: the probabilistic head is two to three orders of magnitude slower than vanilla Mamba. Despite its theoretical appeal, this computational overhead limits scalability to longer sequences or larger latent dimensions n, making Prob-Mamba impractical under standard training budgets and constraining hyperparameter exploration and high-frequency experiments, despite being a mathematically sound model.

Overall, Prob-Mamba shows that introducing principled stochastic dynamics and exact inference into selective SSMs can enhance both point accuracy and variance calibration. With approximate Kalman filtering or custom kernels to overcome its computational bottleneck, future probabilistic Mamba variants could extend this benefit to higher-frequency settings.

Chapter 8

Appendices

8.1 Proofs

Proposition 2.2.1. (ZOH discretisation) The zero-order hold discretisation of the S4 state equation assumes the input $x(s)$ is held constant over the sampling interval $[t, t+\Delta)$, i.e. $x(s) = x_t$ for $s \in [t, t+\Delta)$. This yields

$$\bar{A} = \exp(\Delta A), \quad \bar{B} = \left(\int_0^{\Delta} \exp(\tau A)\, \mathrm{d}\tau \right) B.$$

(i) If A is diagonal, i.e. $A = \operatorname{diag}(a_1, \ldots, a_n)$, then letting $B = (b_1, \ldots, b_n)^T$ gives

$$(\bar{A})_{ii} = \exp(a_i \Delta), \quad \bar{B}_i = \begin{cases} \frac{\exp(a_i\Delta)-1}{a_i} & \text{if } a_i \neq 0; \\ \Delta & \text{if } a_i = 0. \end{cases}$$

(ii) If A is invertible, then

$$\bar{B} = (\Delta A)^{-1}(\exp(\Delta A) - I)\Delta B$$

Proof. Define the integrating factor $I(s) := \exp(-sA)h(s)$. Differentiating gives:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}s} I(s) &= -A \exp(-sA)h(s) + \exp(-sA) \frac{\mathrm{d}}{\mathrm{d}s} h(s) \\ &= \exp(-sA) \left(-Ah(s) + \frac{\mathrm{d}}{\mathrm{d}s} h(s) \right) \\ &= \exp(-sA)Bx(s) \end{aligned}$$

Integrating both sides from t to $t + \Delta$ gives:

$$\begin{aligned} I(t+\Delta) - I(t) &= \int_t^{t+\Delta} \exp(-sA)Bx(s)\, \mathrm{d}s \\ \exp(-(t+\Delta)A)h(t+\Delta) - \exp(-tA)h(t) &= \int_t^{t+\Delta} \exp(-sA)Bx(s)\, \mathrm{d}s \end{aligned}$$

Multiplying both sides by $\exp((t+\Delta)A)$:

$$h(t+\Delta) - \exp(\Delta A)h(t) = Bx_t \int_t^{t+\Delta} \exp((t+\Delta-s)A)\,\mathrm{d}s \quad (x(s) = x_t \text{ for } s \in [t, t+\Delta))$$

Let $\tau = t + \Delta - s$. Then

$$h(t+\Delta) = \exp(\Delta A)h(t) + Bx_t \int_0^{\Delta} \exp(\tau A)\,\mathrm{d}\tau$$

which gives

$$\bar{A} = \exp(\Delta A), \quad \bar{B} = \left(\int_0^{\Delta} \exp(\tau A)\,\mathrm{d}\tau\right) B.$$

(i) For diagonal A, we have

$$\begin{aligned}
\exp(\Delta A) &= \operatorname{diag}(\exp(a_i\Delta)),\\
\int_0^{\Delta} \exp(\tau A)\,\mathrm{d}\tau &= \operatorname{diag}\left(\int_0^{\Delta} \exp(a_i\tau)\,\mathrm{d}\tau\right)\\
\int_0^{\Delta} \exp(a_i\tau)\,\mathrm{d}\tau &= \begin{cases} \frac{\exp(a_i\Delta)-1}{a_i} & \text{if } a_i \neq 0;\\ \Delta & \text{if } a_i = 0. \end{cases}
\end{aligned}$$

This gives

$$(\bar{A})_{ii} = \exp(a_i\Delta), \quad \bar{B}_i = \begin{cases} \frac{\exp(a_i\Delta)-1}{a_i} & \text{if } a_i \neq 0;\\ \Delta & \text{if } a_i = 0. \end{cases}$$

(ii) For invertible A,

$$\begin{aligned}
\int_0^{\Delta} \exp(\tau A)\,\mathrm{d}\tau &= [A^{-1}\exp(\tau A)]_0^{\Delta}\\
&= A^{-1}(\exp(\Delta A) - I)\\
\bar{B} &= A^{-1}(\exp(\Delta A) - I)B\\
&= (\Delta A)^{-1}(\exp(\Delta A) - I)\Delta B.
\end{aligned}$$

□

Lemma 4.2.1 For any $A = (a_{ij}) \in \mathbb{R}^{n\times n}$ and $x \in \mathbb{R}^n$, define the

- operator norm $||A||_{\text{op}} := \sup\limits_{x\neq 0} \dfrac{||Ax||}{||x||}$;
- Frobenius norm $||A||_F := \left[\sum_{i=1}^{n}\sum_{j=1}^{n} |a_{ij}|^2\right]^{1/2}$, where $||\cdot||$ is the Euclidean norm for vectors.

  Then the following holds:

  (i) $||Ax|| \leq ||A||_{\text{op}} \cdot ||x||$;

  (ii) $||A||_{\text{op}} \leq ||A||_F$.

Proof. (i) By definition of the operator norm,

$$\|A\|_{\mathrm{op}} = \sup_{x\neq 0} \frac{\|Ax\|}{\|x\|} \quad \Rightarrow \quad \frac{\|Ax\|}{\|x\|} \le \|A\|_{\mathrm{op}} \ \ \forall\, x \neq 0,$$

so $\|Ax\| \le \|A\|_{\mathrm{op}}\|x\|$ for all x.

