Machine Learning Applications in Empirical Finance: Volatility Modeling and Forecasting

Doctoral Thesis

by

German Rodikov

Doctoral Program in Data Science

Supervisor

Nino Antulov-Fantulin, ETH Zürich, Switzerland

Advisor

Fabrizio Lillo, Scuola Normale Superiore, University of Bologna, Italy

Machine Learning Applications in Empirical Finance: Volatility Modeling and Forecasting

by

German Rodikov

September 4^{th}, 2023

Submitted to the Scuola Normale Superiore di Pisa in partial fulfillment of the requirements for the Doctoral Program in Data Science

Abstract

Predicting volatility in financial markets is crucial for assessing financial risks. While deep learning has significantly progressed, neural networks often require additional features to outperform traditional econometric models for volatility prediction. It could be due to complexities such as market noise, microstructure, heteroscedasticity, news effects, and multiple time scales. Although econometric models for price volatility have evolved considerably over the years, the potential for integrating these models with deep learning techniques, particularly Recurrent Neural Networks (RNNs), still needs to be explored.

In this research, we investigate the performance of Long Short-Term Memory (LSTM) RNNs in predicting volatility and benchmark performance against established econometric models. Specifically, we investigate the impact of hyperparameter optimization, focusing on input dimension and LSTM architecture.

We introduce a novel RNN cell design called the σ-Cell to address the challenges of volatility modeling within deep learning. This design incorporates domain-specific knowledge and time-varying parameters, resulting in a generative network that captures the joint distribution of the stochastic volatility process. It also provides an approximation of the conditional distribution of latent variables. We employ a log-likelihood-based loss function to optimize the model and introduce a specialized Adjusted-Softplus activation function.

Furthermore, we propose a new LSTM cell variant, σ-LSTM, including a stochastic processing layer. By embedding stylized facts related to volatility as a form of inductive bias, we improve the model's predictive accuracy by utilizing a probabilistic loss function.

Our research underscores the value of integrating domain knowledge with deep learning techniques for more accurate volatility prediction in financial markets. It also emphasizes the importance of meticulous hyperparameter tuning, particularly concerning input dimension and architecture.

Publications

Journal Articles

- Can LSTM outperform volatility-econometric models? German Rodikov, Nino Antulov-Fantulin, *https://arxiv.org/abs/2202.11581* (to appear in the Proceedings of MDPI, International Conference on Time Series and Forecasting 2023)

- Introducing the σ-Cell: Unifying GARCH, Stochastic Fluctuations and Evolving Mechanisms in RNN-based Volatility Forecasting, German Rodikov, Nino Antulov-Fantulin, *https://arxiv.org/abs/2309.01565* (under review, Journal of Financial Econometrics — Oxford Academic) German Rodikov and Nino Antulov-Fantulin

- Volatility-inspired σ-LSTM cell German Rodikov, Nino Antulov-Fantulin *https://arxiv.org/abs/2202.11581* (selected, to appear in the Springer, Contributions to Statistics 2023)

Conferences

- DCP 2022 DYNAMICS AND COMPLEXITY, Pisa (Italy), 26-28 May 2022.

- AMASES XLVI Italian Association for Mathematics Applied to Social and Economic Sciences, Palermo (Italy), 22-24 September 2022.

- ITISE 2023 9th International Conference on Time Series and Forecasting. Gran Canaria (Spain), 12-14 July 2023.

Acknowledgments

I am profoundly grateful to my supervisor, Nino Antulov-Fantulin, for his unwavering guidance, support, and encouragement throughout this journey. I also extend my sincere gratitude to my advisor, Fabrizio Lillo, whose insights and expertise have been invaluable in bringing this work to fruition.

My heartfelt thanks go to Dino Pedreschi, the coordinator of the Data Science PhD program, for providing me with the opportunity to be part of this excellent program and for fostering an environment that made research collaboration and access to essential resources possible.

I would also like to extend my appreciation to the other Data Science PhD board members and those affiliated with the program who supported my research endeavors: Anna Monreale, Francesca Chiaromonte, and Silvia Zappulla.

Lastly, I owe immense gratitude to my family, who provided me with unwavering care, warmth, and support throughout this journey and whose belief in me sustained me every step of the way.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Volatility transcends its role as a statistical metric. Volatility holds an integral role in financial markets, shaping risk environments, informing trading strategies, and influencing decision-making. Consequently, volatility has emerged as a foundational construct in finance, garnering extensive research attention.

Volatility is a fundamental concept in finance that quantifies the extent of price fluctuations for a financial asset. It acts as a crucial proxy for risk in financial markets. High volatility often indicates elevated risk and substantial price changes, whereas low volatility suggests more stable market conditions Poon and Granger [2003]. Given that the pricing of derivative financial instruments is directly related to the implied volatility of the underlying assets, understanding volatility is essential for the valuation and trading of these instruments Gatheral [2006]. Furthermore, the derivatives market has seen significant growth in recent decades, making the modeling and forecasting of volatility increasingly vital for both researchers and practitioners Figlewski [2008].

Early stochastic models for price changes, such as Brownian motion and the Wiener process, provided foundational insights into asset price volatility. However, these models are limited in their ability to capture all the empirical nuances of price fluctuations Francq and Zakoian [2010]. More contemporary models, like those in the Autoregressive Conditional Heteroscedasticity (ARCH) family, aim to characterize volatility through a conditional process Engle [1982b]. In these models, the conditional variance fluctuates over time, while the unconditional variance remains relatively stable Francq and Zakoian [2010]. Despite their contributions, traditional volatility models face challenges,

including reliance on strong assumptions and the inherent difficulty of directly observing volatility.

In response to these limitations, researchers have explored alternative methods for modeling and forecasting volatility. One such method is realized volatility (RV), which estimates the integrated variance of a stochastic process over a given timeframe, offering a more precise measure of time-varying volatility Andersen et al. [2003]. Realized volatility is particularly beneficial for capturing the cumulative variability in stock price dynamics and facilitating the development of more advanced financial models that account for the complexities of market dynamics Corsi et al. [2012].

Despite advancements in volatility modeling, traditional methods often rely on strong assumptions and encounter difficulties capturing the intricate temporal structures inherent in financial markets. Moreover, these approaches grapple with the challenge of directly observing volatility, which is often latent and unobservable Cavalcante et al. [2016].

Machine learning techniques, particularly Neural Networks (NNs), have been introduced to address these challenges in recent years. While NNs hold the potential to learn complex, non-linear relationships in data, their application in volatility forecasting has produced mixed results Krauss et al. [2017]. Among the various neural network architectures, RNNs have effectively handled sequential data Fischer and Krauss [2018]. LSTM cells, a specialized variant of RNNs, mitigate the issue of diminishing memory for early inputs through gated mechanisms that regulate information flow Hochreiter and Schmidhuber [1997a]. However, LSTM models have the drawback of requiring the tuning of a large number of training parameters, thereby increasing computational costs and complicating hyperparameter optimization for long sequences Greff et al. [2016].

Despite the increasing adoption of machine learning techniques, including NNs, these methods have yet to improve volatility prediction substantially. This shortfall represents a critical gap in machine learning and financial literature, raising the question: Can machine learning techniques be effectively tailored to capture the nuances of financial volatility?

Addressing this question, our study investigates:

- Can LSTM-based RNNs capture the temporal dynamics of realized volatility?
- How does the optimized LSTM model perform compared to established Heterogeneous Autoregressive (HAR) and Generalized Autoregressive Conditional Heteroskedasticity (GARCH) models in forecasting realized volatility?
- Is it possible to enhance the performance of volatility prediction tasks by integrating RNNs with domain-specific knowledge?
- Can the addition of time-varying parameters improve the performance of these models in volatility prediction tasks?
- Can a modified LSTM RNN cell, enriched with domain knowledge, enhance the performance of machine learning techniques in volatility prediction?

Chapter 2 provides a comprehensive literature review on volatility modeling, covering various approaches such as deterministic, probabilistic, and hybrid models. We also present essential background information on Recurrent Neural Networks, their advanced cell types, and approaches for assessing model performance.

Chapter 3 investigates how Neural Networks, specifically Long Short-Term Memory LSTM-based RNN, can effectively capture the temporal structure of realized volatility. We explore using LSTM models with hyperparameter optimization and specific input dimension to forecast realized volatility. The chapter demonstrates neural networks' effectiveness in forecasting volatility by comparing their predictive capabilities with established models such as HAR and GARCH.

In Chapter 4, we introduce the σ-Cell type models, a specialized design of RNN cells. We discuss how the σ-Cell integrates principles from GARCH, incorporates a stochastic layer, and employs time-varying parameters to address the dynamic nature of financial volatility. We show that this combination creates a generative network that captures the joint distribution of the stochastic volatility process while approximating the conditional distribution of latent variables given the observables. We further describe how we utilize a log-likelihood-based loss function and develop a specialized Adjusted-Softplus activation function to enhance the model's efficacy.

Chapter 5 introduces the σ-LSTM approach, a novel method that combines the advantages of LSTMs with established methodologies in volatility modeling. We elucidate how the σ-LSTM incorporates concepts from the σ-Cell model and the theoretical foundations of latent stochastic processes within an LSTM-based neural network framework. We assess how this new approach addresses the limitations of traditional volatility models by leveraging the capabilities of machine learning and neural networks to capture the temporal structure of realized volatility more effectively.

Chapter 6 presents recommendations for future research, while Chapter 7 concludes the study.

The significance of this research lies in its potential to enhance the modeling and forecasting of financial market volatility. This study aims to offer a nuanced understanding of volatility dynamics by employing machine learning techniques, specifically neural networks. Such insights could broadly affect risk management, trading strategies, and financial policy formulation.

This work contributes to the existing literature by leveraging machine learning techniques to offer a novel perspective on volatility prediction. Specifically, by investigating the capability of neural networks to capture the temporal structure of realized volatility and by introducing the σ-Cell approach, this study aims to advance both the understanding and prediction of volatility in financial markets.

In summary, this research seeks to further the field of financial volatility modeling by integrating machine-learning techniques with domain-specific knowledge. By introducing the σ-Cell, σ-LSTM approaches and exploring LSTM-based neural networks, this study aims to provide a fresh perspective on volatility prediction, thereby contributing to ongoing efforts to understand and forecast fluctuations in financial markets.

Chapter 2

Background

2.1 Volatility Models

Volatility forecasting plays a pivotal role in finance and risk management Poon and Granger [2003], Engle [1982b], Jorion et al. [2007]. As a gauge of market risk and uncertainty, accurate volatility forecasts guide portfolio allocation, option pricing, and value-at-risk calculations Brooks and Persand [2003], Giot [2005]. This makes volatility modeling a crucial capability for investors, financial institutions, and policymakers Engle [1982b], Angelidis and Degiannakis [2008]. However, volatility forecasting remains challenging due to the latent and dynamic nature of volatility arising from fluctuating market conditions Tsay [2005], Francq and Zakoian [2010].

Financial time series exhibits pervasive stylized facts like heteroskedasticity, fat tails, and long memory Cont [2001]. For asset returns $X_t = \log P_{t+1} - \log P_t$, the non-constant volatility presents difficulties for inference and prediction Tsay [2005]. Volatility clustering and sudden spikes violate the constant variance assumption underlying classical methods Francq and Zakoian [2010].

While extensive research explores volatility modeling, gaps remain in translating models into practice Satchell and Knight [2011]. Many studies focus on statistical significance rather than economic value and real-world usage Patton and Sheppard [2009]. This highlights the need for thoughtful volatility model assessment attuned to industry objectives. Overall, volatility forecasting remains active as markets evolve and new models are developed to manage risk in an ever-changing world.

Modeling asset returns is complicated by time-varying and latent volatility dynamics. Let returns be defined as $X_t = \log P_{t+1} - \log P_t$, where P_t is the asset price. The volatility σ_t is the standard deviation of returns conditional on past information $\mathcal{F}_{t-1}$ Shreve et al. [2004], equation 2.1.

$$\sigma_t^2 = \text{Var}[X_t \mid \mathcal{F}_{t-1}] \tag{2.1}$$

Volatility evolves randomly over time, driven by fluctuating market conditions and economic factors Poon and Granger [2003]. This violates the constant variance assumption of basic statistical models Tsay [2005].

Classical financial models like Black-Scholes often disregard heteroskedasticity for tractability Black and Scholes [1973]. However, empirical evidence shows volatility clustering, sudden spikes, and long memory patterns Ding et al. [1993], Poon [2005], Francq and Zakoian [2010]. Capturing these dynamics requires flexible stochastic volatility models Kim et al. [1998].

Popular approaches include GARCH models that represent volatility as an autoregressive process Bollerslev [1986], stochastic volatility models with an unobserved latent volatility process Hull and White [1987b], realized volatility models using high-frequency data Barndorff-Nielsen and Shephard [2002b], and hybrid methods Engle et al. [2013]. Each has advantages and tradeoffs for modeling complex volatility dynamics.

2.1.1 GARCH family models

The ARCH model, introduced by Robert Engle, addresses heteroskedasticity by setting the conditional variance process as autoregressive and modeling log returns as volatility-adjusted white noise where the innovations are independent and identically distributed (i.i.d) with zero mean and unit variance Engle [1982a]. To ensure positive volatility, all model coefficients must be nonnegative.

Bollerslev (1986) further improved the ARCH model by introducing the GARCH (Generalized Autoregressive Conditional Heteroskedasticity) model, integrating an additional autoregressive structure within the conditional variance, equation 2.2

Bollerslev [1986].

GARCH models are used to estimate and forecast volatility by modeling the conditional variance of returns. They capture the effects of past squared returns and past conditional variances on the current conditional variance. GARCH models are widely used in finance due to their ability to handle volatility clustering, leverage effects, and changing conditional volatility, equation 2.2

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{p} \alpha_i x_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2 \tag{2.2}$$

$$x_t \sim i.i.d.\ N\left(0, \sigma_t^2\right) \tag{2.3}$$

The GARCH model, particularly the GARCH(1,1), is favored for financial time series modeling due to its tractability and the inherently discrete nature of financial data Francq and Zakoian [2010].

Volatility in financial markets is characterized by time-varying processes and parameters, which complicates the estimation of risk and uncertainty measures like σ. Various shapes of returns distributions have been identified, including fat tails, volatility clustering, and leverage effects Engle [1982b], Bollerslev [1986], Black [1976], Nelson [1990], Mandelbrot [1967], Fama [1965]. These stylized facts have significant implications for volatility prediction Diebold [1998], Poon and Granger [2003].

Volatility exhibits patterns such as clustering, where returns tend to group in high or low volatility periods, and leverage effects, where volatility increases when the market drops. Other features include mean reversion, where volatility reverts to a long-term mean, and cross-correlation, where volatility correlations across assets and markets often strengthen during downturns Diebold [1998].

Despite its utility, the traditional GARCH model has limitations, such as the assumption of normality and difficulty capturing sudden volatility jumps. To address these issues, various extensions have been introduced. For example, the GJR-GARCH model captures asymmetric volatility reactions 2.4. In equation 2.4, I_{t-1} is an indicator function that takes the value 1 if $x_{t-1} < 0$ and 0 otherwise Glosten et al. [1993a].

$$\sigma_t^2 = \alpha_0 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2 + \sum_{i=1}^{p} \alpha_i x_{t-i}^2 + \gamma x_{t-1}^2 I_{t-1} \tag{2.4}$$

Similarly, the TARCH model is another extension 2.5. Let x_{t-1}^+ be x_{t-1} if $x_{t-1} > 0$ and 0 otherwise, and x_{t-1}^- be x_{t-1} if $x_{t-1} \leq 0$ and 0 otherwise Zakoian [1994].

$$\sigma_t = \alpha_0 + \sum_{j=1}^{q} \beta_j \sigma_{t-j} + \sum_{i=1}^{p} \alpha_i^+ x_{t-i}^+ + \sum_{i=1}^{p} \alpha_i^- x_{t-i}^- \tag{2.5}$$

Another notable variant is Nelson's EGARCH model, which formulates dependencies in log variance $\log(\sigma_t^2)$, as shown in equation 2.6, where $g(Z_t) = \theta Z_t + \lambda(|Z_t| - E(|Z_t|))$. The formulation for g allows the sign and the magnitude to have separate effects on the volatility, which provides a more nuanced understanding of the asymmetric relationships between observations and subsequent volatility shifts Nelson [1991].

$$\log \sigma_t^2 = \omega + \sum_{k=1}^{q} \beta_k g(Z_{t-k}) + \sum_{k=1}^{p} \alpha_k \log \sigma_{t-k}^2 \tag{2.6}$$

The GARCH family of models, building on the foundational ARCH model, offers a robust framework to work with the complex financial time series and the dynamic volatilities therein. With each variant addressing specific nuances, the GARCH models have their place as indispensable tools in mathematical finance.

2.1.2 Stochastic Volatility Models

Stochastic volatility (SV) models allow volatility to follow a stochastic process rather than being constant. This better captures the time-varying and random nature of volatility in financial markets Hull and White [1987b].

Stock prices are often modeled using stochastic differential equations (SDEs) due to the inherent randomness in their movements, equation 2.7, where S is the stock price, μ is the drift coefficient, σ is the diffusion coefficient, and W_t is the Wiener process or Brownian Motion Mandelbrot [1967].

$$\mathrm{d}S = \mu S \mathrm{d}t + \sigma S \mathrm{d}W_t, \tag{2.7}$$

The Geometric Brownian Motion (GBM) model, assuming a constant volatility σ, is given by equation 2.8, where S_0 is the initial stock price.

$$S(t) = S_0 e^{(\mu - \frac{1}{2}\sigma^2)t + \sigma W_t} \tag{2.8}$$

The z_t is defined as a noise term in the context of the SV model, equation 2.9.

$$\log(\sigma_t^2) = \eta + \phi(\log(\sigma_{t-1}^2) - \eta) + z_t \tag{2.9}$$

In this context, z_t is a white noise or shock term that introduces randomness into the log volatility process. Typically, this term is assumed to be normally distributed with a mean of zero and some variance σ_z^2.

$$z_t \sim N\left(0, \sigma_z^2\right) \tag{2.10}$$

It represents unanticipated changes or shocks in the log volatility that are not explained by the model's other components. In time series analysis, noise or shock terms like z_t are essential as they capture unpredictable fluctuations that deterministic or autoregressive components of the model might not capture.

While GARCH models treat volatility as a deterministic process, SV models incorporate it as a latent stochastic process that evolves randomly over time Taylor [1982]. Early SV models, such as the Hull-White model Hull and White [1987b], model log volatility as a mean-reverting Ornstein-Uhlenbeck process Ornstein [1930].

The Heston model Heston [1993] allows volatility to follow a mean-reverting square-root process (the Cox-Ingersoll-Ross or CIR process) that's correlated with returns. This correlation permits closed-form pricing solutions, making the Heston model popular for options pricing and risk management. Notably, the model captures the volatility smile observed in financial markets. The Heston model's SDEs are represented in equations 2.11 and 2.12

$$dS_t = \mu S_t dt + \sqrt{v_t} S_t dW_t^1 \tag{2.11}$$

$$dv_t = \kappa(\theta - v_t)dt + \sigma\sqrt{v_t}dW_t^2 \tag{2.12}$$

For empirical analysis, discrete-time SV models have grown in popularity. One basic specification models log-volatility as an AR(1) process with an additive noise term $z_t \sim N(0, \sigma_z^2)$, where ϕ determines persistence Jacquier et al. [2002], Kim et al. [1998].

Various extensions of the basic SV model have been proposed to capture more empirical features. For example, SVJ models incorporate jumps in returns and volatility Bates [1996]. SVCJ and SABR models further extend the framework, with the latter using a lognormal process for volatility Duffie et al. [2000], Hagan et al. [2002]. Multivariate SV models estimate stochastic volatility for multiple assets via maximum likelihood Sandmann and Koopman [1998] or Bayesian MCMC techniques Jacquier et al. [2002].

In conclusion, SV models offer a flexible approach for modeling complex empirical dynamics, including volatility clustering, jumps, and leverage effects Shephard [2005]. Available in both continuous and discrete-time formats, advancements in Bayesian methods and MCMC have enhanced the estimation of these latent volatility models.