(ii) Write the rows of A as $r_1^\top, \dots, r_n^\top$. For any x with $\|x\| = 1$,

$$\|Ax\|^2 = \sum_{i=1}^{n} (r_i^\top x)^2 \le \sum_{i=1}^{n} \|r_i\|^2 \|x\|^2 = \sum_{i=1}^{n} \|r_i\|^2 = \|A\|_F^2,$$

using Cauchy–Schwarz on each term. Taking the supremum over $\|x\| = 1$ gives $\|A\|_{\mathrm{op}} \le \|A\|_F$.

□

Lemma 4.2.2 Let $x = (x_1, \dots, x_n)^T \in \mathbb{R}^n$. Then

$$\mathrm{softplus}(x) \le |x| + (\log 2)\mathbf{1},$$

where $|x| = (|x_1|, \dots, |x_n|)^T$, $\mathbf{1} = (1, \dots, 1) \in \mathbb{R}^n$ and the softplus function is applied elementwise.

Proof. It suffices to prove the scalar inequality $\log(1+e^u) \le |u| + \log 2$. If $u \ge 0$, then $1+e^u \le 2e^u$, so $\log(1+e^u) \le u + \log 2 = |u| + \log 2$. If $u \le 0$, then $1 + e^u \le 2 \le 2e^{|u|}$, so $\log(1+e^u) \le |u| + \log 2$. Applying this elementwise to $x \in \mathbb{R}^n$ gives

$$\mathrm{softplus}(x) \le |x| + (\log 2)\,\mathbf{1}.$$

□

Lemma 4.3.1 Let $W = (W_t)_{t\in[0,T]}$ be an m-dimensional standard Brownian motion defined on $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\in[0,T]}, \mathbb{P})$ where the filtration is the augmented natural filtration generated by W. Fix t_k and define, for $\tau \in [0, \Delta_k]$,

$$\tilde{W}_\tau := W_{t_k+\Delta_k} - W_{t_k+\Delta_k-\tau}.$$

Then $(\tilde{W})_{\tau\in[0,\Delta_k]}$ is an m-dimensional Brownian motion and is independent of $\mathcal{F}_{t_k}$.

Proof. We verify the Brownian properties for $\widetilde{W}$:

1. *Starts at* 0 *and has continuous paths.* Clearly $\widetilde{W}_0 = 0$, and continuity follows from continuity of W.

2. *Gaussian, stationary increments with the correct covariance.* For $0 \le \tau_1 < \tau_2 \le \Delta_k$,

$$\widetilde{W}_{\tau_2} - \widetilde{W}_{\tau_1} = W_{t_k+\Delta_k-\tau_1} - W_{t_k+\Delta_k-\tau_2}.$$

By stationary increments, this is distributed as $\mathcal{N}\big(0, (\tau_2-\tau_1)I_m\big)$; and increments over disjoint time intervals are independent by the independent-increments property of W.

3. *Independence from $\mathcal{F}_{t_k}$.* For each $\tau \in [0, \Delta_k]$,

$$\widetilde{W}_\tau = W_{t_k+\Delta_k} - W_{t_k+\Delta_k-\tau}$$

is an increment of W over the interval $(t_k + \Delta_k - \tau,\, t_k + \Delta_k]$, which lies strictly after t_k. Hence $\widetilde{W}$ is independent of $\mathcal{F}_{t_k}$.

$\square$

Corollary 4.3.1.1 Suppose $A = \operatorname{diag}(a_1, \ldots, a_n)$. Then under the ZOH discretisation in Theorem 4.3.1 at step k with gate $\Delta_k > 0$, we have

$$\begin{aligned}
\bar{A}_k &= \exp(\Delta_k A) = \operatorname{diag}(\exp(a_1\Delta_k), \ldots, \exp(a_n\Delta_k));\\
\bar{B}_k &= \left(\int_0^{\Delta_k} \exp(\tau A)\,\mathrm{d}\tau\right) B(x_{t_k}) = \operatorname{diag}\left(\frac{\exp(a_1\Delta_k) - 1}{a_1}, \ldots, \frac{\exp(a_n\Delta_k) - 1}{a_n}\right) B(x_{t_k}).
\end{aligned}$$

For the process noise covariance $Q_k = \displaystyle\int_0^{\Delta_k} \exp(\tau A)\Sigma_k\Sigma_k^T \exp(\tau A^T)\,\mathrm{d}\tau$, the entries of Q_k is given by

$$\begin{aligned}
(Q_k)_{ij} &= \left(\int_0^{\Delta_k} \exp(\tau(a_i + a_j)\,\mathrm{d}\tau\right)(\Sigma_k\Sigma_k^T)_{ij}\\
&= \begin{cases} \frac{\exp((a_i+a_j)\Delta_k)-1}{a_i+a_j}(\Sigma_k\Sigma_k^T)_{ij} & \text{if } a_i + a_j \neq 0;\\ \Delta_k(\Sigma_k\Sigma_k^T)_{ij} & \text{if } a_i + a_j = 0. \end{cases}
\end{aligned}$$

Moreover, if $\Sigma_k = \operatorname{diag}(\sigma_{k,1}, \ldots, \sigma_{k,n})$, then Q_k is diagonal with

$$(Q_k)_{ii} = \begin{cases} \frac{\exp(2a_i\Delta_k)-1}{2a_i}\sigma_{k,i}^2 & \text{if } a_i \neq 0;\\ \Delta_k\sigma_{k,i}^2 & \text{if } a_i = 0. \end{cases}$$

Proof. Let $A = \operatorname{diag}(a_1, \ldots, a_n)$. Then

$$\bar{A}_k = \exp(\Delta_k A) = \operatorname{diag}\big(e^{a_1\Delta_k}, \ldots, e^{a_n\Delta_k}\big),$$

because the matrix exponential preserves diagonality and acts entry-wise on the diagonal. Next,

$$\bar{B}_k = \Big(\int_0^{\Delta_k} e^{\tau A}\,\mathrm{d}\tau\Big) B(x_{t_k}).$$