2.1.3 Realized Volatility

Realized volatility (RV) provides a model-free, nonparametric estimate of daily volatility based on high-frequency intraday returns Andersen et al. [2003], Barndorff-Nielsen and Shephard [2002a]. Let $p(t)$ denote the log price of an asset at time t. The intraday return over a period Δ is defined in equation 2.13.

$$r_{t-j\Delta} = p_{t-j\Delta} - p_{t-(j+1)\Delta} \tag{2.13}$$

The daily RV is then calculated as the square root of summed squared intraday returns Andersen et al. [2003], equation 2.14.

$$RV_t^d = \sqrt{\sum_{j=0}^{M-1} r_{t-j\Delta}^2} \tag{2.14}$$

Under general semimartingale conditions, as the frequency of returns increases, RV is a consistent estimator of the integrated variance (IV) Mancini [2009], Barndorff-Nielsen and Shephard [2002b], equation 2.15.

$$IV_t^d = \int_t^{t-1d} \sigma^2(\omega) d\omega \tag{2.15}$$

Where IV represents the true latent daily volatility. This demonstrates RV as a robust realized measure of volatility Patton [2011]. However, market microstructure noise can bias RV, warranting adjustments Hansen and Lunde [2006]. Overall, RV allows model-free volatility estimation, useful for forecasting and risk management Andersen et al. [2007].

Realized volatility (RV) calculated from high-frequency intraday returns provides an accurate nonparametric measure of volatility Andersen et al. [2003]. However, RV alone does not model dynamics. Heterogeneous autoregressive (HAR) models forecast RV by incorporating its persistence across different time horizons Corsi [2009].

The heterogeneity in HAR models reflects varying trading horizons and volatility perspectives among market participants Müller et al. [1993]. Short-term traders react to daily fluctuations, while longer-term investors focus on weekly or monthly volatility Corsi [2009]. The HAR model captures this by including lagged RV components at daily, weekly, and monthly frequencies.

$$RV_t = c + \beta_d RV_{t-1}^{(d)} + \beta_w RV_{t-1}^{(w)} + \beta_m RV_{t-1}^{(m)} + \epsilon_t \tag{2.16}$$

In equation 2.16 $RV_t^{(d)}$, $RV_t^{(w)}$, and $RV_t^{(m)}$ are the daily, weekly (5-day), and monthly (22-day) RV, respectively. The additive cascade structure allows volatility components across horizons to influence forecasts.

To address issues related to negativity and to approximate normal distributions, the log-transformed RV is often used. The log-transformed RV aggregated over n periods is given by:

$$\log \mathrm{RV}t^{(n)} = \frac{1}{n} \left(\log \mathrm{RV}t + \ldots + \log \mathrm{RV}_{t-n+1} \right) \tag{2.17}$$

This log transformation is particularly useful for modeling RV dynamics across

different investment horizons, as it allows for a more stable and accurate estimate of realized volatility Corsi et al. [2012].

$$\log \mathrm{RV}t + 1d^{(d)} = c + \beta^{(d)} \log \mathrm{RV}t^{(d)} + \beta^{(w)} \log \mathrm{RV}t^{(w)} + \beta^{(m)} \log \mathrm{RV}t^{(m)} + \epsilon_{t+1d} \quad (2.18)$$

The HAR model with jumps (HAR-CJ) incorporates separate jump and continuous sample path variance components into the forecasting model Corsi et al. [2012]. This allows disentangling the impacts of jumps vs. diffusion-based continuous price movements on future volatility. An alternative approach models log jumps and continuous log RV in the HAR structure Andersen et al. [2012]. Incorporates lagged log realized variances, bipower variation, and log realized semivariances into the HAR model Zheng et al. [2014]. This captures volatility asymmetry and downside risk.

HAR models effectively incorporate the heterogeneity and memory of volatility arising from varying investor horizons Bollerslev et al. [2016]. Empirical studies show HAR forecasts outperform GARCH models and various volatility proxies Corsi [2009], Patton and Sheppard [2015]. Extensions incorporate other factors like volume, liquidity, and returns Fuertes et al. [2009]. Overall, the flexible HAR framework provides an intuitive approach to modeling RV dynamics.

2.1.4 Hybrid Approaches

Financial volatility modeling is a rapidly evolving field, with researchers continually exploring hybrid approaches that amalgamate various models to capture intricate patterns in financial time series data Shephard [2005]. These hybrid models have gained significant attention in volatility forecasting for their ability to leverage the strengths of different modeling techniques. Broadly, these hybrid approaches can be categorized into four types: Statistical-Statistical Hybrids, Machine Learning-Machine Learning Hybrids, Statistical-Machine Learning Hybrids, and Ensemble Approaches. Each category typically encapsulates specialized modules, rendering these models versatile and adaptive.

One prevalent approach combines ARCH-type and SV models Engle and Gallo

[2006]. In this configuration, the ARCH-type model captures autoregressive patterns in conditional volatility, while the SV model accounts for stochastic volatility. The outputs from both models are then synthesized to produce the final volatility prediction Shephard [2005].

A different hybrid model that integrates elements of GARCH and SV models is the GAS model proposed Christoffersen et al. [2008]. These hybrids aim to capture both autoregressive and stochastic components of volatility. Examples include the SV-GARCH Harvey et al. [1994], GARCH-Jump Maheu and McCurdy [2004], and SV-Jump models Eraker et al. [2003]. While computationally more demanding than standalone GARCH or SV models, they offer more accurate volatility estimates by leveraging the strengths of both frameworks Angelidis et al. [2008]. The flexibility of hybrid models makes them appealing, though estimation complexity remains a practical challenge.

Another innovative approach utilizes the GARCH-MIDAS (Mixed Data Sampling) model to account for the influence of macroeconomic indicators on volatility. This model integrates high-frequency financial data with low-frequency macroeconomic data. The GARCH model is applied to the high-frequency data, while the MIDAS approach incorporates macroeconomic indicators through a set of dynamically updated weights Engle et al. [2013].

Another hybrid approach merges the GARCH model with the Markov Switching model. Here, the GARCH model captures autoregressive patterns in conditional volatility, while the Markov Switching model is applied to the residuals to capture regime-switching behavior Hamilton [2020].

Machine learning techniques have been increasingly employed in financial tasks such as anomaly detection and volatility prediction Cavalcante et al. [2016]. Despite their ability to learn complex patterns from data without imposing rigid functional forms, these machine-learning approaches have shown mixed results in outperforming traditional econometric models for predicting realized volatility Van Dijk and Franses [2003]. This inconsistency in performance could be attributed to challenges in capturing asymmetric volatility dynamics, which arise from factors like heterogeneous agent behavior and information impact effects Black [1976], Nelson [1990].

Similarly, neural networks have been applied to various forecasting tasks, including stock price prediction Khan [2011] and volatility forecasting Bucci [2020]. The performance of these models has been inconsistent; some studies indicate poor out-of-sample forecasting performance for nonparametric models like neural networks Clements and Krolzig [1998], Pavlidis et al. [2012], while others report promising results Rosa et al. [2014], Miura et al. [2019]. This further underscores the challenges and limitations of using machine learning techniques in financial forecasting tasks.

The Neural Stochastic Volatility Model (NSVM) is an unconditional generative model that uses two pairs of RNNs and Multi-Layer Perceptrons (MLPs) to generate sequences of latent and observable variables over time Luo et al. [2018].

A novel hybrid model has emerged that combines GARCH models with Long Short-Term Memory (LSTM) networks, a specific type of Recurrent Neural Networks (RNNs). This approach initially applies GARCH models to financial time series data and then incorporates these predictions as input features into LSTM networks Hochreiter and Schmidhuber [1997b], Hu et al. [2020].

Recent studies have introduced unique models like RNNs that capture the ω-constant of the GARCH process Nguyen et al. [2020] and hybrids that combine Stochastic Volatility (SV) models with Simple Recurrent Units (SRU) Nguyen et al. [2022]. These models show promise in capturing long-term memory effects and auto-dependence of volatility.

One of the emerging trends is the integration of traditional volatility models with RNNs. The σ-Cell methodology, which combines the GARCH process with RNN dynamics, introduces a fresh perspective on volatility estimation Rodikov and Antulov-Fantulin [2023].

Financial volatility modeling is marked by a rich tapestry of hybrid models, each with unique strengths and limitations. As the field continues to evolve, it remains crucial to rigorously assess these models' performance, especially in out-of-sample forecasting tasks.

2.2 Machine Learning

The utility of machine learning algorithms becomes particularly evident when addressing computational tasks of high complexity that defy manual programming. Consider, for example, the abstract problem of inferring underlying structures within a high-dimensional data space; traditional algorithmic approaches may prove insufficient. However, by leveraging a plethora of labeled data instances, one can employ supervised learning techniques to construct an effective predictive model. To formalize the concept of supervised learning, let us consider an input space X and an output space Y. Let D represent the empirical distribution of observed data, comprising tuples (x, y) where $x \in X$ serves as the input and $y \in Y$ is the corresponding ground truth output. The objective is to identify a function $f : X \to Y$ that minimizes the prediction error, quantified as the discrepancy between the predicted and true output values.

While the ultimate goal is to minimize the test error, which is the expected error over the entire distribution D, this is generally infeasible due to the unobservable nature of D. Consequently, the focus shifts to minimizing the training error, which serves as an approximation of the test error. A critical challenge herein lies in ensuring that a model that performs well on the training set also generalizes effectively to unseen data, a dilemma commonly referred to as the generalization problem Hastie et al. [2009], Bishop and Nasrabadi [2006].

One strategy to address the generalization problem is to constrain the hypothesis space $\mathcal{F}$ from which the function f is selected. If the cardinality of $\mathcal{F}$ is relatively small in comparison to the size of the training set, the training error becomes a reliable estimator of the test error. However, this approach has its limitations; specifically, the optimal test error achievable may exceed acceptable thresholds.

The dimensionality of $\mathcal{F}$ should be commensurate with the complexity of the training data. While a smaller $\mathcal{F}$ simplifies the learning process, it may lack the expressive power to capture the intricacies of the task at hand. Conversely, an overly complex $\mathcal{F}$ risks overfitting to the training data. There exists no universal criterion for selecting the optimal $\mathcal{F}$; the choice is contingent upon domain-specific knowledge and empirical evidence from related tasks.

Upon the selection of an appropriate hypothesis space $\mathcal{F}$ and the accumulation

of a sufficiently large training data set, the subsequent computational challenge lies in identifying a function f within $\mathcal{F}$ that minimizes the training error. Although the search for an optimal function is computationally intractable in the general case, one can resort to optimizing smoother approximations of $\mathcal{F}$ through gradient-based methods.

Let f_θ denote a smooth approximation of $\mathcal{F}$, parameterized by θ. Assuming that both f_θ and the loss function L are differentiable, the gradient of the training error with respect to θ can be computed analytically. Under these conditions, Gradient Descent (GD) serves as a viable optimization algorithm for minimizing the training error Ruder [2016]. In GD, the parameter vector θ is iteratively updated in the direction of the negative gradient, thereby reducing the training error. The magnitude of these iterative updates is governed by the learning rate ε, a hyperparameter that necessitates careful tuning.

The convergence properties of GD have been extensively studied and are well-understood under specific conditions. For instance, when the objective function to be minimized is a positive definite quadratic function, GD exhibits predictable convergence rates Nesterov [2003]. For more general classes of functions, although an exact rate of convergence may not be readily derivable, empirical estimates can often provide insights into the algorithm's efficiency in locating a local minimum.

2.2.1 RNN

Recurrent Neural Networks (RNNs) are a type of neural network designed to handle sequential data by maintaining a hidden state that carries information across time steps. RNNs have been applied to time series forecasting, natural language processing, and other sequential tasks. The "vanishing gradient" problem limits the ability of RNNs to learn long-term dependencies in the data. Various modifications, such as the LSTM and GRU cells, have been proposed to address this limitation.

RNNs send information cyclically, encompassing both past and current inputs. The way RNNs relay information from a prior iteration to its hidden layer is captured mathematically Zhang et al. [2021]. This involves defining the hidden and input states at time t, matrices representing input-to-hidden and hidden-to-hidden connections,

and a bias. The aggregated data then interacts with an activation function to ensure compatibility with backpropagation, the equations for hidden and output states 2.19, 2.20.

$$\mathbf{h}_t = \phi_h \left(\mathbf{X}_t \mathbf{W}_{xh} + \mathbf{h}_{t-1} \mathbf{W}_{hh} + \mathbf{b}_h \right) \tag{2.19}$$

$$\mathbf{O}_t = \phi_o \left(\mathbf{h}_t \mathbf{W}_{ho} + \mathbf{b}_o \right) \tag{2.20}$$

Considering that $\mathbf{h}_t$ considers its preceding state, all prior hidden states influence it, we can train RNNs by the Backpropagation technique.

In the process of passing our input $\mathbf{X}_t$ forward through the network, we sequentially calculate the hidden state $\mathbf{H}_t$ and the output state $\mathbf{O}_t$. Subsequently, we employ a loss function $\mathcal{L}(\mathbf{O}, \mathbf{Y})$ to quantify the disparity between all the computed outputs $\mathbf{O}_t$ and their corresponding target values $\mathbf{Y}_t$, equation 2.21.

$$\mathcal{L}(\mathbf{O}, \mathbf{Y}) = \sum_{t=1}^{T} \ell_t \left(\mathbf{O}_t, \mathbf{Y}_t \right) \tag{2.21}$$

This aggregates every individual loss value ℓ_t from each iterative step. Depending on the problem, this loss term ℓ_t can be defined in multiple ways, such as Mean Squared Error, Hinge Loss, and Cross Entropy Loss; in our case, we employed not deterministic, as MSE or MAE, but probabilistic loss using MLE equation 4.30.

The backpropagation approach to cater to RNNs. It unfolds the RNN to resemble a conventional Feedforward Neural Network, making it apt for backpropagation. During the propagation of data forward through this network, hidden and output states are determined in a sequence. A loss function measures the discrepancy between the generated outputs and the target values.

Neural networks can handle complex patterns in data, but they are deterministic, meaning they always produce the same output for a given input. One way to address this is by adding latent variables to the neural networks, but this can make calculations hard. Recent research has found a way to make these calculations more manageable by using a method called variational inference, when we add hidden continuous variables into the neural network structure Kingma and Welling [2013], Rezende et al. [2014].

Another idea is to come up with a way to combine observable and hidden variables in a sequenceChung et al. [2015], while another possibility is to use different types of layers in the network that take advantage of the network's Markovian properties Fraccaro et al. [2016].

2.2.2 Advanced Cells

In more recent developments, various RNN architectures, including LSTM Hochreiter and Schmidhuber [1997c], Gated Recurrent Units (GRU)Cho et al. [2014a], and Statistical Recurrent Units (SRU)Oliva et al. [2017], have demonstrated significant potential in forecasting time-series data sets. Among these, the LSTM architecture stands out, especially for its exemplary performance in the intricate domain of volatility prediction Bucci [2020], Rodikov and Antulov-Fantulin [2022].

Advanced cells, such as the Gated Recurrent Unit (GRU) and the Peephole LSTM Gers et al. [2000], offer variations on the LSTM architecture to improve performance or reduce computational complexity. These cells have been used in various applications and have shown improvements over traditional LSTMs in specific tasks Gers et al. [2000]. The choice of cell type depends on the problem, the data, and the available computational resources.

2.2.3 LSTM

Long Short-Term Memory (LSTM) networks are a type of RNN with a more sophisticated cell structure designed to handle long-term dependencies better. LSTMs use gates (input, forget, and output gates) to control the flow of information within the cell. They have been widely used in applications requiring sequential data processing, including time series forecasting and natural language processing. LSTMs have shown promise in modeling and forecasting volatility in financial markets.

LSTM is a specific cell type in a recurrent neural network capable of catching long-term dependencies in data and fixing this exploding or vanishing gradient issue. Achievement is possible due to the cell state and a combination of four gates interacting. The ability to eliminate or add information to the cell state, carefully regulated by gates structures, is a crucial difference with RNN. LSTM cells contain an additional state,

which helps to internally maintain input memory, making them especially suitable for solving problems associated with sequential. Cell state conveys relative information along the entire chain of the sequence. The state of the cell reflects the corresponding information throughout the processing of the series, so data from earlier time steps can participate in later time steps Olah [2015].

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \tag{2.22}$$

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \tag{2.23}$$

$$\tilde{C}_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right) \tag{2.24}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.25}$$

$$o_t = \sigma\left(W_o\,[h_{t-1}, x_t] + b_o\right) \tag{2.26}$$

$$h_t = o_t * \tanh\left(C_t\right) \tag{2.27}$$

2.2.4 GRU

The Gated Recurrent Unit (GRU) is a variation of the LSTM architecture that simplifies the learning process by eliminating the cell state. On the other hand, the Statistical Recurrent Unit (SRU) is designed to discern long-term dependencies in data. It achieves this by leveraging simple moving averages of summary statistics and linear combinations of historical data.

GRU is a new generation of recurrent neural networks, very similar to LSTMs, a variation on the LSTM, introduced by Cho et al. [2014b]. The update gate combines the forget and input gates, and the cell state merges with the hidden state called the reset gate. As a result, the model is more straightforward, and the training procedure takes less time than the net with standard LSTM instead of GRU. By this modification, LSTM and GRU fix the vanishing/exploding gradient problem encountered by traditional RNN

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right) \tag{2.28}$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right) \tag{2.29}$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right) \tag{2.30}$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \tag{2.31}$$

2.2.5 MUT1 and MUT2

MUT1 and MUT2 are novel RNN cells designed to capture different aspects of temporal dependencies in sequences Jozefowicz et al. [2015]. MUT1 architecture is defined by equations 2.32-2.34

$$z = \text{sigm}\left(W_{xz}x_t + b_z\right) \tag{2.32}$$

$$r = \text{sigm}\left(W_{xr}x_t + W_{hr}h_t + b_r\right) \tag{2.33}$$

$$h_{t+1} = \tanh\left(W_{hh}\left(r \odot h_t\right) + \tanh\left(x_t\right) + b_h\right) \odot z + h_t \odot (1 - z) \tag{2.34}$$

Here, z is the update gate, which decides how much of the new hidden state should be updated. r is the reset gate, which determines how much of the past hidden state h_t should be forgotten. h_{t+1} is the new hidden state, which is a combination of the previous hidden state h_t and a candidate hidden state generated using r and z.

The unique feature of MUT1 is the use of the hyperbolic tangent function directly on the input x_t, which adds an additional non-linearity to the candidate hidden state.

MUT2 is another RNN cell that aims to capture temporal dependencies but with slight modifications to the MUT1 architecture. The equations defining MUT2 in 2.35-2.37.

$$z = \text{sigm}\left(W_{xz}x_t + W_{hz}h_t + b_z\right) \tag{2.35}$$

$$r = \text{sigm}\left(x_t + W_{hr}h_t + b_r\right) \tag{2.36}$$

$$h_{t+1} = \tanh\left(W_{hh}\left(r \odot h_t\right) + W_{xh}x_t + b_h\right) \odot z + h_t \odot (1 - z) \tag{2.37}$$

Similar to MUT1, z is the update gate, and r is the reset gate. However, in MUT2, the update gate z is influenced by both the current input x_t and the previous hidden state h_t. Additionally, the candidate hidden state in h_{t+1} is influenced by a weighted sum of the current input x_t, unlike in MUT1, where a hyperbolic tangent of x_t is used. Both MUT1 and MUT2 aim to provide a nuanced understanding of temporal dependencies in sequence data. They offer variations in the gating mechanisms and the generation of the candidate hidden state, allowing for flexibility in capturing different types of sequence patterns.

2.2.6 Concluding Remarks on RNN Cell Types

In summary, various RNN architectures have been developed to address the challenges of modeling sequential data. LSTM networks have been particularly effective in time-series forecasting, demonstrating superior performance Siami-Namini et al. [2019], Song et al. [2020].

However, other architectures like GRU offer advantages in terms of computational efficiency. GRUs have been shown to converge faster during the training process, making them a suitable choice for applications where computational resources or time constraints Cho et al. [2014b].

The more recent MUT1 and MUT2 cells Jozefowicz et al. [2015] offer interesting alternatives. Preliminary studies suggest these architectures may be particularly well-suited for audio sequence modeling.

Therefore, the choice of RNN cell type should be guided by the specific requirements of the task, the nature of the data, and the available computational resources. While LSTMs may offer the best performance in many scenarios, the faster convergence of GRUs and the specialized capabilities of MUT1 and MUT2 make them worthy contenders for specific applications.

2.2.7 Backpropagation

Backpropagation is the optimization algorithm that trains neural networks, including RNNs and LSTMs. It involves computing the gradient of the loss function with respect to the model parameters and updating the parameters to minimize the loss.

In the architecture of a RNN, we encounter three distinct weight matrices, $\mathbf{W}_{xh}$, $\mathbf{W}_{hh}$ and $\mathbf{W}_{ho}$, Equations 2.19 and 2.20. It is imperative to compute the partial derivatives with respect to each of these weight matrices for the optimization process. Utilizing the chain rule of calculus, we derive the expression for $\mathbf{W}_{ho}$ equation 2.38.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{ho}} = \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial \mathbf{O}_t} \cdot \frac{\partial \mathbf{O}_t}{\partial \phi_o} \cdot \frac{\partial \phi_o}{\mathbf{W}_{ho}} = \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial \mathbf{O}_t} \cdot \frac{\partial \mathbf{O}_t}{\partial \phi_o} \cdot \mathbf{H_t} \tag{2.38}$$

For the partial derivative with respect to $\mathbf{W}_{hh}$ equation 2.39.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{hh}} = \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial \mathbf{O}_t} \cdot \frac{\partial \mathbf{O}_t}{\partial \phi_o} \cdot \frac{\partial \phi_o}{\partial \mathbf{H}_t} \cdot \frac{\partial \phi_h}{\partial \phi_h} \cdot \frac{\partial \phi_h}{\partial \mathbf{W}_{hh}} = \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial \mathbf{O}_t} \cdot \frac{\partial \mathbf{O}_t}{\partial \phi_o} \cdot \mathbf{W}_{ho} \cdot \frac{\partial \mathbf{H}_t}{\partial \phi_h} \cdot \frac{\partial \phi_h}{\partial \mathbf{W}_{hh}} \tag{2.39}$$

For the partial derivative with respect to $\mathbf{W}_{xh}$, equation 2.40.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{xh}} = \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial \mathbf{O}_t} \cdot \frac{\partial \mathbf{O}_t}{\partial \phi_o} \cdot \frac{\partial \phi_o}{\partial \mathbf{H}_t} \cdot \frac{\partial \phi_h}{\partial \phi_h} \cdot \frac{\partial \phi_h}{\partial \mathbf{W}_{xh}} = \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial \mathbf{O}_t} \cdot \frac{\partial \mathbf{O}_t}{\partial \phi_o} \cdot \mathbf{W}_{ho} \cdot \frac{\partial \mathbf{H}_t}{\partial \phi_h} \cdot \frac{\partial \phi_h}{\partial \mathbf{W}_{xh}} \tag{2.40}$$

Owing to the temporal dependencies inherent in the recurrent neural network architecture, each $\mathbf{H}_t$ is contingent upon its preceding time step, as shown in Equations2.41 and 2.42.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{hh}} = \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial \mathbf{O}_t} \cdot \frac{\partial \mathbf{O}_t}{\partial \phi_o} \cdot \mathbf{W}_{ho} \sum_{k=1}^{t} \frac{\partial \mathbf{H}_t}{\partial \mathbf{H}_k} \cdot \frac{\partial \mathbf{H}_k}{\partial \mathbf{W}_{hh}} \tag{2.41}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{xh}} = \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial \mathbf{O}_t} \cdot \frac{\partial \mathbf{O}_t}{\partial \phi_o} \cdot \mathbf{W}_{ho} \sum_{k=1}^{t} \frac{\partial \mathbf{H}_t}{\partial \mathbf{H}_k} \cdot \frac{\partial \mathbf{H}_k}{\partial \mathbf{W}_{xh}} \tag{2.42}$$

The rewritten mathematical formulation is represented in Equations 2.43 and 2.44.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{hh}} = \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial \mathbf{O}_t} \cdot \frac{\partial \mathbf{O}_t}{\partial \phi_o} \cdot \mathbf{W}_{ho} \sum_{k=1}^{t} \left(\mathbf{W}_{hh}^{\top}\right)^{t-k} \cdot \mathbf{H}_k \tag{2.43}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{xh}} = \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial \mathbf{O}_t} \cdot \frac{\partial \mathbf{O}_t}{\partial \phi_o} \cdot \mathbf{W}_{ho} \sum_{k=1}^{t} \left(\mathbf{W}_{hh}^{\top}\right)^{t-k} \cdot \mathbf{X}_k \tag{2.44}$$

In the process of backpropagation through time (BPTT), we need to store powers of $\mathbf{W}_{hh}^{k}$ for each loss term ℓ_t in the overall loss function $\mathcal{L}$. This can lead to numerical instability. Precisely, eigenvalues smaller than 1 vanish, and those larger than one diverge Bengio et al. [1994]. One approach to address the numerical instability is truncating the sum at a computationally convenient size Zhang et al. [2021], a technique known as Truncated BPTT. Truncated backpropagation is often considered the best way to train RNNs Williams and Peng [1990]. It has been effectively used for training RNNs in tasks like word-level language modeling, showing better results than larger N-gram models Mikolov et al. [2010, 2011]. It sets an upper bound on the number of time steps for which the gradient can flow back Sutskever [2013]. This can be thought of as a moving window of past time steps that the RNN considers, ignoring anything before the cut-off time step.

An issue with standard BPTT is that it takes a lot of computational power to update the model's parameters even once. For example, training an RNN on a sequence of k elements is as computationally demanding as running both a forward and backward pass in a neural network with k layers.

We can reduce the computational cost by simply breaking the k-element sequence into more minor sequences and treating each as a separate training example.

Truncated BPTT is similar to the naive method regarding computational cost per update, but it's better at capturing long-term patterns. It processes the sequence step-by-step and updates the parameters every step by looking back one step. The model can learn from many steps while keeping the computational cost low. As a result, the model can use information from far back in the sequence when needed.

Like many neural networks, RNNs suffer from the problem of vanishing or exploding gradients Arjovsky et al. [2016]. In Equations 9 and 10, the term $\frac{\partial \mathbf{H}_t}{\partial \mathbf{H}_k}$ introduces matrix multiplication over a potentially long sequence. Small values than one in this

multiplication can cause the gradient to vanish with each time step, or conversely, large values than one can lead to exploding gradients Zhang et al. [2021].

2.2.8 Adam Optimizer

Adam (Adaptive Moment Estimation) is an adaptive gradient descent optimization algorithm commonly used for training deep neural networks due to its efficiency and low memory requirements Kingma and Ba [2014]. Unlike vanilla stochastic gradient descent (SGD), which uses a fixed learning rate, Adam adjusts the learning rate dynamically during training by estimating first and second moments of the gradient Ruder [2016].

Specifically, Adam computes adaptive learning rates for each parameter by taking an exponential moving average of the gradient and squared gradient. These moments are bias-corrected for initialization effects Kingma and Ba [2014]. By adapting the learning rate per parameter, Adam works well for problems with large parameter spaces or noisy/sparse gradients Reddi et al. [2019].

Let's denote the parameters of the model as θ and the objective function (the loss to be minimized) as $J(\theta)$. In the context of training a model, $J(\theta)$ is typically the sum of the log-likelihood loss overall training samples.

At the t^{th} timestep, the update rule for the Adam optimizer is defined in 2.45, 2.46, 2.47, 2.48, 2.49.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2.45}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{2.46}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.47}$$

$$\hat{v}t = \frac{v_t}{1 - \beta_2^t} \tag{2.48}$$

$$\theta t + 1 = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{2.49}$$

In equations 2.45, 2.46, 2.47, 2.48, 2.49 next notations:

- g_t denotes the gradient of the objective function w.r.t. the model parameters at the current timestep t: $g_t = \nabla_\theta J(\theta)$.

- m_t and v_t are estimates of the first and second moments (the mean and the uncentered variance, respectively) of the gradients.
- $\hat{m}_t$ and $\hat{v}_t$ are bias-corrected estimates of the first and second moments.
- β_1 and β_2 are the exponential decay rates for the moment estimates, typically set to 0.9 and 0.999, respectively.
- η is the learning rate, and ϵ is a small constant for numerical stability, typically set to 10^{-8}.

In Adam, the step size is computed as an exponential moving average of the gradient and the squared gradient, which are then bias-corrected. This feature of Adam makes it suitable for problems with large parameter spaces or when the objective function has a noisy or sparse gradient, making it a good choice for training recurrent neural networks.

Extensions of Adam include incorporating Nesterov momentum Dozat [2016], adaptive normalization techniques Loshchilov and Hutter [2017], and rectified variants Liu et al. [2019]. ADAptive Moment estimation methods remain widely used for training neural networks and other differentiable models Ruder [2016].

Other popular adaptive SGD algorithms include RMSprop Igel and Hüsken [2003] and Adadelta Zeiler [2012]. RMSprop maintains per-parameter learning rates based on the magnitude of recent gradients. Adadelta further adapts the learning rate based on a moving window of gradient updates. Together with Adam, these methods demonstrate the effectiveness of dynamic learning rate adaptation for optimizing complex loss landscapes Wilson et al. [2017].

2.2.9 Hyperparameters

Hyperparameters are settings that govern the model training process but are not estimated from data. Selecting optimal hyperparameters is a crucial step in developing high-performance machine learning models. While model parameters are learned during training, hyperparameters must be set a priori based on domain expertise, heuristics, or search strategies Claesen and De Moor [2015]. Major hyperparameters include the

learning rate, batch size, number of epochs, model architecture (e.g., number of layers and units), and regularization parameters Goodfellow et al. [2016].

A variety of techniques exist for hyperparameter tuning. Simple methods like grid search and random search evaluate performance over a manually defined subset of the hyperparameter space Bergstra and Bengio [2012]. More advanced Bayesian optimization techniques construct a probabilistic model to guide the search process towards promising regions Snoek et al. [2012]. The choice of tuning strategy involves tradeoffs between search efficiency and implementation complexity Feurer and Hutter [2019].

Proper hyperparameter configuration can substantially influence model accuracy, training time, and generalization capabilities Maclaurin et al. [2015], Domhan et al. [2015]. Thoughtful hyperparameter selection tailored to the data set, model class, and problem setting is critical for developing performant and robust machine learning solutions. The search for optimal hyperparameters remains an active area of research across deep learning and other model families.

2.3 Accuracy Metrics

Synthetic data sets, where the true generative process is known, are valuable for evaluating statistical and machine-learning models Gretton et al. [2009], Moreno-Torres et al. [2012]. With real-world data, the underlying data-generating process is usually unknown. Synthetic data provides a means of assessing model performance against a ground truth Quinonero-Candela et al. [2008]. In time series analysis and econometrics, synthetic data enables testing volatility predictions when the true volatility path is observable Francq and Zakoian [2019]. This helps determine model accuracy in volatility estimation prior to deployment on real financial data.

Synthetic data generation requires specifying a data model that captures key properties of the real data. For financial data, this may include stylistic properties like autocorrelation, heteroskedasticity, jumps, and fat tails Cont [2001]. Controlled experiments can then evaluate model performance under different controlled generative processes Athey and Imbens [2015]. Cross-validation on real data is still needed, but

synthetic data provides an additional diagnostic.

Real-world financial data sets also have advantages complementary to synthetic data. They capture the nuances of actual market conditions Fama [1998]. Economic events, investor behavior, and market microstructure are naturally embedded O'hara [1998]. Models developed and tested solely on synthetic data may fail to generalize to real data. Testing on real data sets from different time periods and markets is essential Pagan [1996].

In practice, a combination of synthetic and real data is ideal for developing and evaluating financial models Francq and Zakoian [2019], Francq and Zakoïan [2015]. Synthetic data allows diagnosing accuracy and tuning models. Real data then evaluates real-world performance across different markets and time periods. The dual use of synthetic and real data leverages the strengths of each in developing robust and generalizable models.

2.3.1 Volatility Forecast Evaluation Metrics

Our study utilized multiple models to evaluate their forecasting performance. To assess the accuracy of these models, we employed multiple approaches. The R^2 of Mincer-Zarnowitz forecasting regressions Mincer and Zarnowitz [1969].

Mean Absolute Error (MAE) metric evaluates the average magnitude of errors between predicted and observed values without considering their direction, equation 2.50, where σ_t is the observed value and $\hat{\sigma}_t$ is the predicted value at observation t, and T is the total number of observations. In contrast to Root Mean Squared Error (RMSE), MAE treats all errors equally.

$$\text{MAE} = \frac{1}{T}\sum_{i=1}^{n} |\sigma_t - \hat{\sigma}_t| \tag{2.50}$$

Root Mean Squared Error metric assesses the average magnitude of errors between predicted and observed values, equation 2.51, where σ_t is the observed value and $\hat{\sigma}_t$ is the predicted value at observation t, and T is the total number of observations. It is particularly sensitive to outliers since it gives more weight to larger errors than smaller ones. A model with a lower RMSE value is considered better fitting.

$$\text{RMSE} = \sqrt{\frac{1}{T}\sum_{i=1}^{n}(\sigma_t - \hat{\sigma}_t)^2} \tag{2.51}$$

The heteroskedasticity adjusted root mean square error (HRMSE) Bollerslev and Ghysels [1996], which is calculated in equation 2.52, where σ_t is the variance at time t and $\widehat{\sigma}_t$ is the corresponding forecast. HRMSE is a modified version of the RMSE that takes into account the presence of heteroscedasticity in the data.

$$\text{HRMSE} = \sqrt{\frac{1}{T}\sum_{t=1}^{T}\left(\frac{\sigma_t - \widehat{\sigma}_t}{\sigma_t}\right)^2} \tag{2.52}$$

However, as the HRMSE is not considered a robust loss function Patton [2011], we also employed the QLIKE loss function, defined in equation 2.53.

$$\text{QLIKE} = \frac{1}{T}\sum_{t=1}^{T}\left(\log\sigma_t + \frac{\widehat{\sigma}_t}{\sigma_t}\right) \tag{2.53}$$

The QLIKE loss function measures how well a model predicts a set of observations, considering both the mean and variance of the predicted values. It is particularly useful for evaluating volatility models where the focus is on forecasting the variance of returns, which is robust in the context Patton [2011].

The Negative Log-Likelihood (NLL) metric quantifies the fit of a model to observed data by calculating the negative logarithm of the likelihood of the observed data given the model. Lower NLL values indicate a better fit of the model to the observed data. In this study, NLL is defined as shown in equation 2.54, where r_t is the observed return at time $t, \hat{\sigma}_t$ is the model's predicted volatility at time t, and $P\left(r_t \mid \hat{\sigma}_t\right)$ represents the likelihood of observing return r_t given the predicted volatility $\hat{\sigma}_t$.

$$\text{NLL} = -\sum_{t=1}^{T}\log P(r_t|\hat{\sigma}_t) \tag{2.54}$$

Certain evaluation metrics are better suited for assessing volatility forecast models. Metrics like R^2, while popular for evaluating predictive models, have limitations for volatility forecasting Van Dijk and Franses [2003]. R^2 is prone to overfitting, sensitive to scale, and better suited for linear models Tofallis [2015].

Instead, robust loss functions like QLIKE and HRMSE are preferred Patton [2011]. These are designed to handle the challenges of volatility forecasting. Financial returns exhibit conditional heteroskedasticity, where the variance changes over time Engle [1982b]. Standard loss functions like MSE perform poorly under heteroskedasticity. In contrast, QLIKE and HRMSE are robust loss functions that account for time-varying volatility Patton [2011].

For likelihood-based volatility models, the negative log-likelihood (NLL) is an appropriate scoring metric Bollerslev [1986]. NLL measures the model's likelihood of generating the observed data. Models with lower NLL values have higher likelihood and better fit. NLL naturally handles the probability distribution of returns conditional on predicted volatility.

Overall, QLIKE, HRMSE, and NLL are well-suited for evaluating volatility forecasts due to their robustness to heteroskedasticity and basis in likelihood theory Patton [2011]. R^2 has significant limitations in this context that may lead to poor model selection. Careful choice of scoring metrics tailored to the modeling task is crucial for rigorous evaluation.

2.3.2 Value at Risk

Value at Risk (VaR) is a widely used risk measure in finance and banking Jorion [2007], Dowd [2007]. VaR estimates the maximum expected loss on an investment over a given time period at a specified confidence level Dowd [2007].

VaR provides an aggregated risk exposure amount, capturing tail risks and volatility Angelidis and Degiannakis [2008]. It is computed using the distribution of historical returns or Monte Carlo simulation Hendricks [1996]. Common confidence levels are 95% or 99% for daily VaR Jorion [2007]. The time horizon is often one day or ten days, reflecting investment holding periods Pérignon et al. [2008].

VaR has limitations; it does not account for losses exceeding VaR or give insights into the loss distribution Artzner et al. [1999]. Conditional VaR addresses these issues by estimating losses conditional on breaching VaR Rockafellar and Uryasev [2002]. VaR remains widely used in banking regulation and risk management due to its conceptual simplicity Jorion [2000]. Proper modeling of volatility dynamics is key for accurate

VaR forecasting across different confidence levels Angelidis et al. [2004].

2.3.3 Diebold-Mariano Test

The Diebold-Mariano (DM) test is a statistical test for evaluating the predictive accuracy of competing forecasts Diebold and Mariano [1995]. It tests the null hypothesis that the difference in loss between two forecast models is zero against the alternative that there is a significant difference Giacomini and White [2006].

For volatility forecasting, standard loss functions for the DM test are MSE and MAD Patton [2011]. MSE penalizes more significant errors, while MAD weights all errors equally. The DM test computes the loss differential between the two models and assesses if it is significantly different from zero Diebold and Mariano [1995].

The DM test has some limitations to consider. It assumes forecast errors are uncorrelated over time, but volatility exhibits autocorrelation Engle [1982b]. Robust DM tests using Newey-West standard errors help control for autocorrelation Harvey et al. [1997]. The test may also have low power when forecast losses are highly persistent Clark and McCracken [2001].

The DM test provides a formal statistical framework for comparing rival volatility forecast models Patton and Sheppard [2009]. We apply it using MSE and MAD loss functions at 5% significance. However, robust variants and power limitations warrant consideration when interpreting the results.

2.3.4 Model Confidence Set Test

The Model Confidence Set (MCS) test provides a statistical framework for comparing multiple forecast models to determine a superior set of models Hansen et al. [2011]. The MCS test sequentially eliminates inferior models until the remaining set contains the best model with a specified confidence level Barbieri and Berger [2004].

The MCS test uses bootstrapping to construct the empirical distribution of loss differentials between models Politis and Romano [1994]. Bootstrapping is advantageous when the theoretical distribution is unknown or complex to derive Efron and Tibshirani [1994]. It involves resampling with replacement to generate a sampling distribution for a statistic of interest Davison and Hinkley [1997].

We apply the MCS test for volatility forecasting using 10,000 bootstrap iterations at a 5% confidence level Hansen et al. [2011]. The bootstrap-based MCS test makes no distributional assumptions and accounts for the joint evaluation of multiple models Clark and McCracken [2013]. It provides a robust statistical approach for identifying the top-performing subset of volatility forecast models.

Chapter 3

LSTM-RV

3.1 Motivation

Volatility prediction for financial assets is crucial for understanding financial risks. Despite recent advancements in deep learning, they often struggle to outperform robust econometric volatility models due to complexities arising from noise, market microstructure, heteroscedasticity, news effects, and various time scales, among other factors. This study examined the Long Short-Term Memory (LSTM) recurrent neural networks for volatility prediction and compared them with prominent volatility-econometric models. Our investigation focused on the influence of the input dimension hyperparameter and LSTM architecture on performance. The results suggest that the optimal input dimension ranges from 7 to 12 values, and deeper LSTM models may not guarantee improved performance. We also investigated the effects of both preprocessing and hyperparameter optimization in improving the overall results. Our study underscores the importance of carefully considering the input dimension hyperparameter, LSTM architecture, and hyperparameter tuning in volatility prediction tasks.

3.2 Introduction

Volatility modeling has long been a cornerstone in financial mathematics, with its roots tracing back to early stochastic models like Brownian motion Bachelier [1900], Jarrow

et al. [2004]. Traditional models, such as the ARCH family, have been widely used but come with limitations, particularly in capturing the full range of empirical observations associated with asset price volatility Engle [1982b], Bollerslev [1986].