Since $e^{\tau A} = \operatorname{diag}(e^{a_1\tau}, \ldots, e^{a_n\tau})$, the integral is diagonal and computed componentwise:

$$\int_0^{\Delta_k} e^{a_i\tau}\,\mathrm{d}\tau = \begin{cases} \dfrac{e^{a_i\Delta_k} - 1}{a_i}, & a_i \neq 0,\\ \Delta_k, & a_i = 0. \end{cases}$$

Hence, we have

$$\bar{B}_k = \operatorname{diag}\Big(\frac{e^{a_1\Delta_k} - 1}{a_1}, \ldots, \frac{e^{a_n\Delta_k} - 1}{a_n}\Big) B(x_{t_k}).$$

For $Q_k = \int_0^{\Delta_k} e^{\tau A}\, \Sigma_k \Sigma_k^\top \, e^{\tau A^\top}\, \mathrm{d}\tau$, its (i,j)-th hentry equals

$$(Q_k)ij = \int_0^{\Delta_k} e^{(a_i+a_j)\tau}\, (\Sigma_k \Sigma_k^\top)ij \, \mathrm{d}\tau = \begin{cases} \dfrac{e^{(a_i+a_j)\Delta_k} - 1}{a_i + a_j} \, (\Sigma_k \Sigma_k^\top)ij, & a_i + a_j \neq 0, \\ \Delta_k \, (\Sigma_k \Sigma_k^\top)ij, & a_i + a_j = 0. \end{cases}$$

If moreover $\Sigma_k = \mathrm{diag}(\sigma_{k,1}, \ldots, \sigma_{k,n})$, then $(\Sigma_k \Sigma_k^\top)ij = \sigma k, i^2 \, \mathbf{1}\{i = j\}$, so

$$(Q_k)ii = \begin{cases} \dfrac{e^{2a_i \Delta_k} - 1}{2a_i} \, \sigma_{k,i}^2, & a_i \neq 0, \\ \Delta_k \, \sigma_{k,i}^2, & a_i = 0, \end{cases}$$

and off-diagonals vanish. □

Theorem 4.4.1 (Exact inference for Probabilistic Mamba) Assume the initial state belief $h_0 = \xi$ is Gaussian, with $h_0 \sim \mathcal{N}(\mu_{0|0}, P_{0|0})$. Then for all time steps $k \geq 0$,

(i) The filtering posterior $p(h_k \mid y_{1:k})$ is Gaussian;

(ii) The posterior parameters $(\hat{h}_{k|k}, P_{k|k})$ can be computed recursively; and

(iii) Given $(\hat{h}_{k|k}, P_{k|k})$, the parameters at step $k+1$ are obtained by the two–step prediction-update procedure:

- Prediction:
$$\hat{h}_{k+1|k} = \bar{A}_k \hat{h}_{k|k} + \bar{B}_k x_k, \quad P_{k+1|k} = \bar{A}_k P_{k|k} \bar{A}_k^T + Q_k$$

- Innovation:
$$e_{k+1} = y_{k+1} - C_{k+1} \hat{h}_{k+1|k}, \quad S_{k+1} = C_{k+1} P_{k+1|k} C_{k+1}^T + R_{k+1}$$

- Gain:
$$K_{k+1} = P_{k+1|k} C_{k+1}^T S_{k+1}^{-1}$$

- Update:
$$\begin{aligned} \hat{h}_{k+1|k} &= \hat{h}_{k+1|k} + K_{k+1} e_{k+1} \\ P_{k+1|k+1} &= (I - K_{k+1} C_{k+1}) P_{k+1|k} (I - K_{k+1} C_{k+1})^T + K_{k+1} R_{k+1} K_{k+1}^\top \end{aligned}$$

Moreover, conditional on past outputs and the inputs, the one-step predictive distribution is

$$y_k | y_1, \ldots, y_{k-1}, x_1, \ldots, x_k, \theta \ \sim \ \mathcal{N}(C_{k+1} \hat{h}_{k+1|k}, S_{k+1}),$$

so the conditional marginal log-likelihood of the outputs (by marginalising the latent states $h_1, \ldots, h_T$ out) is given by

$$\log[p(y_1, \ldots, y_T) | x_1, \ldots, x_T] = -\frac{1}{2} \sum_{k=0}^{T-1} [\log \det S_{k+1} + e_{k+1}^T S_{k+1}^{-1} e_{k+1} + d_y \log(2\pi)]$$

Proof. We proceed by induction on k.

Base case. For $k = 0$, by assumption $p(h_0) \sim \mathcal{N}(\mu_{0|0}, P_{0|0})$, which is Gaussian.

Assume at some step $k \in \mathbb{Z}^+$ the filtering distribution $p(h_k|y_1, \ldots, y_k) \sim \mathcal{N}(\hat{\mu}_{k|k}, P_{k|k})$ is Gaussian. Then at step $k+1$,

- Predictive step: The predictive distribution is

$$p(h_{k+1}|y_1, \ldots, y_k) = \int p(h_{k+1}|h_k)p(h_k|y_1, \ldots, y_k)\, \mathrm{d}h_k$$

  , where $p(h_{k+1}|h_k) \sim \mathcal{N}(\bar{A}_k h_k + \bar{B}_k, Q_k)$ by definition of LGSSM. Since this integral is a linear transformation of a Gaussian plus Gaussian noise, the result is Gaussian by Fact 1:

$$p(h_{k+1}|y_1, \ldots, y_k) = \mathcal{N}(\hat{\mu}_{k+1|k}, P_{k+1|k})$$

  with $\hat{\mu}_{k+1|k} = \bar{A}_k \hat{\mu}_{k|k} + \bar{B}_k, \quad P_{k+1|k} = \bar{A}_k P_{k|k} \bar{A}_k^T + Q_k.$