In recent years, Neural Networks (NNs) have gained traction in financial econometrics, offering promising results in various applications ranging from bond rating to stock price prediction Dutta and Shekhar [1988], Kamijo and Tanigawa [1990]. However, their application in volatility forecasting has been limited, often serving as a supplementary tool to traditional GARCH models Hajizadeh et al. [2012], Maciel et al. [2016]. Neural Networks (NN) have been extensively employed for forecasting tasks, such as predicting stock prices Khan [2011] or volatility with additional input Bucci [2020]. Some studies have combined conditional volatility models with NN Arnerić et al. [2014]. The performance results of these works in volatility prediction tasks have been mixed. In some instances, nonparametric models, including NN, have shown poor forecasting performance for out-of-sample tests Clements and Krolzig [1998], Pavlidis et al. [2012]. In Vortelinos [2017], it was demonstrated that feed-forward NN approximation is insufficient. Moreover, Vortelinos [2017], Bucci et al. [2017], Miura et al. [2019] reported mixed forecast accuracy for out-of-sample realized volatility using NN. Nonetheless, promising results for out-of-sample realized volatility forecasts have been provided by Rosa et al. [2014], Miura et al. [2019]. Implied and realized volatility forecasting tasks were investigated in Hamid and Iqbal [2004], which showed comparable NN performance for realized volatility. Complex neural network architectures, such as Jordan Neural Networks (JNN), have shown potential in volatility forecasting Arnerić et al. [2018].

Advancements in machine learning have introduced Long Short-Term Memory (LSTM) networks, which are particularly adept at capturing long-term dependencies in time-series data Hochreiter and Schmidhuber [1997c]. While LSTMs have been applied to volatility forecasting, existing studies often do not delve deeply into the nuances of LSTM architecture or the selection of appropriate hyperparameters Rosa et al. [2014], Miura et al. [2019], Bucci [2020], Bucci et al. [2017].

This gap in the literature brings us to our research. While machine learning models, particularly LSTMs, hold promise for volatility forecasting, there is a lack of

comprehensive studies that explore the impact of architecture and hyperparameters on their performance.

Our study aims to fill this gap by analyzing LSTM-based models for volatility forecasting. We focus on optimizing the architecture and hyperparameters to enhance the model's predictive capabilities. Specifically, we conduct a search of hyperparameter space for LSTM models for volatility tasks. Moreover, we introduced hyperparameter optimization for input dimension (LSTM-RV) for realized volatility forecasting tasks.

3.3 Preliminaries

3.3.1 Baseline Volatility Models

This section provides a brief overview of widely used models for measuring and forecasting volatility, primarily based on historical behavior.

The GARCH family of models estimates historical volatility or conditional variance Bollerslev [1986]. These models account for clustering effects and have been extended into versions like Exponential GARCH, GJR-GARCH, and TGARCH. However, an increase in complexity does not necessarily ensure better results and can make calculations more intricate.

The HAR-RV model, introduced by Corsi [2009], assumes that agents in financial markets have differing perceptions of volatility depending on their investment horizons. The model comprises an additive cascade of partial volatilities generated at different time horizons following an autoregressive process. This approach offers a stable and accurate estimate for realized volatility 2.16.

The ARIMA model facilitates the modeling of integrated or difference-stationary time series. It involves assessing the stationarity of the series, transforming it by taking differences, and constructing an ARMA model for the transformed series Box et al. [2015]. However, ARIMA models may fail to capture the heteroscedastic volatility properties commonly observed in financial time series.

The EWMA method calculates volatility by weighting individual periodic RV and assigning greater weight to recent observations. Though it serves as a practical benchmark, EWMA is considered simplistic when compared to more advanced volatility

models.

3.3.2 LSTM Input Dimension Hyperparameter

A neural network can be envisioned as a layered structure of interconnected neurons. Recurrent neural networks (RNNs) are a class of neural networks that utilize previous outputs as inputs, enabling them to handle sequential data. RNN neurons possess a cell state or memory; inputs are processed according to this internal state, achieved through the recurrent mechanism. RNNs contain repeating activation modules of layers that allow them to store information; however, this storage capacity is often limited 2.19, 2.19. RNNs frequently suffer from vanishing gradient issues, which can cause model training to become exceedingly slow or cease entirely. LSTM is a specific cell type within a recurrent neural network designed to capture long-term dependencies in data and address the exploding or vanishing gradient problem Hochreiter and Schmidhuber [1997a]. This achievement is made possible due to the cell state and the interaction of four gates 2.22,2.23,2.24,2.25. The ability to selectively add or remove information from the cell state, meticulously regulated by gate structures, is a crucial distinction from simple RNNs. LSTM cells contain an additional state, which helps to internally retain input memory 2.26, making them particularly suited for solving problems related to sequential data. The cell state conveys relevant information throughout the entire sequence chain. This state reflects the corresponding information during the entire series processing, allowing data from earlier time steps to participate in later time steps.

The construction of an NN typically involves the enumeration and identification of the optimal architecture that minimizes the error. In Panchal et al. [2010], authors propose information criteria to aid in determining an appropriate neural network structure. However, the most common approach remains to train various numbers of architectures by some search algorithm and select the one with the lowest error Bergstra and Bengio [2012], Hutter et al. [2011]. This iterative process enables researchers and practitioners to develop neural networks tailored to specific tasks, ultimately enhancing the performance and accuracy of their models Bergstra and Bengio [2012], Snoek et al. [2012].

For our steady, we consider a univariate time series; only one feature is used as input. Therefore, the input dimension will be equal to one. However, for LSMT with optimized dimensionality hyperparameter (LSTM-RV), we consider a sliding window approach incorporating a sequence of past values as input for each time step.

Implementing input dimension hyperparameter optimization can help better adapt to varying data characteristics. In this optimized architecture, the input matrix $\mathbf{X} \in R^{m \times n}$ is utilized, where m represents the number of input samples, and n is the optimized input dimension length of each t input. This optimized hyperparameter allows the LSTM model to capture relevant information across different time scales efficiently.

Leveraging this optimized input dimension, the LSTM model can learn complex patterns and dependencies in sequential data more effectively, enhancing its predictive capabilities.

3.4 Experiments

This study investigates the performance of predicting realized volatility across various financial assets, including stocks, index, and cryptocurrencies. We also assess the impact of data granularity on model performance. This section describes the experiments conducted, including the accuracy measures used for evaluation, the data, and the hyperparameter optimization and data preprocessing steps involved in setting up the LSTM model and other models.

Mean Squared Error (MSE) minimizes the average squared deviations between actual and predicted values, providing a more accurate estimate of the distance between them. Mean Absolute Error (MAE) uses absolute values to estimate this distance, offering robustness against outliers 2.50. Additionally, Root Mean Squared Error (RMSE) 2.51 and Mean Absolute Percentage Error (MAPE) are utilized to assess accuracy.

The Diebold-Mariano test Diebold [2015] compares the forecast accuracy of two models, with the null hypothesis stating equal accuracy. The alternative hypothesis may claim differing levels of accuracy or one model's superiority over the other.

Value at Risk (VaR) Jorion [2000] estimates the maximum expected loss for a given period and specified probability. VaR quantifies potential losses in a portfolio over a fixed period, with common time horizons of one, five, or ten days and risk levels set at 95 or 99 percent.

3.4.1 Data

This study examines the application of neural networks (NN) for estimating and predicting realized volatility across various market structures, specifically focusing on indices, stocks, and cryptocurrencies. We also analyze the effects and dependencies of data granularity, as shown in Table 3.1 and Table 3.2.

We calculate the daily Realized Volatility (RV) for the stock market data based on 1-minute price observations. However, for GARCH-family models, returns are computed using the last daily closing price. Consequently, our experimental data set comprises intraday returns and their corresponding RV values. The data set is divided into training, validation, and testing. The validation and testing samples for stocks consist of 252 data points, equivalent to one trading year.

Next, we investigate the S&P 500 index data set, which corresponds to daily price data, encompassing 17,923 price data points for calculating 815 RV observation points analogous Bucci [2020]. The methodology is analogous to stock data calculations. However, for this case, we compute monthly RV based on daily log returns instead of daily RV based on a 1-minute time frame for stock data. We use 245 data points for both the validation and test data sets.

Lastly, we explore cryptocurrency data, specifically the (Bitcoin-USD) BTCUSDT and Ethereum-USD (ETHUSDT) pairs. Bitcoin price data corresponds to a 1-minute time frame, while Ethereum data is obtained from a high-frequency data set corresponding to a 1-second time frame. Similar to the stock and index data, cryptocurrency data is divided into three parts: training, validation, and testing. The validation and test samples are the same size as the stock data set validation and test sets, consisting of 252 data points each.

In summary, our study investigates the performance of neural networks in estimating and predicting realized volatility across distinct market structures and time frames. By

analyzing stocks, indices, and cryptocurrencies, we aim to understand the dependencies and effects of data granularity on the effectiveness of neural network-based models in forecasting volatility. The data sets are partitioned into training, validation, and testing samples, allowing for a comprehensive evaluation of the proposed models in various financial contexts.

Table 3.1: Description of Asset Types and Aggregation Scales

Asset	Name	Time Frame
Stock	Dell Inc.	1 minute
Index	S&P500	1 day
Crypto.c	Bitcoin-USD	1 minute
Crypto.c	Ethereum-USD	1 second

Table 3.2: Detailed Description of Asset Data

Asset	Name	From	To	Price Points	RV Points
Stock	Dell Inc.	02.01.98	29.11.13	1,730,585	3,982
Index	S&P500	01.02.50	01.12.17	17,923	815
Crypto-c	Bitcoin-USD	31.12.11	22.04.19	3,837,857	2,670
Crypto-c	Ethereum-USD	01.02.20	21.05.20	8,237,492	2,650

Note: The table presents an overview of the data used in the analysis, including the asset type, the date range of the data, and the number of realized volatility (RV) points. Aggregation is done on different time scales. Aggregation is done on different time scales: Stock and BTCUSDT on a day, S&P500 on a month, and ETHUSDT on an hour.

3.4.2 Hyperparameter Optimization and Data Preprocessing

Our study explored various configurations concerning dynamic and objective outcomes through a trial-and-error approach to identify optimal setups for the given task. Identifying the best neural network configuration necessitates conducting multiple experiments with different hyperparameters, as described in Panchal et al. [2010]. Consequently, we established and analyzed the results of numerous training epochs, selecting an appropriate configuration based on low MSE and MAE metrics for the validation data set. Hidden size, the number of layers, loss function, activation function, batch size, dropout, epochs, and optimizer are hyperparameters that influence a model's

architecture, training process, and performance, affecting its ability to learn complex patterns, prevent overfitting, and optimize accuracy[1]. In this study, we optimize the listed hyperparameters to find the most appropriate setting for our task.

A key aspect of our design involved searching for an optimal input dimension hyperparameter of LSTM-RV to capture relevant information across different time horizons. LSTM possesses a state layer C_t, which practically assists in capturing long dependencies besides an input dimension. Consequently, LSTM should be adequate for any reasonable input dimension. However, appropriately defining the input dimension hyperparameter is vital for achieving high accuracy in this task.

Notably, considering HAR or ARIMA models requires defining a lag. Determining how long the series must contain the necessary data to forecast the next prediction horizon is essential. However, differentiating price data results in the loss of certain data properties (since they are contained in the price) in order to obtain a stationary series. This might be a key factor in LSTM's success, as a series with a length of n lags is preserved due to the cell memory mechanism Laptev et al. [2017].

Another crucial hyperparameter to consider is the activation function. Limited research has examined the relationship between various activation functions and their impact on performance for volatility prediction tasks. We evaluated several potential options for our objective, as outlined in Table 3.3. Furthermore, another vital hyperparameter is the optimizer. While the commonly implemented Adam optimizer is widely used, it may not be suitable for all tasks. However, as we demonstrate later, the appropriate selection of activation functions and optimizers can substantially influence the results.

Standardization is an essential preprocessing step for training LSTM models, which can often enhance their efficiency. To this end, we employed a min-max scaling approach for normalizing input data within the range of 0 to 1. We deliberately avoided using augmented approaches that incorporated external information.

Furthermore, we optimized the parameter α of the EWMA to improve MSE and MAE accuracy measures. We also conducted an optimization procedure for the HAR-RV model targeting the MSE and MAE metrics, while retaining the standard

[1]The hyperparameter notations are aligned with the PyTorch library

model with parameters 1, 5, and 22. In addition, ARIMA GARCH and GJR-GARCH were optimized based on information criteria, as suggested in Burnham and Anderson [2004].

Table 3.3: The Table of LSTM and LSTM-RV Hyperparameters

Hyperparameters	Values
Input Dimension*	[1-50]
Hidden size	[1-100]
Number of layers	[5-20]
Dropout	[0.01, 0.05, 0.1, 0.2, 0.3]
Activation fn	[linear, ReLU, SoftMAX, Tanh]
Loss fn	[MAE, MSE, HUBER]
Epochs	[1-10,20,30,50,100,1000]
Batch size	[1,2,4,8,16,32,64]
Optimizer	[RMSprop, SGD, ADAM]

Note: The hyperparameter optimization procedure is only for the LSTM-RV model.
* Indicates that the hyperparameter is specific to the LSTM-RV model.

3.5 Results

In this empirical study, we aimed to address the following research questions: How does the architectural configuration of LSTM networks, including factors such as the number of layers and hidden units, affect predictive performance? Can a hyperparameter-optimized LSTM model surpass established benchmarks like the HAR and GARCH models in realized volatility forecasting? What is the influence of the dimension hyperparameter on an LSTM model efficiency and accuracy for this task, and what value maximizes performance?

In this section, we present the results for three types of data sets considered in our experiment: stock, indices, and cryptocurrencies, along with three Realized Volatility (RV) aggregations: monthly, daily, and hourly. We begin by discussing the performance of the examined models and providing detailed comparisons. Standard accuracy measures were employed for one-step-ahead prediction using RMSE and MAE metrics[2].

[2]The results presented in a table have been scaled for ease of readability. To obtain the actual values, please apply the scaling factor provided in a table header.

Subsequently, we assessed the significance of improvement using the Diebold-Mariano test and Value at Risk metrics.

Table 3.4: Hyperparameter Sets for Model Convergence

Hyperparameters	h1 set	h2 set	h3 set
Input Dimension	1	8	20
Layers	2	2	2
Hidden size	20	20	20
Epochs	500	500	500
Optimizer	Adam	Adam	Adam

Note: This table summarizes the hyperparameters used in figure 3.2 for the models.

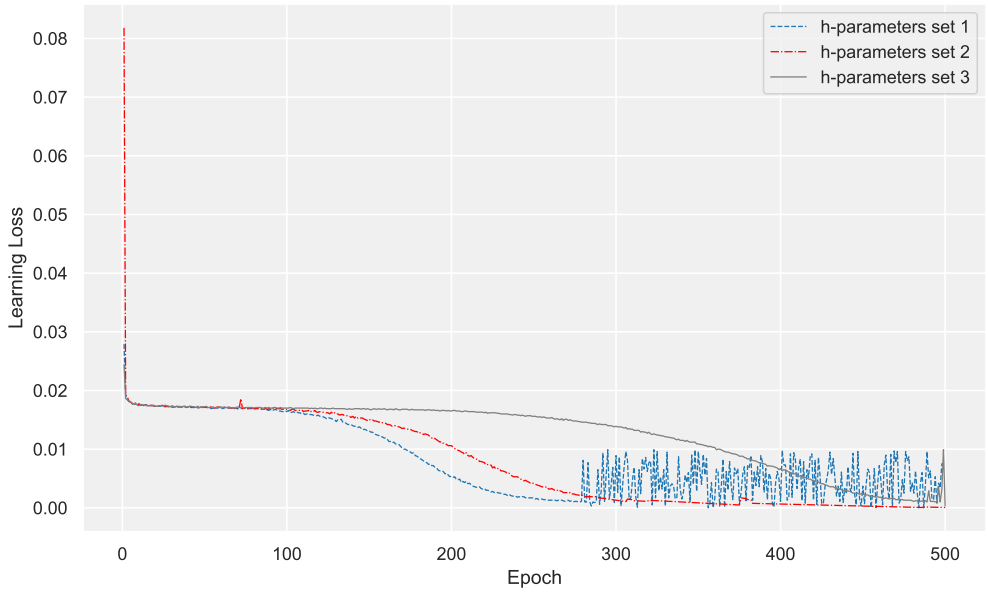

Figure 3.1: The following plot illustrates the Convergence Rate of MSE loss for three different input dimension hyperparameters. The Dell Inc. stock training data set.

The best performance in the stock market was demonstrated by LSTM-RV. LSTM-RV achieved the highest MAE accuracy for the validation data set Table 3.10. However, for RMSE hyperparameter-optimized for MSE HAR model surpassed other models. Nevertheless, in terms of out-of-sample RMSE and MAE accuracy, the LSTM-RV model outperformed competing models. The DM test also indicated

Input Dimensions	Input Length								
	1	*5*	*10*	*15*	*20*	*30*	*40*	*50*	*100*
1	21.450	18.175	16.830	16.998	13.500	13.896	15.096	16.010	17.220
2	20.061	17.273	15.659	15.647	13.121	13.523	14.967	16.042	17.258
3	18.963	16.249	14.488	14.422	12.922	12.891	14.412	15.494	16.879
4	18.765	15.961	14.367	14.429	12.742	13.197	14.570	15.425	16.907
5	20.043	17.199	15.725	15.883	13.544	13.291	14.905	15.810	17.508
6	18.515	16.046	14.088	13.784	12.721	12.579	14.468	15.525	16.859
7	17.274	15.057	13.338	13.095	12.450	12.460	14.151	15.161	16.659
8	15.610	13.596	11.760	11.796	**11.596**	11.696	13.396	14.396	15.596
9	17.347	15.924	13.833	13.570	12.428	12.891	15.066	15.953	17.358
10	17.501	16.562	14.923	14.761	13.327	13.789	15.497	16.361	18.469
11	17.096	16.124	14.799	14.666	13.547	13.875	15.624	16.505	18.063
12	16.654	15.699	14.233	14.172	13.403	13.980	15.467	16.342	17.988
13	15.539	14.938	13.674	13.734	13.078	13.651	15.689	16.318	17.846
14	17.430	16.100	14.154	13.590	12.484	13.095	15.190	16.272	17.649
15	16.348	16.642	14.998	14.818	13.331	14.027	15.664	16.525	18.761
16	16.204	16.230	14.878	14.803	13.844	13.941	15.862	16.718	18.135
17	17.220	16.195	14.877	14.775	13.800	14.636	15.993	16.878	18.763
18	16.462	15.780	14.762	14.630	14.003	14.883	16.242	16.998	19.040
19	15.570	15.705	14.988	14.994	13.746	14.840	16.750	17.437	19.197
20	14.910	16.229	15.304	15.467	13.964	15.330	17.540	17.990	19.574

Figure 3.2: The following plot illustrates the input dimension and input length interconnection loss for the Dell Inc. stock validation data set.

a significant improvement in forecast quality compared to the closest competitor, Table 3.11.

To compare performance under different market conditions, we investigated the S&P 500 index data set, where data granularity is more limited than stock data sets or cryptocurrencies. We deliberately consider this particular data set to enable comparison with the results reported in Bucci [2020]. In the validation phase, the model did not yield the best results for RMSE and MAE, although the gap was not significant at 0.1% accuracy, ranking it as the second most accurate model Table 3.7. However, the LSTM-RV model exhibited good RMSE accuracy with a substantial margin for the out-of-sample results Table 3.12. The DM test also indicated statistically significant forecast accuracy. This is noteworthy, considering the complexity of training a network with numerous parameters and the need for large amounts of training data compared to econometric models.

The LSTM-RV model also demonstrated high accuracy performance in the cryptocurrency market in both validation Table 3.8, Table 3.9, and out-of-sample tests

Table 3.5: Performance Metrics for Number of Layers of LSTM-RV on Different Validation Data Sets

Number of Layers	Dell Inc.		S&P500		BTCUSDT		ETHUSDT	
	MAE^*	MSE^*	MAE^*	MSE^*	MAE^*	MSE^*	MAE^*	MSE^*
1	7.462	0.134	8.666	0.426	7.229	0.163	6.414	0.158
2	6.936	0.134	9.482	0.452	7.804	0.163	5.525	0.067
3	7.802	0.232	9.626	0.464	8.139	0.261	6.065	0.068
4	8.285	0.323	9.647	0.507	8.958	0.334	6.217	0.156
5	8.339	0.325	9.682	0.513	8.998	0.341	6.277	0.162
6	8.384	0.329	9.685	0.518	9.054	0.351	6.284	0.168
7	8.416	0.331	9.760	0.520	9.054	0.353	6.327	0.171
8	8.486	0.325	9.817	0.524	9.186	0.359	6.368	0.179
9	8.525	0.336	9.685	0.513	9.229	0.363	6.454	0.179
10	8.621	0.341	9.967	0.527	9.306	0.369	6.530	0.185
20	9.239	0.427	10.605	0.579	9.943	0.380	6.701	0.202

Note: The table presents a comparative analysis of performance metrics across a numbers of layers for multiple data sets. The metrics employed are the scaled Mean Absolute Error (MAE^*), calculated as $MAE \times 10^3$, and the scaled Mean Squared Error (MSE^*), calculated as $MSE \times 10^3$. The LSTM-RV architecture specifications are outlined in tables of validation data set results for each data set under consideration.

Table 3.14, Table 3.16. It should be noted, however, that the Bitcoin out-of-sample HAR-RV performance is on par with the LSTM-RV model. Our findings suggest that dimensionality hyperparameterization of LSTM-RV is crucial, as these parameters increase with more granular data.

In the case of high-frequency data, the LSTM-RV model yielded more stable results. Conversely, the standard LSTM exhibited reduced performance accuracy.

Table 3.6: Performance Metrics for Various Models on the Dell Inc. Stock Validation Data Set

Model	Description	Parameters	RMSE (10^3)	MAE (10^3)	MAPE
EWMA MSE	alpha	0.12	11.0926	7.7043	30.4275
EWMA MAE	alpha	0.15	11.1070	7.6868	30.1312
HAR MSE	d,w,m:	1, 12, 26	**10.9870**	7.8279	31.8926
HAR MAE	d,w,m:	1, 11, 28	11.0194	7.8179	31.8276
HAR	d,w,m:	1, 5, 22	11.1143	7.9304	32.2523
ARIMA	p,d,q:	1, 1, 2	11.5286	8.1161	32.3217
GARCH	order	1,1	13.4846	8.9360	59.3186
GJR-GARCH	order	1,1	13.4841	8.9330	59.3172
LSTM	input-dim	20	13.9635	9.0714	27.9306
LSTM-RV	input-dim	8	11.5960	**6.9357**	23.2722

Note: The table presents the performance metrics for a 1-step ahead prediction on the Dell Inc. stock validation data set. The data set size is 252 points. LSTM architecture: Hidden size = 20; Number of layers = 2; Loss function = MSE; Dropout = 0.1; Activation function = ReLU; Batch size = 32; Epochs = 500; Optimizer = Adam.

Table 3.7: Performance Metrics for Various Models on S&P 500 Validation Data Set

Model	Description	Parameters	RMSE (10^3)	MAE (10^3)	MAPE
EWMA MSE	alpha	0.2	20.4632	9.3206	23.2058
EWMA MAE	alpha	0.28	20.5475	9.2023	22.5636
HAR MSE	d,w,m	3, 4, 55	**20.1926**	8.5367	20.7522
HAR MAE	d,w,m	3, 15, 30	20.2359	**8.4996**	20.4421
HAR	d,w,m	1, 5, 22	21.2955	8.9237	20.7829
ARIMA	p,d,q	1, 1, 1	23.2479	10.147	24.5666
GARCH	order	1,1	28.7892	23.970	72.8169
GJR-GARCH	order	1,1	28.8595	23.847	71.8954
LSTM	input-dim	20	22.2736	26.547	66.4353
LSTM-RV	input-dim	12	20.6301	8.6655	20.6121

Note: The table presents the performance metrics for a 1-step ahead prediction on the S&P 500 validation data set. The data set size is 252 points. LSTM architecture: Hidden size = 20; Number of layers = 1; Loss function = MSE; Dropout = 0.01; Activation function = ReLU; Batch size = 32; Epochs = 100; Optimizer = Adam.

Table 3.8: Performance Metrics for Various Models on BTCUSDT Validation Data Set

Model	Description	Parameters	RMSE (10^3)	MAE (10^3)	MAPE
EWMA MSE	alpha	0.42	13.1625	7.9259	21.6890
EWMA MAE	alpha	0.83	13.9643	7.6130	21.1396
HAR MSE	d,w,m	1, 5, 45	13.1465	8.9144	29.9428
HAR MAE	d,w,m	1, 4, 18	13.2281	8.7833	29.1916
HAR	d,w,m	1, 5, 22	13.1945	8.8089	29.3820
ARIMA	p,d,q	3, 1, 3	15.8378	8.5929	23.2321
GARCH	order	1,1	18.2987	14.584	61.8887
GJR-GARCH	order	1,1	17.6519	13.770	58.7079
LSTM	input-dim	20	15.7557	9.4235	23.9942
LSTM-RV	input-dim	8	**12.7771**	**7.2293**	21.0681

Note: The table presents the performance metrics for a 1-step ahead prediction on the BTCUSDT cryptocurrency for a 1-step ahead prediction. The data set size is 252 points. LSTM architecture details are as follows: Hidden size = 20; Number of layers = 1; Loss function = MSE; Dropout = 0.1; Activation function = ReLU; Batch size = 32; Epochs = 100; Optimizer = Adam.

Table 3.9: Performance Metrics for Various Models on ETHUSDT Validation Data Set

Model	Description	Parameters	RMSE (10^3)	MAE (10^3)	MAPE
EWMA MSE	alpha	0.5	8.5095	5.7709	22.5619
EWMA MAE	alpha	0.7	8.6151	5.6981	22.3058
HAR MSE	d,w,m	1, 5, 30	8.3400	6.1236	26.7363
HAR MAE	d,w,m	1, 4, 24	9.0411	6.0497	22.9636
HAR	d,w,m	1, 5, 22	8.3460	6.1170	26.7087
ARIMA	p,d,q	1, 1, 2	8.6465	5.7329	22.8089
GARCH	order	1,1	10.019	6.0207	28.4042
GJR-GARCH	order	1,1	9.5977	5.9829	28.7266
LSTM	input-dim	20	15.698	11.812	40.9822
LSTM-RV	input-dim	7	**8.1961**	**5.5248**	22.8914

Note: The table summarizes the performance metrics for various models on the ETHUSDT cryptocurrency for a 1-step ahead prediction. The data set size is 252 points. LSTM architecture details are as follows: Hidden size = 50; Number of layers = 2; Loss function = MSE; Dropout = 0.1; Activation function = ReLU; Batch size = 32; Epochs = 100; Optimizer = RMSProp.

Table 3.10: Out-of-Sample Performance Metrics for the Dell Inc. Stock (Part 1)

Model	RMSE (10^3)	MAE (10^3)	MAPE	VaR
EWMA MSE	11.8803	7.7360	43.3397	0.2544
EWMA MAE	11.7882	7.5838	41.6260	0.2544
HAR MSE	11.7178	8.2606	53.1560	0.1533
HAR MAE	11.6789	8.2126	52.9801	0.1533
HAR	11.6789	8.2126	52.9801	0.1533
ARIMA	12.0444	7.4324	40.7973	0.1986
GARCH	18.2725	13.720	62.4810	0.2140
GJR-GARCH	18.0589	11.716	60.0866	0.2028
LSTM	14.1276	9.0725	56.4351	0.1196
LSTM-RV	**11.4416**	**6.8530**	35.8922	0.1280

Note: The table summarizes the out-of-sample performance metrics for various models on the Dell Inc. stock for a 1-step ahead prediction. The data set size is 252 points. VaR 10 days indicates the 10-day Value at Risk measure.

Table 3.11: Out-of-Sample Performance Metrics for the Dell Inc. stock (Part 2)

Model	DM.t MSE[3]	P.val MSE[3]	DM.t MAE[3]	P.val MAE[3]
EWMA MSE	0.4295	0.6679	3.6767	0.0002
EWMA MAE	0.1543	0.8774	3.2005	0.0015
HAR MSE	0.0507	0.9595	5.5384	0.0000
HAR MAE	0.1722	0.8633	5.5707	0.0000
HAR	0.1722	0.8633	5.5707	0.0000
ARIMA	8.7439	0.0000	12.052	0.0000
GARCH	1.6996	0.0904	4.6902	0.0000
GJR-GARCH	1.4385	0.1515	4.1625	0.0000
LSTM	4.7601	0.0000	6.7555	0.0000
LSTM-RV	-	-	-	-

Note: This table is a continuation of Table 3.10 and provides additional out-of-sample performance metrics for various models on the Dell Inc. stock for a 1-step ahead prediction.

[3] The asterisks and scientific notation represent statistical significance and small p-values. DM.t MSE refers to the Diebold-Mariano test for Mean Squared Error, and DM.t MAD denotes the Diebold-Mariano test for Mean Absolute Deviation.

Table 3.12: Out-of-Sample Performance Metrics for S&P 500 Index (Part 1)

Model	RMSE (10^3)	MAE (10^3)	MAPE	VaR
EWMA MSE	22.7377	14.8143	32.0655	0.2807
EWMA MAE	21.8396	14.0323	30.1342	0.2805
HAR MSE	22.2281	13.4078	28.1917	0.3185
HAR MAE	22.1787	13.2569	27.3338	0.3177
HAR	21.2263	**12.7762**	26.790	0.3185
ARIMA	21.4788	14.1858	30.8716	0.2823
GARCH	31.7454	24.1938	37.0463	0.1440
GJR-GARCH	31.9215	24.5954	36.9429	0.1446
LSTM	23.5619	14.4471	32.9160	0.1511
LSTM-RV	**20.4150**	12.8474	28.505	0.3374

Note: The table summarizes the out-of-sample performance metrics for various models on the S&P 500 index for a 1-step ahead prediction. The data set size is 252 points. VaR 10 days indicates the 10-day Value at Risk measure.

Table 3.13: Out-of-Sample Performance Metrics for S&P 500 Index (Part 2)

Model	DM.t MSE[3]	P.val MSE[3]	DM.t MAD[3]	P.val MAD[3]
EWMA MSE	3.4105	0.0007	4.3333	0.0000
EWMA MAE	2.9263	0.0037	2.7277	0.0068
HAR MSE	2.2882	0.0229	1.7231	0.0861
HAR MAE	2.2819	0.0233	1.3088	0.1918
HAR	2.5911	0.0101	-0.3437	0.7313
ARIMA	3.0891	0.0022	6.0096	0.0000
GARCH	4.5905	0.0000	18.972	0.0000
GJR-GARCH	4.6029	0.0000	19.613	0.0000
LSTM	3.4729	0.0000	12.181	0.0000
LSTM-RV	-	-	-	-

Note: This table is a continuation of Table 3.12 and provides additional out-of-sample performance metrics for various models on the S&P 500 index for a 1-step ahead prediction.

[3] The asterisks and scientific notation represent statistical significance and small p-values. DM.t MSE refers to the Diebold-Mariano test for Mean Squared Error, and DM.t MAD denotes the Diebold-Mariano test for Mean Absolute Deviation.

Table 3.14: Out-of-Sample Performance Metrics for BTCUSDT (Part 1)

Model	RMSE (10^3)	MAE (10^3)	MAPE	VaR
EWMA MSE	16.6472	8.3999	20.6981	0.3956
EWMA MAE	16.8880	8.7378	21.7385	0.3956
HAR MSE	16.0889	9.0112	25.4136	0.2805
HAR MAE	16.0846	8.9371	25.1927	0.2793
HAR	16.1576	8.9817	25.2597	0.2795
ARIMA	19.6889	9.6716	23.0919	0.2598
GARCH	34.3660	17.607	40.3906	0.1491
GJR-GARCH	36.1548	18.061	40.9336	0.1495
LSTM	22.4670	15.245	40.6017	0.1493
LSTM-RV	**15.6290**	**7.8785**	20.3574	0.2583

Note: The table summarizes the out-of-sample performance metrics for various models on BTCUSDT cryptocurrency for a 1-step ahead prediction. The data set size is 252 points. VaR 10 days indicates the 10-day Value at Risk measure.

Table 3.15: Out-of-Sample Performance Metrics for BTCUSDT (Part 2)

Model	DM.t MSE[3]	P.val MSE[3]	DM.t MAD[3]	P.val MAD[3]
EWMA MSE	1.4792	0.1403	1.8250	0.0691
EWMA MAE	0.6404	0.5224	2.0189	0.0445
HAR MSE	1.3917	0.1652	5.5401	0.0000
HAR MAE	1.2148	0.2255	5.2337	0.0000
HAR	1.6145	0.1076	5.6381	0.0000
ARIMA	3.7090	0.0002	7.6181	0.0000
GARCH	3.8704	0.0001	7.2925	0.0000
GJR-GARCH	3.7308	0.0002	7.9830	0.0000
LSTM	4.0497	0.0001	6.8073	0.0000
LSTM-RV	-	-	-	-

Note: This table is a continuation of Table 3.14 and provides additional out-of-sample performance metrics for various models on BTCUSDT cryptocurrency for a 1-step ahead prediction.

[3] The asterisks and scientific notation represent statistical significance and small p-values. DM.t MSE refers to the Diebold-Mariano test for Mean Squared Error, and DM.t MAD denotes the Diebold-Mariano test for Mean Absolute Deviation.

Table 3.16: Out-of-Sample Performance Metrics for ETHUSDT (Part 1)

Model	RMSE (10^3)	MAE (10^3)	MAPE	VaR
EWMA MSE	12.0455	6.4798	26.0383	0.2472
EWMA MAE	12.3271	6.6222	26.8891	0.2469
HAR MSE	11.4807	6.6540	29.3306	0.1787
HAR MAE	12.1568	7.1052	31.6274	0.1813
HAR	**11.4797**	6.6583	29.3342	0.1787
ARIMA	12.3204	7.3057	31.4821	0.1801
GARCH	76.4503	54.010	39.247	0.0931
GJR-GARCH	71.6640	51.818	39.280	0.0931
LSTM	16.5935	10.357	37.0491	0.0946
LSTM-RV	11.6203	**6.0079**	22.6919	0.1461

Note: The table summarizes the out-of-sample performance metrics for various models on ETHUSDT cryptocurrency for a 1-step ahead prediction. The data set size is 252 points. VaR 10 days indicates the 10-day Value at Risk measure.

Table 3.17: Out-of-Sample Performance Metrics for ETHUSDT (Part 2)

Model	DM.t MSE[3]	P.val MSE[3]	DM.t MAD[3]	P.val MAD[3]
EWMA MSE	0.6170	0.5377	1.6013	0.1105
EWMA MAE	0.5746	0.5660	1.8285	0.0686
HAR MSE	-0.2493	0.8033	2.2370	0.0261
HAR MAE	0.7861	0.4325	3.0047	0.0029
HAR	-0.2573	0.7971	2.2608	0.0246
ARIMA	2.7745	0.0059	7.1878	0.0000
GARCH	4.6473	0.0000	14.769	0.0000
GJR-GARCH	5.0685	0.0000	15.610	0.0000
LSTM	3.1206	0.0020	10.646	0.0000
LSTM-RV	-	-	-	-

Note: This table is a continuation of Table 3.16 and provides additional out-of-sample performance metrics for various models on ETHUSDT cryptocurrency for a 1-step ahead prediction.

[3] The asterisks and scientific notation represent statistical significance and small p-values. DM.t MSE refers to the Diebold-Mariano test for Mean Squared Error, and DM.t MAD denotes the Diebold-Mariano test for Mean Absolute Deviation.

3.6 Conclusion

This study aimed to investigate the efficacy of LSTM recurrent neural networks in predicting volatility and to compare their performance with well-established econometric models across various markets, including stocks and the increasingly popular cryptocurrency markets. The focus was on using realized volatility as the target variable.

A critical contribution of this paper is adding the procedure of input dimensions hyperparameterization, which considerably enhanced the model's accuracy. It was determined that an efficient range for this hyperparameter lies between 7 and 12 periods, enabling LSTM-RV to outperform robust models in the domain. Furthermore, out-of-sample accuracy tests revealed that LSTM-RV provided substantial benefits across different market types.

Moreover, we examined the impact of LSTM architecture - a number of layers and a number of neurons in each layer. We discovered that each of them significantly influences the final results, and typically, deeper structures do not offer substantial improvements over less deep LSTM models.

Additionally, it was found that hyperparameter tuning is essential for further performance improvement. Our findings underscore the importance of considering LSTM model architecture, the input dimensions hyperparameter, preprocessing steps, and hyperparameter tuning.

Despite the effectiveness of neural networks, challenges persist, such as the black-box nature of the model, which impedes the analysis of relationships within the data. Another challenge lies in the high number of parameters to be learned during training compared to HAR or EWMA models. Nonetheless, contemporary computing power permits relatively swift training of RNN models for volatility prediction tasks.

Chapter 4

σ-Cell

4.1 Motivation

Econometric price volatility models have seen considerable empirical and theoretical progress over the past fifty years. Despite this robust evolution, the integration of volatility modeling within deep learning, especially Recurrent Neural Networks (RNNs), remains underexplored. This paper introduces a step in this direction with the σ-Cell type models, a particular RNNs cells design. The σ-Cell integrates the Generalized Autoregressive Conditional Heteroskedasticity (GARCH) principles, a stochastic layer, and time-varying parameters to address the dynamic nature of financial volatility. This union creates a generative network that embodies the joint distribution of the stochastic volatility process while offering an approximation of the latent variables' conditional distribution, given the observables. Such a design adeptly captures the temporal intricacies of volatility in financial time series. We utilize a log-likelihood-based loss function and develop a particular Adjusted-Softplus activation function to enhance the model's efficacy further. Experimental results on synthetic and real-world data validate our approach, indicating superior out-of-sample forecasting performance compared to traditional GARCH family models and the Stochastic Volatility model. This work represents a significant step towards integrating domain knowledge with the capabilities of neural networks.

4.2 Introduction

Price volatility serves as a cornerstone in econometrics, offering a lens to understand the variability of financial asset prices. Over its half-century history, the field has seen significant advancements in probability and statistics Bachelier [1900]. While early stochastic models like Brownian motion and the Wiener process were pioneering, they were not comprehensive enough to capture all empirical observations related to asset price fluctuations Cont [2001].

In financial markets, volatility acts as a crucial indicator of risk. Its variations reflect the oscillations in asset prices, with increased volatility signaling heightened market risk. Given the critical role of implied volatility in pricing derivative instruments, a nuanced understanding is essential, particularly as the derivatives market expands Engle [1982b], Bollerslev [1986], Poon and Granger [2003].

However, classical models like ARCH and GARCH, despite their groundbreaking contributions, encounter limitations. These include the unobservability of intrinsic volatility and assumptions that may not align with real-world financial dynamics Poon and Granger [2003], Andersen and Bollerslev [1998], Hansen and Lunde [2005].

In recent years, Neural Networks (NNs) have gained traction in financial econometrics, offering promising results in various applications ranging from bond rating to stock price prediction Dutta and Shekhar [1988]. However, the application of Neural Networks in volatility forecasting has been limited, often serving merely as supplementary tools to traditional models Hajizadeh et al. [2012]. This chapter bridges the gap between traditional and modern approaches by introducing the σ-Cell. Our approach integrates the capabilities of Recurrent Neural Networks (RNNs) with the proven methodologies of GARCH and the theoretical foundation of latent stochastic processes. We evaluate the σ-Cell using synthetic data, the S&P 500 index, and the cryptocurrency pair of Bitcoin-USD (BTCUSDT) to demonstrate its predictive capabilities.

4.3 Preliminaries

Volatility stands as a fundamental measure in financial markets, representing the degree of variation of trading prices over time. The ARCH Model, introduced by Engle in 1982, identifies time-varying volatility within time series data, serving as the foundation for many subsequent models Engle [1982b], Bollerslev [1986]. The GARCH Model, Bollerslev's 1986 innovation, the GARCH model, extended the ARCH model by incorporating past conditional variances into current estimations, akin to the AR to ARMA model transition Bollerslev [1986]. Almost a decade after Bollerslev's foundational GARCH model, Engle and Kroner in 1995 introduced a multivariate extension. This allowed the model to handle multiple financial series simultaneously, making use of historical data dependencies in complex financial scenarios Bollerslev [1986], Engle and Kroner [1995].