- Update step: The filtering distribution is obtained via Bayes' rule:

$$p(h_{k+1}|y_1 \ldots, y_{k+1}) \propto p(y_{k+1}|h_{k+1})p(h_{k+1}|y_1, \ldots, y_k),$$

  where $p(y_{k+1}|h_{k+1})$ is the Gaussian likelihood. By Fact 3, the product of two Gaussians is Gaussian (up to normalisation), and the normalisation constant is the marginal likelihood:

$$p(h_{k+1}|y_1, \ldots, y_k + 1) = \mathcal{N}(\hat{mu}_{k+1|k+1|}, P_{k+1|k+1}),$$

  with Kalman gain $K_{k+1} = P_{k+1|k} C_{k+1}^T (C_{k+1} P_{k+1|k} C_{k+1}^T + R_{k+1})^{-1}$ and

$$\begin{aligned}\hat{\mu}_{k+1|k+1} &= \hat{\mu}_{k+1|k} + K_{k+1}(y_{k+1} - C_{k+1}\hat{\mu}_{k+1|k}),\\ P_{k+1|k+1} &= (I - K_{k+1}C_{k+1})P_{k+1|k}.\end{aligned}$$

By induction, all filtering and predictive distributions are Gaussian. The conditional marginal log-likelihood is given by $p(y_1, \ldots, y_T) = \prod_k p(y_k|y_1, \ldots, y_{k-1})$ □

Proposition 8.1.1 (Parameter count of simple RNN with linear output layer)**.** Consider a simple RNN implemented in PyTorch with input dimension I, hidden size H, number of layers L, and output dimension $O = 1$. Each RNN layer includes input-to-hidden and hidden-to-hidden weight matrices, together with two bias vectors. A linear output layer $\mathbb{R}^H \to \mathbb{R}^O$ is appended to produce the final prediction. The total number of trainable parameters is

$$\text{Number of parameters} = (2L-1)H^2 + (I + 2L + O)H + O.$$

Proof. For the first RNN layer ($\ell = 0$), PyTorch creates:

- an input-to-hidden weight matrix $W_{xh} \in \mathbb{R}^{H \times I}$ with HI parameters;
- a hidden-to-hidden weight matrix $W_{hh} \in \mathbb{R}^{H \times H}$ with H^2 parameters;
- two bias vectors $b_{ih}, b_{hh} \in \mathbb{R}^H$, contributing $2H$ parameters.

The parameter count for layer 0 is therefore $HI + H^2 + 2H$.

For each subsequent RNN layer $\ell = 1, \ldots, L-1$, the input dimension is H rather than I. Each such layer contributes

$$H^2 \text{ (input-to-hidden)} \; + \; H^2 \text{ (hidden-to-hidden)} \; + \; 2H \text{ (biases)} \; = \; 2H^2 + 2H.$$

Summing across all L layers gives

$$(HI + H^2 + 2H) + (L-1)(2H^2 + 2H) = HI + (2L-1)H^2 + 2LH.$$

Finally, the linear output layer $W_o \in \mathbb{R}^{O \times H}$ with bias $b_o \in \mathbb{R}^O$ contributes $OH + O$ parameters. Adding this term yields

$$\text{Number of parameters} = (2L-1)H^2 + (I + 2L + O)H + O,$$

as required. □

8.2 Data pre-processing pipeline results

Momentum correlation results

The following table shows the correlation between `mom`, `mom1`, `mom2`, `mom3` and k-day future returns, $k = \{1, 2, 3, 5, 10\}$:

Table 8.1: Correlation of momentum features with k-day future returns

Feature	**1-day**	**2-day**	**3-day**	**5-day**	**10-day**
			NYSE dataset		
`mom`	-0.9991	-0.6735	-0.5941	-0.4331	-0.3227
`mom1`	0.0891	-0.6736	-0.4901	-0.4094	-0.2682
`mom2`	-0.0893	-0.0042	-0.5941	-0.4665	-0.3243
`mom3`	0.0324	-0.0414	0.0167	-0.4103	-0.2786
			IXIC dataset		
`mom`	-0.9992	-0.6714	-0.5793	-0.4303	-0.3214
`mom1`	0.0967	-0.6715	-0.4903	-0.3943	-0.2567
`mom2`	-0.0533	0.0298	-0.5793	-0.4384	-0.3061
`mom3`	0.0366	-0.0117	0.0466	-0.3959	-0.2645

VIF Analysis results

The following are results from the VIF pruning on NYSE and IXIC datasets:

- NYSE:

Table 8.2: Features retained and dropped after VIF pruning (threshold = 10.0)

Retained features
Vol., weekday, ROC_5, ROC_10, ROC_15, ROC_20, EMA_10, WTI-oil, FTSE-F, HSI-F, Gold-F, NZD, Brent, DBAA, XAU, AUD, AMZN, RUSSELL-F, CNY, MSFT, silver-F, CAD, DAX-F, DJI-F, XAG, XOM, EUR, WFC, Dollar Index-F, GE, copper-F, RUT, JPM, GAS-F, JPY, wheat-F, GBP, SSEC, Nikkei-F, CHF, KOSPI-F, AAPL, CAC-F, NASDAQ-F, JNJ, TE1, DE1, DE2

Dropped features (VIF greater than 10)
TE6, DTB6, DE4, TE5, DTB4WK, DAAA, DGS10, DE5, DTB3, DE6, EMA_20, CTB6M, CTB3M, EMA_50, CTB1Y, TE2, GSPC, DGS5, S&P-F, FCHI, EMA_200, GDAXI, oil, TE3, IXIC, HSI, FTSE, Dollar Index, DJI

- IXIC:

Table 8.3: Features retained and dropped after VIF pruning (threshold = 10.0)

Retained features
Vol., weekday, ROC_5, ROC_10, ROC_15, ROC_20, EMA_10, WTI-oil, FTSE-F, HSI-F, Gold-F, NZD, Brent, DBAA, XAU, AUD, AMZN, RUSSELL-F, CNY, MSFT, silver-F, CAD, DAX-F, DJI-F, XAG, XOM, EUR, WFC, Dollar Index-F, GE, copper-F, RUT, JPM, GAS-F, JPY, wheat-F, GBP, SSEC, Nikkei-F, CHF, KOSPI-F, AAPL, CAC-F, NASDAQ-F, JNJ, TE1, TE5, DE1, DE2

Dropped features (VIF greater than 10)
DAAA, DTB6, DTB4WK, DGS10, TE3, DE4, TE2, DE5, DTB3, DE6, EMA_20, CTB6M, CTB3M, EMA_50, CTB1Y, EMA_200, DGS5, S&P-F, FCHI, GSPC, GDAXI, oil, NYSE, HSI, FTSE, Dollar Index, TE6, DJI

8.3 Measure Theoretic Probability and Stochastic Calculus

8.3.1 Measure-theoretic Probability

Measure theory provides a rigorous foundation for modern probability theory, offering a coherent framework for defining random variables and expectations. It also underpins the formulation and solutions of stochastic differential equations. This section is largely based on materials I learnt during my undergraduate studies in Mathematics and Statistics at the University of Warwick.

Definition 8.3.1 (σ-algebra)**.** Let Ω be a set. A collection $\mathcal{F}$ of subsets of Ω is called a $\boldsymbol{\sigma}$**-algebra** if

(i) $\Omega \in \mathcal{F}$;

(ii) $A \in \mathcal{F} \implies A^c = \Omega \setminus A \in \mathcal{F}$;

(iii) For all sequences of subsets $(A_n)_{n\in\mathbb{N}} \subseteq \mathcal{F}$, we have $\bigcup\limits_{n=1}^{\infty} A_n \in \mathcal{F}$.

Put shortly, a σ-algebra of subsets of Ω is a collection of sets that contain Ω and is closed under complements and countable unions.

Definition 8.3.2 (Probability measure)**.** Let $(\Omega, \mathcal{F})$ be a measurable space. A function $\mathbb{P} : \mathcal{F} \to [0,1]$ is called a **probability measure** if

(i) $\mathbb{P}(\varnothing) = 0$;

(ii) $\mathbb{P}(\Omega) = 1$;

(iii) (Countable additivity) For any sequence $(A_n)_{n\in\mathbb{N}}$ of *pairwise disjoint* elements of $\mathcal{F}$,

$$\mathbb{P}\left(\bigcup_{n=1}^{\infty} A_n\right) = \sum_{n=1}^{\infty} \mathbb{P}(A_n).$$

The triple $(\Omega, \mathcal{F}, \mathbb{P})$ is called a **probability space**.

Intuitively, a probability space can be interpreted as a model for a probabilistic experiment:

- Sample space Ω: This contains all points that can be chosen;
- Set of events $\mathcal{F}$: This contains subsets A of Ω where $\mathbb{P}(A)$ is known;
- Probability measure $\mathbb{P}$: $\mathbb{P}(\Omega) =$ Probability that a point in Ω is chosen $= 1$.

Definition 8.3.3 (Measurable function)**.** Let $(\Omega, \mathcal{F})$ and $(E, \mathcal{E})$ be measurable spaces. Then a function $f : (\Omega, \mathcal{F}) \to (E, \mathcal{E})$ is called **measurable** if

$$f^{-1}(A) \in \mathcal{F} \quad \text{for all } A \in \mathcal{E},$$

where $f^{-1}(A) = \{\omega \in \Omega : f(\omega) \in A\}$ is the pre-image of f.

An immediate example of a measurable function is a **random variable**.

Definition 8.3.4 (Random variable)**.** Let $(\Omega, \mathcal{F})$ and $(E, \mathcal{E})$ be measurable spaces. A **random variable** is a measurable function $X : (\Omega, \mathcal{F}) \to (E, \mathcal{E})$. A real-valued random variable is a random variable with $E = \mathbb{R}$, $\mathcal{E} = \mathcal{B}(\mathbb{R})$. If X is a random variable, the collection of subsets of Ω

$$\sigma(X) := \{X^{-1}(A) : A \in \mathcal{E}\} \subseteq \mathcal{F}$$

can be shown to be a σ-algebra, known as the **σ-algebra generated by** X. This represents the smallest σ-algebra that X is measurable.

Apart from the σ-algebra $\mathcal{F}$ that comes with the probability space $(\Omega, \mathcal{F}, \mathbb{P})$, one often consider sub σ-algebras contained in $\mathcal{F}$, for example $\sigma(X)$. This is because σ-algebras serve to model *information* or *knowledge*, as illustrated in the following example ([34]):

Example 8.3.1. Consider an experiment of tossing a coin and then rolling a die. Take the sample space as

$$\Omega = \left\{ \begin{array}{cccccc} (\mathrm{H},1), & (\mathrm{H},2), & (\mathrm{H},3), & (\mathrm{H},4), & (\mathrm{H},5), & (\mathrm{H},6), \\ (\mathrm{T},1), & (\mathrm{T},2), & (\mathrm{T},3), & (\mathrm{T},4), & (\mathrm{T},5), & (\mathrm{T},6) \end{array} \right\},$$

the σ-algebra $\mathcal{F}$ as the power set of Ω, 2^Ω, and the probability measure as

$$\mathbb{P}(A) = \frac{|A|}{12} \qquad \text{for each } A \in \mathcal{F}.$$

Imagine an observer who can see the result of the coin toss, but not the die roll. Then, this observer is not able to identify precisely which outcome ω has occurred, but they can still say that certain events have, or have not occurred. For example, they will be able to identify whether the event

$$\{(\mathrm{H},1), (\mathrm{H},2), (\mathrm{H},3), (\mathrm{H},4), (\mathrm{H},5), (\mathrm{H},6)\}$$

or

$$\{(\mathrm{T},1), (\mathrm{T},2), (\mathrm{T},3), (\mathrm{T},4), (\mathrm{T},5), (\mathrm{T},6)\}$$

has occurred, because this only requires knowledge about the coin toss. On the other hand, the observer will not be able to tell whether the event

$$\{(\mathrm{H},1), (\mathrm{H},2), (\mathrm{H},3)\}$$

has occurred, since it would require knowledge about the die roll. For the observer, the collection of events that are resolved is

$$\mathcal{G} = \{\varnothing, \Omega, \{(\mathrm{H},1), (\mathrm{H},2), (\mathrm{H},3), (\mathrm{H},4), (\mathrm{H},5), (\mathrm{H},6)\}, \{(\mathrm{T},1), (\mathrm{T},2), (\mathrm{T},3), (\mathrm{T},4), (\mathrm{T},5), (\mathrm{T},6)\}\}.$$