SV models propose that volatility follows latent stochastic processes. Unlike deterministic models, current volatility in SV models is influenced by unseen processes, even with complete historical data. For example, Heston's model posits that the variance process is influenced by underlying factors Heston [1993]. While these models are theoretically rich, they often require assumptions that may not always align with empirical scenarios. The MCMC-based models framework has emerged to enhance volatility forecasting. These models, though powerful, demand extensive data and sometimes struggle with multivariate time series Kastner and Frühwirth-Schnatter [2014], Wu et al. [2013].

Volatility models delineate the behavior of volatility processes, aiding in the estimation and prediction of time series fluctuations. Given the reliance on historical data for forecasting pertinent quantities, we postulate that the conditional variance depends on past information. This dependence can be classified as either deterministic or stochastic, resulting in two distinct categories of volatility models.

4.3.1 Baseline Volatility Models

We employ the GARCH and GJR-GARCH models as deterministic baselines for comparison Bollerslev [1986], Engle [1982a]. The GARCH(1,1) model is especially

popular for its simplicity and effectiveness in capturing financial volatility Francq and Zakoian [2010]. The GJR-GARCH model adds complexity by accounting for asymmetric volatility reactions to market returns Glosten et al. [1993a]. These models serve as robust benchmarks for evaluating our proposed approach.

We also employ SV models, such as the Hull-White and Heston models, as stochastic baselines Hull and White [1987a], Heston [1993]. These models use stochastic differential equations to capture volatility dynamics. We include SV models for two main purposes: to benchmark our model against established stochastic frameworks and to test its robustness and predictive accuracy. This comparative analysis aims to provide a comprehensive evaluation of various volatility modeling approaches.

4.3.2 Recurrent Neural Networks

The world of financial volatility modeling is continuously evolving as researchers work on hybrid approaches that combine various approaches to model volatility and capture more complex patterns in financial time series data.

In the domain of volatility forecasting in financial markets, hybrid approaches have garnered significant attention for their potential to combine the strengths of different modeling techniques. Generally, these hybrid models can be classified into several categories: Statistical-Statistical Hybrids, Machine Learning-Machine Learning hybrids, Statistical-Machine Learning hybrids, and Ensemble Approaches.

Deep learning techniques, which have demonstrated prowess in fields like image and speech recognition, are now being harnessed to study volatility LeCun et al. [2015]. These models have shown immense potential in diverse applications, ranging from machine translation to pattern recognition Krizhevsky et al. [2012], Chorowski et al. [2015], Bahdanau et al. [2014].

NNs have been extensively employed for forecasting tasks, such as predicting stock prices Khan [2011] or volatility with additional input Bucci [2020]. Some studies have combined conditional volatility models with NNs Arnerić et al. [2014]. The performance results of these works in volatility prediction tasks have been mixed. Nonparametric models, including NNs, have shown poor forecasting performance for out-of-sample tests Clements and Krolzig [1998], Pavlidis et al. [2012]. In Vortelinos [2017], it

was demonstrated that feed-forward NNs approximation is insufficient. Moreover, Vortelinos [2017], Bucci et al. [2017], Miura et al. [2019] reported mixed forecast accuracy for out-of-sample realized volatility using NNs. Nevertheless, promising results for out-of-sample realized volatility forecasts had been provided by Rosa et al. [2014], Miura et al. [2019]. Implied and realized volatility forecasting tasks were investigated in Hamid and Iqbal [2004], which showed comparable NN performance for realized volatility. Complex neural network architectures, such as Jordan Neural Networks (JNN), have shown potential in volatility forecasting Arnerić et al. [2018].

RNNs are designed to model sequential data, thus making them suitable for volatility modeling. RNNs process information cyclically, taking into account both past and current inputs. The method RNNs use to relay information from one iteration to the next in the hidden layer is detailed mathematically in Zhang et al. [2021]. This involves defining the hidden and input states at time t, matrices representing input-to-hidden and hidden-to-hidden connections, and a bias. The aggregated data then interacts with an activation function to ensure compatibility with backpropagation, the equations for hidden and output states 2.19, 2.20.

However, hybrid models typically encapsulate modules, making them more versatile. This paper presents a novel approach, the σ-Cell, which is inspired by the integration possibility of domain knowledge and the efficiency of RNNs for forecasting in sequential data. The σ-Cell RNN is a specialized RNN cell designed for volatility modeling that integrates the principles of the GARCH, a stochastic approach, and the time-varying parameters' dynamic with the capabilities of RNNs.

4.4 σ-Cell RNNs Volatility Models

In this section, we introduce the estimation of conditional volatility in a time series using a modified GARCH process integrated with RNN dynamics. Given a time series $X = \{x_1, x_2, \ldots, x_T\}$, our primary aim is to ascertain the conditional volatility, σ_t, at each discrete time t. The volatility, σ_t^2, is characterized by the subsequent relation, equation 4.1.

$$\sigma_t^2 = F\left(x_t\right)_1 \cdot \sigma_{t-1}^2 + F\left(x_t\right)_2 \cdot \epsilon_t^2 + b \tag{4.1}$$

In equation 4.1, $F : \mathbb{R}^d \to \mathbb{R}^2$ denotes a function mapping the value x_t onto a vector in $\mathbb{R}^2$. Notably, unlike traditional GARCH$(1, 1)$ where parameters remain constant, the entities of this vector, denoted by $F\left(x_t\right)_1$ and $F\left(x_t\right)_2$, act as dynamic parameters, varying based on the data point x_t. And b is a constant term, providing an additional degree of flexibility to the model. The residual error ϵ_t at an instance t is computed as shown in equation 4.2, where $G : \mathbb{R}^d \to \mathbb{R}$ signifies a function that associates the observed value x_t with its residual 4.3.

$$\epsilon_t = x_t - G\left(x_t\right) \tag{4.2}$$

$$G(x_t) \sim N(0, \sigma_t^2) \tag{4.3}$$

Equation 4.1 defines the process in which our methodology operates. It delineates a GARCH-like mechanism tailored for the precise estimation of conditional volatility inherent in a provided time series. Central to this approach is the explicit estimation of volatility, σ_t, for every point t within the time series X, and dynamic volatility modeling wherein the model determines σ_t^2 by integrating the prevailing time series value, x_t, with the preceding volatility, σ_{t-1}^2, and the corresponding error term, ϵ_t^2. The function F is a pivotal element, offering a dynamic mapping from the current observation, x_t, to a bivariate vector, bestowing the model with real-time modulation of the GARCH parameters and thus affording enhanced adaptability. The error quantification term ϵ_t captures the difference between the actual and the anticipated values at time t, serving as a crucial metric to gauge the precision of the model.

For simplicity, in our model, the time series is assumed to be univariate with dimensionality $d = 1$, but in the general form, it could be multivariate. This dimensionality allows for capturing intricate patterns from multiple time series variables, enriching the model's capability in volatility estimation.

Our methodology provides a method to forecast volatility in multivariate time series data. By fusing the principles of the GARCH process with increased parameter

flexibility through function F. Based on this, subsequent sections delve into how this model can be seamlessly merged with RNNs, thus potentially benefiting GARCH and RNN architectures to enhance volatility predictions.

4.4.1 σ-Cell: Nonlinear GARCH-based

This section introduces the nonlinear GARCH-based RNN Cell, abbreviated as σ-Cell. This innovative approach combines the predictive power of the GARCH model, known in econometrics for its volatility forecasting capabilities, with the ability of recurrent neural networks to process and learn from sequential data.

The motivation for the σ-Cell comes from the need for better volatility management in sequences, especially in financial data. RNNs, capable of processing information across long sequences, are combined with GARCH principles to handle this challenge more effectively.

Another innovation of the σ-Cell is its introduction of nonlinearity into the traditional GARCH model using an activation function. This approach allows the model to detect more complex patterns in sequential data, significantly improving its effectiveness compared to linear models Franses and Van Dijk [1996]. The σ-Cell calculates conditional volatility at each time point using a nonlinear transformation, represented by ϕ. This transformation adjusts the weighted mix of squared past volatility and squared input, as expressed in equation 4.4:

$$\tilde{\sigma}_t^2 = \phi(\tilde{\sigma}_{t-1}^2 W_s + x_{t-1}^2 W_r + b_h) \tag{4.4}$$

$$\sigma_t^2 = \phi_o(\tilde{\sigma}_t^2 W_o + b_o) \tag{4.5}$$

In equation 4.4, σ_t^2 denotes the hidden state of the RNN at time t. The weight matrices W_s and W_r correspond to the previous volatility and input, respectively.

In the RNN described by equation 4.4, the weight matrices W_s and W_r each have parameters equal to the product of the input size and hidden size. In this specific case, our input is scalar, and we have chosen a small hidden size of 10, making the number of parameters for both W_s and W_r relatively small. The bias vector b_h has a number of parameters equal to the hidden size. In equation 4.5, the weight matrix W_o has

parameters corresponding to the product of the hidden size and output size, with our output consisting of just one value. The bias vector b_o has parameters equal to the output size. Summing up the parameters from all these elements, the total number of parameters in this system comes to 41.

The optimization of the σ-Cell RNN model is carried out in two phases, with each phase refining one set of weights to enhance the model's stability. This novel approach augments the traditional GARCH model with nonlinearity, better capturing complex volatility fluctuations. By integrating the σ-Cell, we expect to achieve a more effective model for forecasting volatile patterns, benefiting from the flexibility and adaptability of deep learning architectures.

The selection of the optimal activation function, denoted as ϕ and ϕ_o, is pivotal to neural network efficacy. In this study, the Adjusted Softplus activation function (equation 4.34) serves as ϕ, introducing nonlinearity in hidden layers. Conversely, the ReLU activation function (equation 4.32) is adopted for ϕ_o to impart efficient nonlinearity to the output. Prior research underscores the pronounced influence of activation functions on network performance Karlik and Olgac [2011], Ramachandran et al. [2017]. Thus, careful consideration should be given when selecting an activation function from the various options available He et al. [2015].

4.4.2 σ-Cell-N: Integrating Stochastic Layer

Building on the σ-Cell RNN volatility model introduced in the previous section, we present the enhanced σ-Cell-N model. This iteration integrates a stochastic layer into the σ-Cell's RNN dynamics, adding depth and complexity to its predictive capabilities.The stochastic layer introduces a stochastic component to the residuals. Specifically, for each time instance t, the residual is given by 4.6.

$$\tilde{x}_{t-1} = x_{t-1} - N(0, \sigma_{t-1}) \tag{4.6}$$

In equation 4.6, N represents the Gaussian distribution, and σ_{t-1} is the volatility from the previous time point. The stochastic layer in the 4.6 model is formulated to directly incorporate the volatility from the previous time point as a variance measure,

thereby coupling the past volatility's influence with the current observation in a manner reminiscent of GARCH model dynamics. This design choice ensures a continuous and coherent propagation of uncertainty through the series, allowing for more nuanced volatility predictions. While the notation draws inspiration from RNN structures, it essentially mirrors the error term in traditional GARCH models as depicted in equation 4.1.

The variance dynamics over time are represented by equations 4.7, 4.8.

$$\tilde{\sigma}_t^2 = \phi(\tilde{\sigma}_{t-1}^2 W_s + \tilde{x}_{t-1}^2 W_r + b_h) \tag{4.7}$$

$$\sigma_t^2 = \phi_o(\tilde{\sigma}_t^2 W_o + b_o) \tag{4.8}$$

In equations 4.7, 4.8, $\sigma_{t-1}^2 W_s$ captures the influence of past variance on current variance, echoing GARCH models legacy effects. The $\tilde{x}_{t-1}^2 W_r$ term assesses the squared residual effects on current variance, paralleling GARCH models' disturbance influence.

The weights W_s, W_r, and W_o are learned parameters potentially derived from neural network structures. The functions ϕ and ϕ_o are activation functions 4.34, 4.32, respectively, adding nonlinearity to the model, while b_h and b_o are bias terms.

By combining past variance, disturbances, and a stochastic layer, the σ-Cell-N offers a more comprehensive perspective on volatility dynamics. This model exemplifies the blend of GARCH principles with neural architecture-derived weights and nonlinear activations, providing a richer understanding of time series volatility.

4.4.3 σ-Cell-RL: Integrating Residuals RNN Layer

The enhanced model introduces a novel mechanism for calculating residuals, diverging from the purely stochastic approach seen in the σ-Cell-N configuration. Instead of solely relying on stochastic deviations, this version uses the discrepancies between empirical data and predictions provided by an added RNN layer G, as shown in equations 4.9, 4.10, where we use the hyperbolic tangent (tanh) activation function φ LeCun et al. [1989].

$$h_t = \varphi\left(x_{t-1}W_{xh} + h_{t-1}W_{hh} + b_h\right) \tag{4.9}$$

$$G(h_t) = \varphi\left(h_t W_{ho} + b_o\right) \tag{4.10}$$

$$\tilde{x}_{t-1} = x_{t-1} - G(h_t) \tag{4.11}$$

Here, $\tilde{x}_{t-1}$ represents the residuals at the t^{th} time instance. The residual computation takes into account the actual value x_{t-1} as input for G. G is computed using the prior hidden state h_{t-1} and the current input x_{t-1}, reflecting the behavior of recurrent cells. In essence, h_{t-1} captures historical context, which, combined with x_{t-1}, aids in current forecasting. For variance, the modeling remains fundamentally consistent, as shown in equation 4.12.

$$\tilde{\sigma}_t^2 = \phi(\tilde{\sigma}_{t-1}^2 W_s + \tilde{x}_{t-1}^2 W_r + b_h) \tag{4.12}$$

$$\sigma_t^2 = \phi_o(\tilde{\sigma}_t^2 W_o + b_o) \tag{4.13}$$

The critical distinction is that the residuals, $\tilde{x}_t$, now stem from the RNN layer's predictions. This tie-in of the RNN layer inherently adjusts the variance equation based on the predictive capabilities of the RNN cell.

The σ-Cell-RL model integrates an RNN component to craft residuals. Instead of using stochastic elements for unpredictability, it harnesses the RNN's forecasting deviations. This marriage of time series modeling with neural networks enhances adaptability, potentially elevating the model's capability to detect complex data patterns in sequences.

4.4.4 σ-Cell-NTV: Integrating Time-Varying Approach

Building upon previous discussions on fixed parameter weights W_r and W_s, this section delves into a time-varying approach for these parameters. First, the input vector $\mathbf{x}_{t-1}$ undergoes a transformation 4.14.

$$w_{t-1} = \tilde{\varphi}\left(\mathbf{W}\mathbf{x}_{t-1} + \mathbf{b}\right) \tag{4.14}$$

Here, the linear combination of $\mathbf{x}_{t-1}$ with weight matrix $\mathbf{W}$ and bias vector $\mathbf{b}$ is passed through a nonlinear function, $\tilde{\varphi}$, resulting in the vector w_{t-1}.

Further, w_{t-1} is split into two components 4.15, 4.16.

$$W_{s,t} = \pi_1\left(w_{t-1}\right) \tag{4.15}$$

$$W_{r,t} = \pi_2\left(w_{t-1}\right) \tag{4.16}$$

Using π_1 and π_2, the first and second halves of w_{t-1} are extracted, respectively. For a w_{t-1} with $2n$ elements 4.17, 4.18.

$$\pi_1\left(w_{t-1}\right) = w_{t-1}[1:n] \tag{4.17}$$

$$\pi_2\left(w_{t-1}\right) = w_{t-1}[n+1:2n] \tag{4.18}$$

The residual computation employs stochasticity similar to 4.6, we provide it in equation 4.19.

$$\tilde{x}_{t-1} = x_{t-1} - N(0, \sigma_{t-1}) \tag{4.19}$$

Lastly, the variance evolution, similar to σ-Cell dynamics, is described in equation 4.20

$$\tilde{\sigma}_t^2 = \phi(\tilde{\sigma}_{t-1}^2 W_{s,t} + \tilde{x}_{t-1}^2 W_{r,t} + b_h) \tag{4.20}$$

$$\sigma_t^2 = \phi_o(\tilde{\sigma}_t^2 W_o + b_o) \tag{4.21}$$

In 4.20, the variance σ_t^2 hinges on past variance, σ_{t-1}^2, modulated by the time-varying parameter $W_{s,t}$ and the squared residuals $\tilde{x}t^2$ modulated by Wr, t.

Conclusively, by integrating time-varying parameters $W_{s,t}$ and $W_{r,t}$, we achieve a

fusion of traditional time series techniques with deep learning approaches, by refining variance modeling by time-varying parameters.

4.4.5 σ-Cell-RLTV: Integrating Time-Varying Approach

The proposed model marries the dynamic attributes of the time-varying method with the recurrent, residual features of the σ-Cell-RL. For each time step t, we describe it 4.22, 4.23, 4.24, 4.25, 4.26, 4.27, 4.28.

$$w_t = \tilde{\varphi}(\mathbf{W}\mathbf{x}_{t-1} + \mathbf{b}) \tag{4.22}$$

$$W_{s,t} = \pi_1(w_t) \tag{4.23}$$

$$W_{r,t} = \pi_2(w_t) \tag{4.24}$$

$$h_t = \varphi\left(x_{t-1}W_{xh} + h_{t-1}W_{hh} + b_h\right) \tag{4.25}$$

$$G(h_t) = \varphi\left(h_t W_{ho} + b_o\right) \tag{4.26}$$

$$\tilde{x}_{t-1} = x_{t-1} - G(h_t) \tag{4.27}$$

$$\tilde{\sigma}_t^2 = \phi(\tilde{\sigma}_{t-1}^2 W_{s,t} + \tilde{x}_{t-1}^2 W_{r,t} + b_h) \tag{4.28}$$

$$\sigma_t^2 = \phi_o(\tilde{\sigma}_t^2 W_o + b_o) \tag{4.29}$$

Here, σ_{t-1}^2 represents the historical variance, modulated by the recurrent, time-varying weights $W_{s,t}$ and $W_{r,t}$. Parameters - $\mathbf{W}$ and $\mathbf{b}$ denote weights and biases. And $\tilde{\varphi}, \pi_1, \pi_2, f$, and ϕ are the operational functions in the network. However, h_{t-1}, $\tilde{\sigma}_{t-1}$ are the hidden states from the prior time step. Analogous with σ-RL, we

implement $\tilde{x}_{t-1}$ in equations 4.25, 4.26, 4.27.

In this approach, we improved memory retention and heightened resistance to specific noise disturbances. However, the model comes with an increased computational burden and a potential to overfit, particularly in sparser data sets.

4.4.6 Loss-function

The objective of this study is to learn the optimal functions F and G that best describe our data. In order to do this, we employ the principle of maximum likelihood estimation. The maximum likelihood method aims to find the parameters of a statistical model that maximizes the likelihood of the data given the model.

Our task then becomes the minimization of the negative log-likelihood of the observed data given the model. This is expressed in the following loss function in equation 4.30.

$$L = \sum_{t} \left[\log\left(\sigma_t^2\right) + \frac{\left(x_t - G\left(x_t\right)\right)^2}{\sigma_t^2} \right] \tag{4.30}$$

The first term inside the sum is the logarithm of the variance, σ_t^2. The second term is the squared difference between the observed value, x_t, and the predicted value, $G(x_t)$, scaled by the inverse of the variance. When the predicted value is close to the observed value, this term becomes small, whereas when the predicted value deviates from the observed value, this term becomes large, thus increasing the loss L.

To find the functions F and G that yield the minimum value of this loss, we formulate the following optimization problem 4.31.

$$\underset{F,G}{\operatorname{argmin}}\ L(F, G) \tag{4.31}$$

In this context, we are seeking the functions F and G that minimize the negative log-likelihood. This task is typically approached via gradient-based optimization methods, such as Gradient Descent or Stochastic Gradient Descent. In these methods, we iteratively update the parameters of F and G in the direction that most reduces L, until we reach a point where the loss can no longer be decreased (a minimum).

The use of negative log-likelihood as opposed to the standard log-likelihood stems

from the nature of the optimization problem. The standard log-likelihood aims to find the parameters that maximize the likelihood, but optimization algorithms are traditionally designed to minimize a given function. Therefore, by taking the negative of the log-likelihood, the maximization problem is transformed into a minimization problem, making it more tractable for standard optimization algorithms.

4.4.7 Activation Function

Activation functions are a fundamental component of neural networks. They determine the output of a neuron given an input or set of inputs, hence controlling the complexity of the model and its ability to fit nonlinear patterns within the data. When predicting financial market volatility, nonlinearity and complexity are important considerations. Financial markets are influenced by a multitude of interconnected variables that often interact in nonlinear ways. As such, the ability to model these intricate relationships is critical for accurate predictions.

Furthermore, financial market volatility can be influenced by a variety of unexpected events such as geopolitical changes or economic shocks, leading to abrupt jumps or falls in prices. This reality requires an activation function that can effectively model sudden changes or discontinuities, something that linear activation functions can't handle.

In this context, nonlinear activation functions like Rectified Linear Unit (ReLU) and Softplus become particularly useful Nair and Hinton [2010], Glorot et al. [2011]. They introduce the necessary nonlinearity to the model while avoiding the issues of vanishing or exploding gradients, thereby improving the training of deep neural networks. The ReLU function is widely used due to its simplicity and efficiency, while Softplus provides a smooth and differentiable approximation of ReLU, thereby maintaining a balance between computational efficiency and the ability to handle complex, nonlinear relationships.

Understanding the nature of the task and the specific properties of each activation function allows us to make informed decisions when designing and training our neural networks for volatility prediction.

In the following subsections, we will describe the activation functions we have chosen for this task.

ReLU (Rectified Linear Unit) Activation Function

The Rectified Linear Unit, or ReLU, is a simple, nonlinear activation function that has become a default choice in many neural network architectures Nair and Hinton [2010], Glorot et al. [2011], Ramachandran et al. [2017]. The ReLU function is defined in equation 4.32. This means that it outputs the input directly if it is positive; otherwise, it outputs zero. The simplicity of ReLU reduces the computational cost and mitigates the vanishing gradient problem, which aids in training deep neural networks.

$$f(x) = max(0, x) \tag{4.32}$$

Adjusted Softplus Activation Function

The Softplus function is another nonlinear activation function, which can be considered as a smooth approximation to the ReLU function Glorot et al. [2011]. Softplus is defined in equation 4.33. Unlike ReLU, Softplus does not have a sharp transition at 0 and never produces an absolute zero activation, except at negative infinity,

$$f(x) = log(1 + e^x) \tag{4.33}$$

We develop a particular softplus activation function equation 4.34, the function is modified to have zero output for negative input values and is scaled such that the Softplus of 1 is 1. This modified Softplus function can be particularly useful in scenarios where the benefits of a smooth activation function are desired, and where output values are preferably non-negative and normalized with respect to a specified scale.

$$\text{Adjusted Softplus}(x) = \max\left(0, \left(\frac{1}{\beta}\log\left(1 + e^{\beta x}\right) - \frac{\log(2)}{\beta}\right) \cdot \frac{1}{\frac{1}{\beta}\log\left(1 + e^{\beta}\right) - \frac{\log(2)}{\beta}}\right) \tag{4.34}$$

4.4.8 Adam Optimizer

For training our proposed model, we employ the Adam (Adaptive Moment Estimation) optimization algorithm. Adam is particularly well-suited for problems like ours that

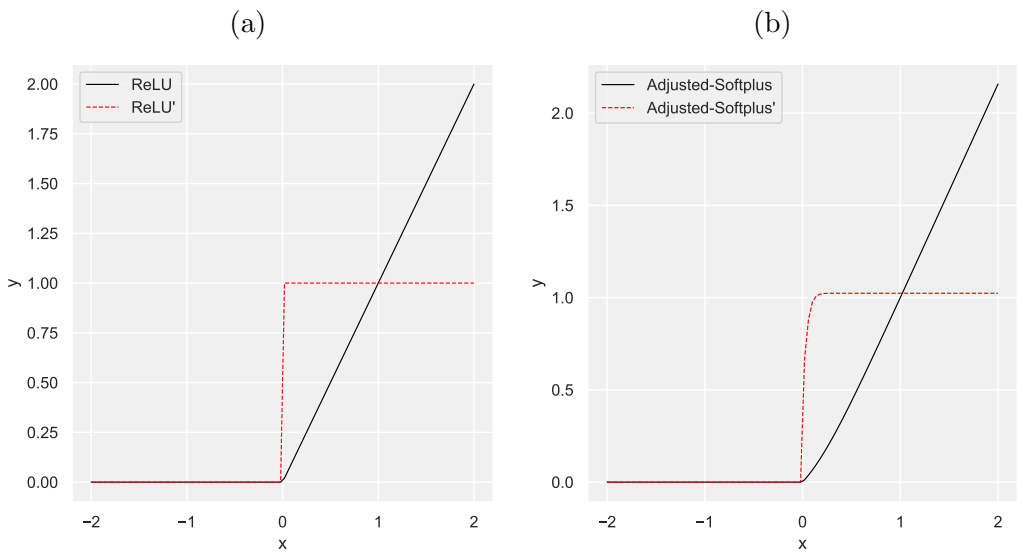

Figure 4.1: The following plot illustrates Activation Functions ReLU and Adjusted-Softplus. a) The plot illustrates the behaviors of the ReLU and derivative b) Adjusted-Softplus activation functions and derivatives. ReLU, which is zero for negative inputs and linear with slope one for positive inputs, provides a simple, computationally efficient nonlinearity. On the other hand, Adjusted-Softplus is a smoothed version of ReLU for $x > 0$.

involve large parameter spaces and potentially noisy or sparse gradients. Its adaptive learning rates for different parameters make it highly efficient, allowing for quicker convergence while requiring lower memory resources compared to other optimization algorithms Kingma and Ba [2014]. The algorithm's ability to compute exponential moving averages of the gradient and the squared gradient, which are then bias-corrected, offers a robust approach to navigating the complex loss landscape commonly encountered in financial time series modeling.

The choice of Adam as our optimization algorithm aligns well with the intricacies of volatility forecasting, where the objective function can be highly nonlinear, and the gradients can vary widely across different regions of the parameter space. By leveraging Adam's adaptive learning rates and moment estimates, as defined in equations 2.45 to 2.49, we aim to achieve a more stable and efficient training process, thereby enhancing the generalizability and predictive power of our model.

4.4.9 Training

The weights of the σ-Cell RNNs are initialized using the Xavier Uniform distribution, which provides a balanced initial scale for the weights, facilitating the learning process, and the biases are initialized to zero Glorot and Bengio [2010], Sutskever et al. [2013]. Gradient clipping is employed as a regularization technique to avoid the adverse effects of exploding gradients, which can destabilize the learning process Pascanu et al. [2013]. The gradients are clipped to a maximum norm of 1.0, ensuring that the updates to the weights during training do not become excessively large and destabilize the network.

The iterative learning process is guided by the Adam optimizer, which is well-suited for training deep neural networks with large-scale data. The Adam optimizer adapts the learning rate for each weight based on the historical gradients, providing an efficient and stable optimization approach. During each epoch of training, the weights of the σ-Cell RNNs are updated based on the gradients of the employed Log-likelihood loss equation 4.30 with respect to the weights. This iterative update of the weights continues through multiple epochs until the convergence criteria are met. As the training progresses, the model gradually learns to capture the underlying patterns in the data, refining the weights and biases to minimize the loss function. This iterative learning approach allows the model to effectively learn the complex dynamics in the data, and the resulting trained σ-Cell RNN provides an accurate and robust model for capturing the volatility dynamics in financial time series data.

4.5 Experimental Approach: Synthetic and Real Data

In our study, we leveraged a comprehensive suite of metrics and statistical tests to rigorously evaluate the forecasting performance of various models. We employed the Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) to gauge the average magnitude of errors between predicted and observed values, defined in equations 2.50 and 2.51, respectively. To account for heteroskedasticity in the data, we also utilized the Heteroskedasticity Adjusted Root Mean Square Error (HRMSE),

formulated in equation 2.52 Bollerslev and Ghysels [1996]. Recognizing the limitations of HRMSE, we further incorporated the QLIKE loss function, which is particularly robust for volatility forecasting, as shown in equation 2.53 Patton [2011].

To quantify the goodness-of-fit of our models to the observed data, we employed the Negative Log-Likelihood (NLL) metric, defined in equation 2.54. Additionally, we used the R^2 of Mincer-Zarnowitz forecasting regressions to assess the explanatory power of our models Mincer and Zarnowitz [1969].

For comparative evaluation, we applied the Diebold-Mariano (DM) test, focusing on Mean Squared Error (MSE) as the loss function, to discern statistically significant differences between the forecasting models Diebold and Mariano [1995]. To further substantiate our findings, we employed the Model Confidence Set (MCS) test, which identifies a set of best-performing models with a high level of statistical confidence Hansen et al. [2011]. This test was conducted using bootstrapping techniques, allowing us to compare multiple forecasting models and identify the most reliable ones with a 5

4.5.1 Synthetic Data Generation

Our synthetic data generation is inspired by the cyclical and often fluctuating nature of financial market volatility. We consider a sequence of 2000 data points where volatility (σ_i) at each point i is generated using equation 4.35.

$$\sigma_i = 1 + A \sin\left(\frac{\pi i}{B}\right) \tag{4.35}$$

Here, the parameters A and B govern the amplitude and frequency of the sine wave, respectively. These are set to $A = 0.7$ and $B = 50$ in our specific synthetic data generation process. The sin function imbues our model with cyclical fluctuations in the volatility, embodying the frequently changing volatility regimes often observed in financial markets.

The synthetic return (x_i) at each point i is then created using the equation:

$$x_i = \sigma_i \cdot \epsilon_i \tag{4.36}$$

In 4.36 ϵ_i is a random number drawn from a standard normal distribution ($\epsilon_i \sim$

$N(0,1)$). This ensures that our synthetic returns (x_i) are directly influenced by our generated volatility (σ_i).

We split this synthetic data set into a training set (the first half of the data) and a test set (the second half) to evaluate the model's performance on unseen data.

This synthetic data provides a robust testing ground for our model, enabling us to compare the predicted volatility values with the true known volatility. This synthetic data set, with known underlying dynamics, is a valuable benchmark for evaluating the performance of the volatility modeling. It is essential to note that the synthetic data has been structured to reflect some stylized facts of financial returns, which will aid in better understanding the model's efficacy in real-world scenarios.

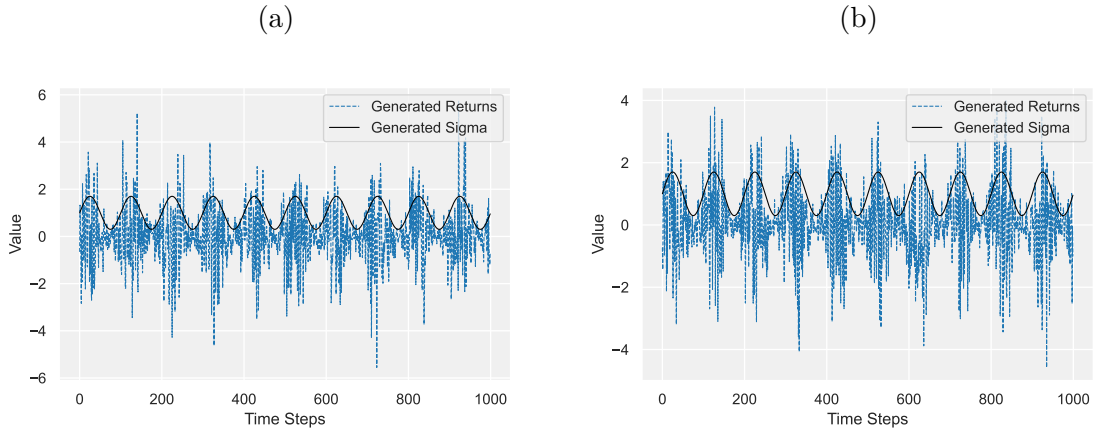

Figure 4.2: The following plot illustrates the generated synthetic data, which includes a sequence of returns and their associated volatility denoted by σ. Panel (a) represents the training data, while panel (b) displays the out-of-sample data. The blue dashed line represents the true generated returns, and the black solid line depicts the true sigma values. This synthetic data set is characterized by known underlying dynamics.

In the following sections, we will expand our experimental analysis to real financial data, bringing additional complexity and testing the model's performance in capturing more intricate, real-world dynamics.

4.5.2 Real Data

In this study, we investigate the proposed models for estimating and predicting realized volatility in diverse market structures. We focus on two specific asset types: an index, represented by the S&P 500, and a cryptocurrency, represented by the Bitcoin-USD

(BTCUSDT) pair. We analyze daily data to examine the effects and dependencies associated with this granularity 5.1.

For the S&P 500 data, we use 1-minute price observations from March 10, 2007, to March 1, 2022, resulting in 3,800 days of realized volatility observations. RV and returns are calculated based on the last daily closing price. Our experimental data set consists of intraday returns and corresponding RV. We partition the data set into training, validation, and test subsets. The validation and test subsets each contain 252 points, equivalent to one trading year, with the remaining data allocated to the training subset.

For the cryptocurrency data, we study the Bitcoin-USD pair. We use 1-minute price data from January 1, 2013, to April 20, 2020. This data set provides extensive price points, from which we calculate 2,667 RV observations. As with the S&P 500 data set, we divide the cryptocurrency data into training, validation, and test subsets, with the validation and test subsets each containing 252 points.

Table 4.1: Data Set Description for S&P 500 and BTCUSDT

Asset	Time Frame	From	To	RV Points
S&P 500	1 minute	10.03.07	01.03.22	3,800
BTCUSDT	1 minute	01.01.13	20.04.20	2,667

Note: The table presents an overview of the data used in the analysis, including the asset, the time frame of the data, the date range of the data, and the number of realized volatility (RV) points. The last 252 data points are used for out-of-sample testing, and the preceding 252 data points are used for validation.

The mean, median, standard deviation, skewness, and kurtosis were calculated for each asset's returns, giving us valuable insights into the underlying data distributions.

The S&P 500 stock has a mean return of 0.00029, which is close to zero, indicating no clear trend in either direction, Table 5.2. The standard deviation is 0.0128, implying a relatively moderate degree of price fluctuation. The negative skewness of -0.1886 suggests a left-leaning data distribution, while the kurtosis of 13.9915 indicates a distribution with relatively heavy tails, implying occasional large changes in stock prices. The median return of 0.0007 further supports the near-zero mean, confirming the absence of a clear trend.

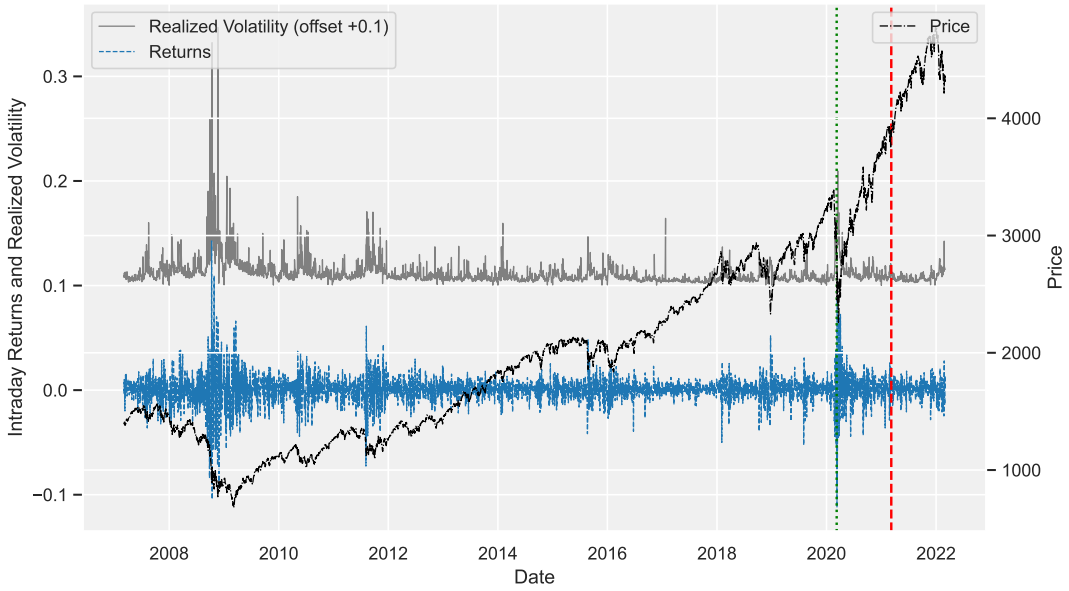

Figure 4.3: This plot depicts the Realized Volatility, Returns, and Price of the S&P 500 Index between 10th March 2007 and 1st March 2022. The gray solid line represents the realized volatility (offset by +0.1), the blue dashed line represents intraday returns, and the black dash-dot line indicates the price. Both the RV and returns, derived from daily data, are presented on the primary y-axis, while the price is shown on a secondary y-axis. The vertical red dashed and green dotted lines demarcate the beginnings of the test and validation sets, respectively, with each set comprising 252 points. All remaining data serve as the training set.

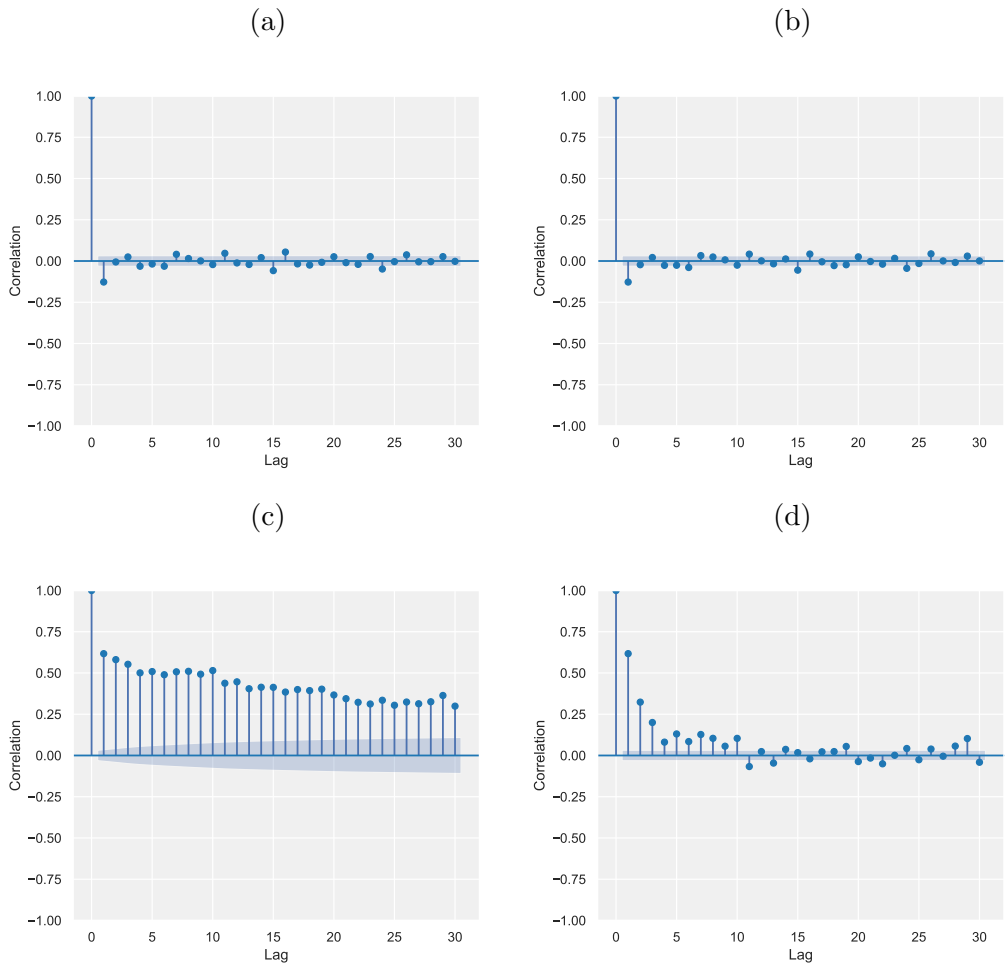

Figure 4.4: This plot displays the autocorrelation (ACF) and partial autocorrelation (PACF) for the returns and volatility of the S&P 500 Index. The ACF illustrates the extent of a linear relationship between current values and their lags, while the PACF captures the correlation between a value and its lag that isn't explained by shorter lags. a) ACF for returns b) PACF for returns c) ACF for volatility d) PACF for volatility

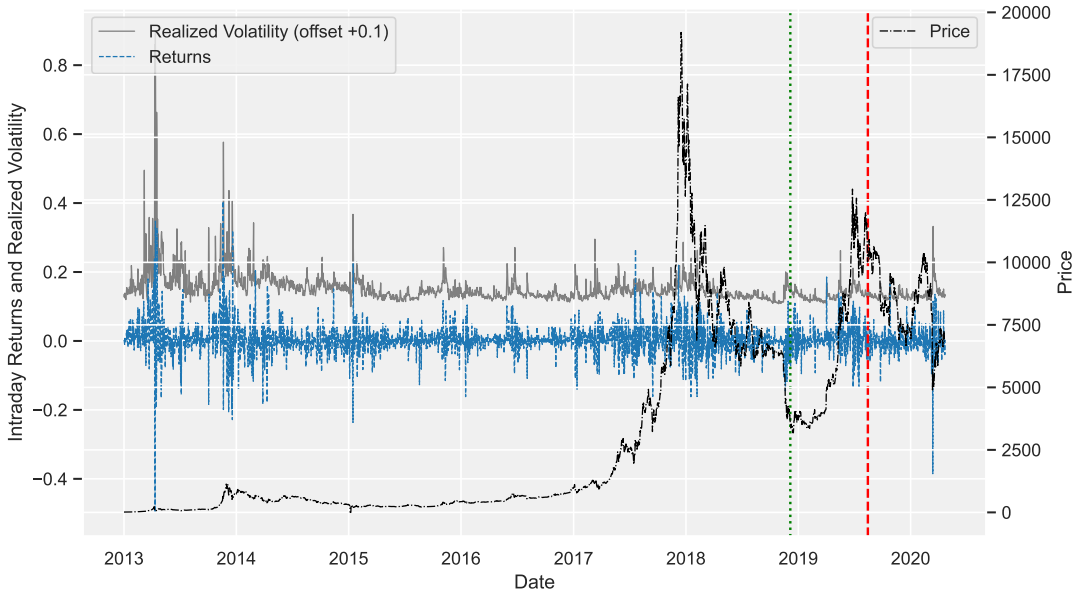

Figure 4.5: This plot depicts the Realized Volatility, Returns, and Price of the Bitcoin-USD Pair between 1st January 2007 and 20th April 2020. The gray solid line represents the realized volatility (offset by +0.1), the blue dashed line represents intraday returns, and the black dash-dot line indicates the price. Both the RV and returns, derived from daily data, are presented on the primary y-axis, while the price is shown on a secondary y-axis. The vertical red dashed and green dotted lines demarcate the beginnings of the test and validation sets, respectively, with each set comprising 252 points. All remaining data serve as the training set.