This collection $\mathcal{G}$ is a sub-σ-algebra of $\mathcal{F}$. In particular, for $c = \{H, T\}$, $d = \{1 \ldots, 6\}$ and $\omega = (c, d)$, if we define the random variable $X : \Omega \to \{H, T\}$, where $X(\omega) = c$, then $\mathcal{G} = \sigma(X)$, since

$$\sigma(X) = \{X^{-1}(H), X^{-1}(T), \Omega, \varnothing\} = \mathcal{G}.$$

The above example illustrates how partial knowledge about the experimental outcome implies a σ-algebra contained in $\mathcal{F}$. Conversely, given a σ-algebra $\mathcal{G} \subseteq \mathcal{F}$, one can imagine an observer with partial knowledge: this observer may not be able to say exactly which outcome $\omega \in \Omega$ has occurred, but for each set $A \in \mathcal{G}$, they are able to say whether or not A has occurred.

Definition 8.3.5 (Filtration)**.** Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space, and let $\mathcal{T}$ be an index set (e.g., $\mathcal{T} = \{1, 2, 3, \ldots\}$ or $\mathcal{T} = [0, \infty)$). A sequence of sub σ-algebras $(\mathcal{F}_t)_{t \in \mathcal{T}}$ is called a **filtration** if

$$\mathcal{F}_s \subseteq \mathcal{F}_t \subseteq \mathcal{F} \quad \text{for all } s \leq t,\ s, t \in \mathcal{T}.$$

In the discrete-time setting where $\mathcal{T} = \{1, 2, 3, \ldots\}$, this is also equivalent to

$$\mathcal{F}_t \subseteq \mathcal{F}_{t+1} \subseteq \mathcal{F} \quad \text{for all } t \in \mathcal{T}.$$

Intuitively, $\mathcal{F}_t$ represents all the information available up to time t, and your information accumulates as time progresses. A filtration ensures that this flow of information never decreases: as time advances, you may learn more, but never lose knowledge you already had. We call $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\in\mathcal{T}}, \mathbb{P})$ a **filtered probability space**.

Definition 8.3.6 (Adapted process)**.** Let $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\in\mathcal{T}}, \mathbb{P})$ be a filtered probability space. A stochastic process $(X_t)_{t\in\mathcal{T}}$ is said to be **adapted** to the filtration $(\mathcal{F}_t)_{t\in\mathcal{T}}$ if each X_t is $\mathcal{F}_t$-measurable.

Adaptivity is essential in stochastic processes since it formalises the principle that your model or strategy cannot “see into the future.”

Definition 8.3.7 (Conditional Expectation)**.** Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space, $\mathcal{G} \subseteq \mathcal{F}$ be a sub σ-algebra and X be an integrable random variable (i.e. $\mathbb{E}[|X|] < \infty$). The **conditional expectation** of X given $\mathcal{G}$ is a random variable $\mathbb{E}(X|\mathcal{G})$ satisfying:

(i) (Measurability) $\mathbb{E}[X|\mathcal{G}]$ is $\mathcal{G}$-measurable;

(ii) (Partial averaging) $\mathbb{E}[\mathbb{E}[X|\mathcal{G}] \cdot \mathbf{1}_A] = \mathbb{E}[X \cdot \mathbf{1}_A]$ for all $A \in \mathcal{G}$.

When $\mathcal{G} = \sigma(Y)$ for some random variable Y, we write $\mathbb{E}[X|Y)]$ instead of $\mathbb{E}[X|\sigma(Y)]$.

Intuitively, $\mathbb{E}[X|\mathcal{G}]$ represents the best guess for X given the information $\mathcal{G}$ we have so far: Measurability ensures this guess only uses information from $\mathcal{G}$, while partial averaging ensures that this is the best guess of X given $\mathcal{G}$.

Definition 8.3.8 (Martingale)**.** Let $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\in\mathcal{T}}, \mathbb{P})$ be a filtered probability space. A stochastic process $(M_t)_{t\in\mathcal{T}}$ is said to be a **martingale** (with respect to $(\mathcal{F}_t)_{t\in\mathcal{T}}$ and $\mathbb{P}$) if

(i) (Adaptivity) $(M_t)_{t\in\mathcal{T}}$ is adapted to $(\mathcal{F}_t)_{t\in\mathcal{T}}$;

(ii) (Integrability) $\mathbb{E}[|M_t|] < \infty$ for all $t \in \mathcal{T}$;

(iii) (Fair-game) $\mathbb{E}[M_t \mid \mathcal{F}_s] = M_s$ almost surely for all $0 \leq s \leq t$.

If $(M_t)_{t\in\mathcal{T}}$ is a discrete-time process, then (iii) is equivalent to $\mathbb{E}[M_{t+1} \mid \mathcal{F}_t] = M_t$ almost surely.

Intuitively, a martingale models a “fair game”: given all information up to time t, your best prediction for the next value at time $t + 1$ is simply the current value — there is no predictable trend or bias in either direction. In quantitative finance, discounted asset prices are traditionally modelled as a martingale under the risk-neutral measure in a no-arbitrage setting.

8.3.2 Brownian Motion and Stochastic Calculus

This section is largely based on Chapters 2 and 3 of [13] as well as [34]. No details derivations or proofs are provided here.

An important class of stochastic process that is central to the theory of stochastic differential equations is **Brownian Motion**, which is also known as the **Weiner process** in mathematical literature.

Definition 8.3.9 (Brownian Motion)**.** Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space. A continuous-time stochastic process $(W_t)_{t\geq 0}$ on $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$ is a **Brownian Motion** if

(i) (Continuous sample paths) For each $\omega \in \Omega$, there exists a continuous function $t \mapsto W_t$ for all $t \geq 0$ (which we call a *sample path*) such that $W_0 = 0$;

(ii) (Independent increments) The increment $W_t - W_s$ is independent of $(W_r)_{r \leq s}$;

(iii) (Gaussian increments) $W_t - W_s \sim \mathcal{N}(0, t-s)$.

Intuitively, we can interpret each properties of Brownian Motion as follows:

- Continuous sample paths: The trajectory of a Brownian motion evolves smoothly over time without jumps;
- Independent increments: The increment $W_t - W_s$ depends only on the time interval $[s, t]$, not on how the process behaved before s;
- Gaussian increments: This is a consequence of the Central Limit Theorem when viewing Brownian Motion as the continuous limit of a scaled symmetric random walk.

Definition 8.3.10 (Augmented natural filtration for Brownian Motion)**.** Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space on which a Brownian Motion $(W_t)_{t\geq 0}$ is defined. An **augmented filtration for a Brownian Motion** is a filtration that is generated by the Brownian motion and is right-continuous and complete, i.e. all null sets are measurable at all times.

This definition is quite subtle but is required to define any stochastic integral which we will define:

Definition 8.3.11 (Stochastic (Itô) integral)**.** Let $(W_t)_{t\geq 0}$ be a standard Brownian motion on a filtered probability space $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\geq 0}, \mathbb{P})$ where $(\mathcal{F}_t)_{t\geq 0}$ is the augmented natural filtration for Brownian motion. A process $H = (H_t)_{t\geq 0}$ is called *admissible* if it is progressively measurable with respect to $(\mathcal{F}_t)$ and satisfies

$$\mathbb{E}\left[\int_0^T H_t^2 \, dt\right] < \infty \quad \text{for all } T > 0.$$

For such H, the stochastic integral of H with respect to W is the process

$$I_t = \int_0^t H_s \, dW_s, \qquad t \geq 0,$$

defined first for simple (piecewise constant) processes by

$$\int_0^t H_s \, dW_s := \sum_{k=0}^{n-1} H_{t_k} \left(W_{t_{k+1}\wedge t} - W_{t_k \wedge t}\right),$$

and then extended in $L^2(\Omega)$ to all admissible H. The Itô integral $I = (I_t)$ is a square-integrable martingale with

$$\mathbb{E}[I_t] = 0, \qquad \mathbb{E}[I_t^2] = \mathbb{E}\left[\int_0^t H_s^2 \, ds\right].$$

We also state 2 results that we used in Chapter 4.3:

Definition 8.3.12 (Itô's product rule.)**.** For two Itô processes X_t and Y_t,

$$d(X_tY_t) \;=\; X_t\,dY_t \;+\; Y_t\,dX_t \;+\; d\langle X,Y\rangle_t,$$

where $\langle X,Y\rangle_t$ denotes the quadratic covariation. In particular, if

$$dX_t = \mu_t^X\,dt + \sigma_t^X\,dW_t, \qquad dY_t = \mu_t^Y\,dt + \sigma_t^Y\,dW_t,$$

then

$$d(X_tY_t) = (X_t\mu_t^Y + Y_t\mu_t^X + \sigma_t^X\sigma_t^Y)\,dt + (X_t\sigma_t^Y + Y_t\sigma_t^X)\,dW_t.$$

Definition 8.3.13 (Itô's isometry.)**.** If $H = (H_t)_{t\geq 0}$ is a square-integrable, progressively measurable process, then the Itô integral satisfies

$$\mathbb{E}\left[\left(\int_0^T H_t\,dW_t\right)^2\right] = \mathbb{E}\left[\int_0^T H_t^2\,dt\right].$$

More generally, for $0 \leq s < t$,

$$\mathbb{E}\left[\left(\int_s^t H_\tau\,dW_\tau\right)^2 \,\middle|\, \mathcal{F}_s\right] = \mathbb{E}\left[\int_s^t H_\tau^2\,d\tau \,\middle|\, \mathcal{F}_s\right].$$

8.4 Usage of Generative AI

Generative AI is used in this project for the following:

- Assisting in the proofs of theorems and corollaries, especially Theorem 4.3.1;
- Assisting in implementation of models and helper functions, in Python, especially translating Prob-Mamba into a working pipeline; and
- Proofreading and polishing the presentation of my work.

Bibliography

[1] Philipp Becker, Niklas Freymuth, and Gerhard Neumann. 2024. KalMamba: Towards Efficient Probabilistic State Space Models for RL under Uncertainty. arXiv:2406.15131 [cs.LG] `https://arxiv.org/abs/2406.15131`

[2] Tim Bollerslev. 1986. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics* 31, 3 (April 1986), 307–327. `https://doi.org/None`

[3] G.E.P. Box, G.M. Jenkins, G.C. Reinsel, and G.M. Ljung. 2015. *Time Series Analysis: Forecasting and Control*. Wiley. `https://books.google.co.uk/books?id=rNt5CgAAQBAJ`

[4] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. 2019. Neural Ordinary Differential Equations. arXiv:1806.07366 [cs.LG] `https://arxiv.org/abs/1806.07366`

[5] Rama Cont. 2005. Volatility Clustering in Financial Markets: Empirical Facts and Agent-Based Models. *Long Memory in Economics* 1 (05 2005). `https://doi.org/10.2139/ssrn.1411462`

[6] Robert Engle. 2001. GARCH 101: The Use of ARCH/GARCH Models in Applied Econometrics. *Journal of Economic Perspectives* 15, 4 (December 2001), 157–168. `https://doi.org/10.1257/jep.15.4.157`

[7] Robert F Engle. 1982. Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation. *Econometrica* 50, 4 (July 1982), 987–1007. `https://doi.org/None`

[8] Nick Firoozye. 2025. Time Series Statistics. Lecture slides, COMP0051 Algorthmic Trading, University College London.

[9] Albert Gu and Tri Dao. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752* (December 2023).

[10] Albert Gu, Karan Goel, and Christopher Ré. 2022. Efficiently Modeling Long Sequences with Structured State Spaces. arXiv:2111.00396 [cs.LG] `https://arxiv.org/abs/2111.00396`

[11] Ali Hatamizadeh and Jan Kautz. 2025. MambaVision: A Hybrid Mamba-Transformer Vision Backbone. arXiv:2407.08083 [cs.CV] `https://arxiv.org/abs/2407.08083`

[12] Rudolf E. Kalman. 1960. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering* 82, 1 (1960), 35–45. https://doi.org/10.1115/1.3662552

[13] Ioannis Karatzas and Steven E. Shreve. 1991. *Brownian Motion and Stochastic Calculus* (second ed.). Graduate Texts in Mathematics, Vol. 113. Springer-Verlag, New York. xxiv+470 pages. https://doi.org/10.1007/978-1-4612-0949-2

[14] Feyza Duman Keles, Pruthuvi Mahesakya Wijewardena, and Chinmay Hegde. 2022. On The Computational Complexity of Self-Attention. arXiv:2209.04881 [cs.LG] https://arxiv.org/abs/2209.04881

[15] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

[16] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG] https://arxiv.org/abs/1412.6980

[17] Xuanqing Liu, Tesi Xiao, Si Si, Qin Cao, Sanjiv Kumar, and Cho-Jui Hsieh. 2019. Neural SDE: Stabilizing Neural ODE Networks with Stochastic Noise. arXiv:1906.02355 [cs.LG] https://arxiv.org/abs/1906.02355

[18] Ali Mehrabian, Ehsan Hoseinzade, Mahdi Mazloum, and Xiaohong Chen. 2024. Mamba Meets Financial Markets: A Graph-Mamba Approach for Stock Price Prediction. *arXiv preprint arXiv:2410.03707* (2024).

[19] Jakub Michańków. 2025. Forecasting Probability Distributions of Financial Returns with Deep Neural Networks. arXiv:2508.18921 [q-fin.RM] https://arxiv.org/abs/2508.18921

[20] S Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf. Accessed:2025-09-05.

[21] NASDAQ. 2023. NASDAQ Composite Index Methodology. https://www.nasdaq.com/. Accessed: 2025-09-05.

[22] New York Stock Exchange. 2023. NYSE Composite Index. https://www.nyse.com/. Accessed: 2023-09-05.

[23] NumPy Developers. 2024. numpy.expm1 — NumPy v2.2 Manual. https://numpy.org/doc/2.2/reference/generated/numpy.expm1.html Accessed: 2025-09-06.

[24] Katsuhiko Ogata. 1987. *Discrete-Time Control Systems.* Prentice Hall, Australia, Sydney.

[25] Peter Orbanz. 2024. Probabilistic and Unsupervised Learning. Lecture slides, COMP0086 Probabilistic and Unsupervised Learning, University College London. https://www.gatsby.ucl.ac.uk/teaching/courses/ml1/slides_COMP0086.pdf

[26] Badri N. Patro and Vijay S. Agneeswaran. 2024. SiMBA: Simplified Mamba-Based Architecture for Vision and Multivariate Time series. arXiv:2403.15360 [cs.CV] https://arxiv.org/abs/2403.15360

[27] William Peebles and Saining Xie. 2023. Scalable Diffusion Models with Transformers. arXiv:2212.09748 [cs.CV] `https://arxiv.org/abs/2212.09748`

[28] Pedro Pessoa, Paul Campitelli, Douglas P Shepherd, S Banu Ozkan, and Steve Pressé. 2025. Mamba time series forecasting with uncertainty quantification. *Machine Learning: Science and Technology* 6, 3 (July 2025), 035012. `https://doi.org/10.1088/2632-2153/adec3b`

[29] K. B. Petersen and M. S. Pedersen. 2008. The Matrix Cookbook. `http://www2.imm.dtu.dk/pubdb/p.php?3274` Version 20081110.

[30] Alec Radford and Karthik Narasimhan. 2018. Improving Language Understanding by Generative Pre-Training. `https://api.semanticscholar.org/CorpusID:49313245`

[31] Liliang Ren, Yang Liu, Yadong Lu, Yelong Shen, Chen Liang, and Weizhu Chen. 2024. Samba: Simple Hybrid State Space Models for Efficient Unlimited Context Language Modeling. *arXiv preprint* (2024). `https://arxiv.org/abs/2406.07522`

[32] Zhuangwei Shi. 2024. MambaStock: Selective state space model for stock prediction. *arXiv preprint arXiv:2402.18959* (February 2024). Available at: `https://arxiv.org/abs/2402.18959`.

[33] Elke Thönnes. 2023. ST221 Linear Statistical Modelling. Lecture notes, ST221 Linear Statistical Modelling, University of Warwick.

[34] Daniel Valesin. 2024. ST401 Stochastic Methods in Finance. Lecture notes, ST401 Stochastic Methods in Finance, University of Warwick.

[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *CoRR* abs/1706.03762 (2017). arXiv:1706.03762 `http://arxiv.org/abs/1706.03762`

[36] Steven Y. K. Wong, Jennifer S. K. Chan, and Lamiae Azizi. 2024. Quantifying neural network uncertainty under volatility clustering. arXiv:2402.14476 [q-fin.ST] `https://arxiv.org/abs/2402.14476`

[37] Hongyu Zhang, Jian Li, Sheng Wang, and Yang Liu. 2025. FinMamba: Market-Aware Graph Enhanced Multi-Level Mamba for Stock Movement Prediction. *arXiv preprint arXiv:2502.06707* (January 2025). Available at: `https://arxiv.org/abs/2502.06707`.