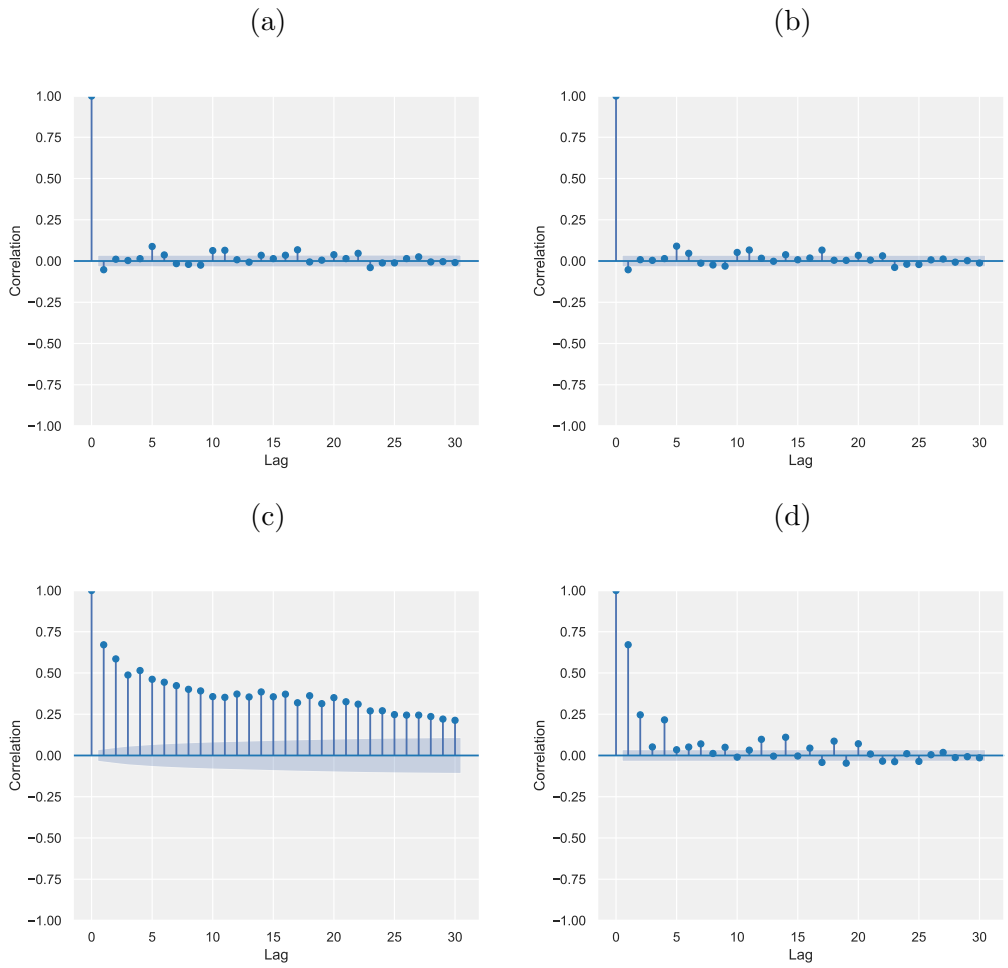

Figure 4.6: This plot displays the autocorrelation (ACF) and partial autocorrelation (PACF) for the returns and volatility of the Bitcoin-USD Pair. The ACF illustrates the extent of a linear relationship between current values and their lags, while the PACF captures the correlation between a value and its lag that isn't explained by shorter lags. a) ACF for returns b) PACF for returns c) ACF for volatility d) PACF for volatility

In contrast, the Bitcoin asset has a higher mean return of 0.00349, reflecting a slight upward trend. It also has a higher standard deviation of 0.0466, indicating more substantial price volatility compared to the S&P 500 stock, table 5.2. The slightly negative skewness of -0.0726 and the higher kurtosis of 14.3508 suggest a distribution with an extreme fat tail, indicating a higher likelihood of significant price changes. The median return of 0.0020 aligns with the positive mean, further supporting the upward trend.

We also generated plots for each asset to offer a more intuitive understanding of the data distributions. Figure X showcases the distribution of intraday returns, and Figure Y presents a box plot of the intraday returns. These graphical representations further illustrate the unique characteristics of each asset's return distribution, serving as a visual supplement to our statistical analysis.

Table 4.2: Statistical Summary of Returns for S&P 500 and BTCUSDT.

Asset	Mean	Median	STD	Skewness	Kurtosis
S&P 500	0.00029	0.0007	0.0128	-0.1886	13.9915
BTCUSDT	0.00349	0.0020	0.0466	-0.0726	14.3508

Note: The table presents key statistical metrics of the interdaily returns for the S&P 500 index and the Bitcoin-USD (BTCUSDT) trading pair. The metrics include mean, median, standard deviation (STD), skewness, and kurtosis. A positive skewness indicates a right-side heavier tail of the probability density function, while a negative skewness indicates a left-side heavier tail. Kurtosis measures the "tailedness" of the probability distribution of returns. Higher kurtosis indicates a heavier tail, signifying a higher probability of extreme outcomes. For the analysis, we use the entire data set without dividing it into validation and test sets.

By focusing on these two contrasting asset types, we aim to gain a comprehensive understanding of how various σ-Cell can estimate and predict realized volatility across different market structures accurately.

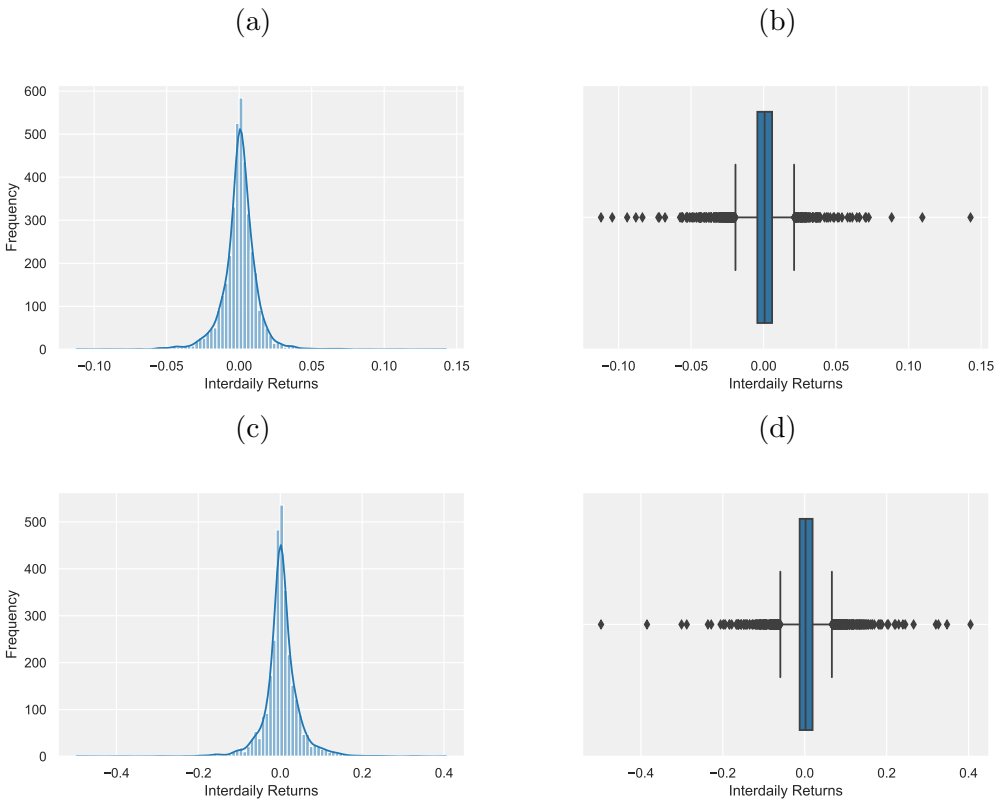

Figure 4.7: This plot showcases the distribution and Box Plot of Returns for the S&P 500 and Bitcoin-USD. Panel a) displays the histogram of the S&P 500 returns, while Panel b) provides its box plot, highlighting the spread of data and potential outliers. Similarly, Panel c) illustrates the histogram of Bitcoin-USD returns, and Panel d) presents its box plot, showcasing data dispersion and any outliers. These visualizations offer insights into the central tendency, dispersion, and shape of the return distributions for both assets.

4.6 Results

In this study, we sought to gauge the predictive efficacy of the proposed σ-Cell models. Our evaluation encompassed various experiments using synthetic and real-world financial data sets. The proposed models' performance was contrasted against traditional GARCH family models and the Stochastic Volatility (SV) model in the synthetic data scenario. However, the comparison was made against GARCH, SV models, as well as the Heterogeneous Autoregressive (HAR) model proposed by Corsi (2009) in the actual data context Corsi [2009].

4.6.1 Synthetic data set

The comparative analysis delineated in Table 4.3 elucidates the performance of various σ-Cell model variants vis-à-vis established models across multiple evaluation metrics, namely RMSE, MAE, NLL, δ Mean, and δ Amplitude. Notably, the σ-Cell-RLTV variant demonstrates superior performance in RMSE and MAE metrics, outclassing all other models under consideration. Conversely, the TARCH model lags behind, registering the least favorable scores in these categories.

In terms of the NLL metric, the σ-Cell-NTV variant emerges as the most efficient model, while the TARCH model once again exhibits suboptimal performance. Furthermore, the σ-Cell-NTV variant also leads in the δ Mean and δ Amplitude metrics, underscoring its robustness across multiple dimensions of evaluation.

Turning our attention to Table 4.4, which assesses out-of-sample performance on synthetic test data, the σ-Cell-RLTV variant continues to distinguish itself. It excels not only in RMSE and MAE but also registers the lowest NLL among its σ-Cell counterparts.

When juxtaposed with other volatility models such as GARCH(1,1), EGARCH, TARCH, GJR-GARCH, and SV, the σ-Cell-RLTV variant maintains its competitive edge in RMSE and MAE metrics.

Overall, the σ-Cell-RLTV model shows superior performance in RMSE, MAE, and NLL metrics compared to other models. Its performance in the δ Mean and δ Amplitude metrics also suggest its consistency and accuracy in predicting volatility

Table 4.3: Comparative Performance Metrics of Volatility Models on In-Sample Synthetic Data

Model	RMSE	MAE	NLL	δ Mean	δ Amplitude
σ-Cell	0.3207	0.2362	0.9703	-0.0810	-1.4876
σ-Cell-N	0.3039	0.2292	0.9651	-0.0316	-1.2188
σ-Cell-NTV	0.2741	0.2223	0.9565	0.0318	-0.4003
σ-Cell-RL	0.3217	0.2401	0.9707	-0.0496	-1.6247
σ-Cell-RLTV	0.2614	0.2043	0.9531	-0.0265	-0.7845
GARCH(1,1)	0.3058	0.2295	1.1977	-0.0537	-1.4702
EGARCH	0.3712	0.2376	1.2255	-0.0594	-4.8949
TARCH	0.3844	0.2471	1.2182	-0.0361	-5.1248
GJR-GARCH	0.3160	0.2383	1.1890	-0.0304	-1.6385
SV	0.3246	0.2762	0.9716	-0.1170	-0.7082

Note: The table presents the performance metrics for five variants of the σ-Cell model and five other volatility models using synthetic in-sample data for validation. The evaluation metrics include RMSE, MAE, NLL, δ Mean, and δ Amplitude. The σ-Cell-RLTV variant is notable for its performance in the RMSE and MAE metrics, while the σ-Cell-NTV variant stands out in the NLL, δ Mean, and δ Amplitude metrics. Values highlighted in bold indicate the best performance for each metric.

changes. These results highlight the potential effectiveness of the σ-Cell-RLTV model in predicting realized volatility on synthetic test data.

Table 4.4: Comparative Performance Metrics of Volatility Models on Out-of-Sample Synthetic Data

Model	RMSE	MAE	NLL	δ Mean	δ Amplitude
σ-Cell	0.3269	0.2584	0.9724	-0.0426	-1.3954
σ-Cell-N	0.3271	0.2510	0.9724	0.0075	-1.0481
σ-Cell-NTV	0.3176	0.2479	0.9694	0.0027	-1.4280
σ-Cell-RL	0.3382	0.2633	0.9761	-0.0231	-1.5759
σ-Cell-RLTV	0.2873	0.2307	0.9602	-0.0110	-0.9959
GARCH(1,1)	0.3214	0.2643	1.1571	-0.0343	-1.0934
EGARCH	0.3230	0.2617	1.1609	-0.0341	-1.2120
TARCH	0.3602	0.2688	1.1834	-0.0460	-3.5309
GJR-GARCH	0.3244	0.2650	1.1644	-0.0354	-1.3509
SV	0.4070	0.3568	1.0565	-0.0087	0.4855

Note: The table presents the performance metrics for various volatility models on out-of-sample synthetic data. The metrics include Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Negative Log-Likelihood (NLL), the difference in mean (δ Mean), and the difference in amplitude (δ Amplitude). The models being compared include five variants of the σ-Cell model as well as five other well-known volatility models. The σ-Cell-RLTV variant stands out with a comparatively low RMSE, MAE, and NLL, indicating superior accuracy and consistency among the σ-Cell variants. Values highlighted in bold indicate the best performance for each metric.

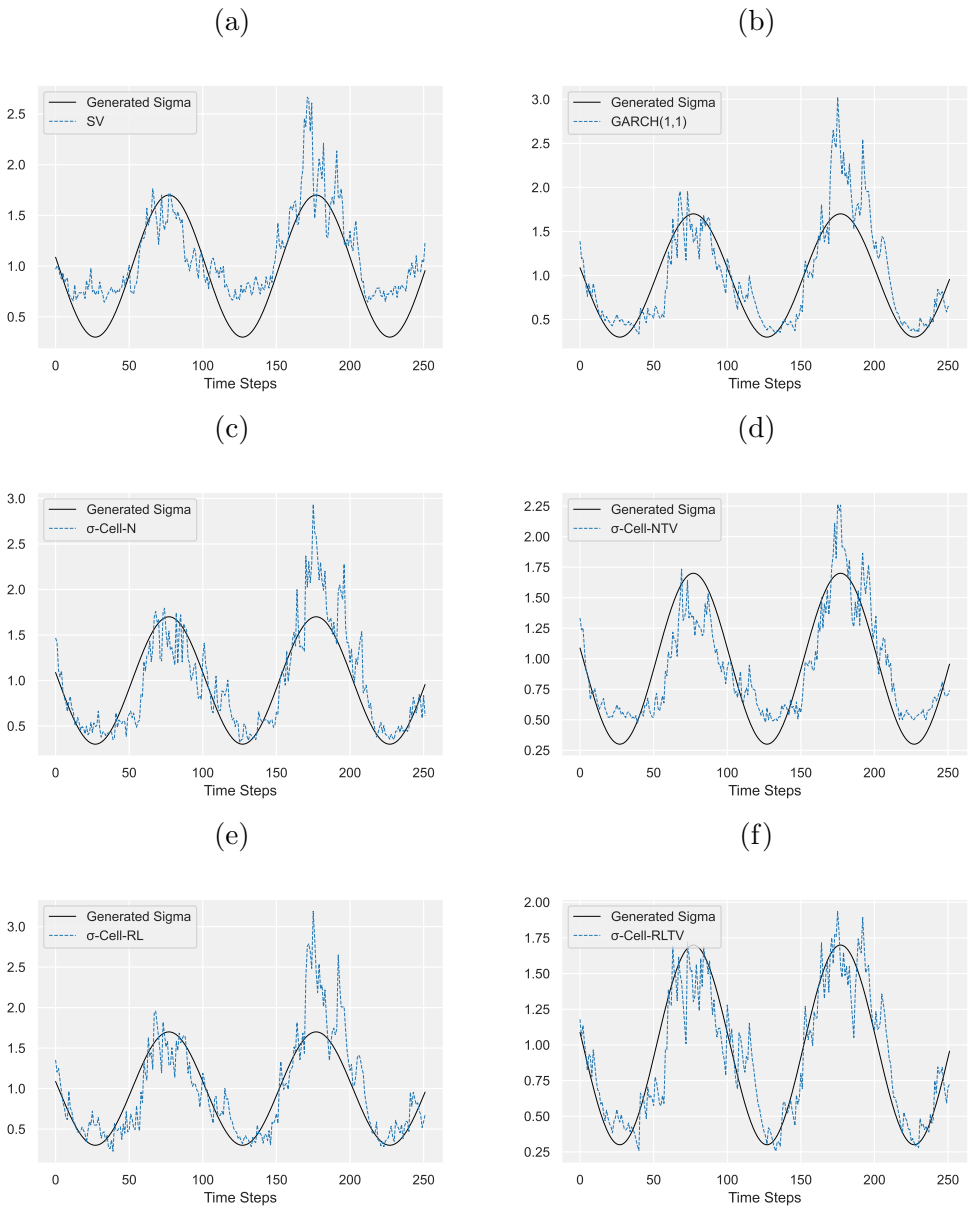

Figure 4.8: The following plot illustrates the predictions for in-sample synthetic data for different forecasting models: (a) Stochastic Volatility (SV) model's prediction of generated sigma values. (b) GARCH(1,1) model's prediction of generated sigma values. (c) σ-Cell-N model's prediction of generated sigma values. (d) σ-Cell-NTV model's prediction of generated sigma values. (e) σ-Cell-RL model's prediction of generated sigma values. (f) σ-Cell-RLTV model's prediction of generated sigma values.

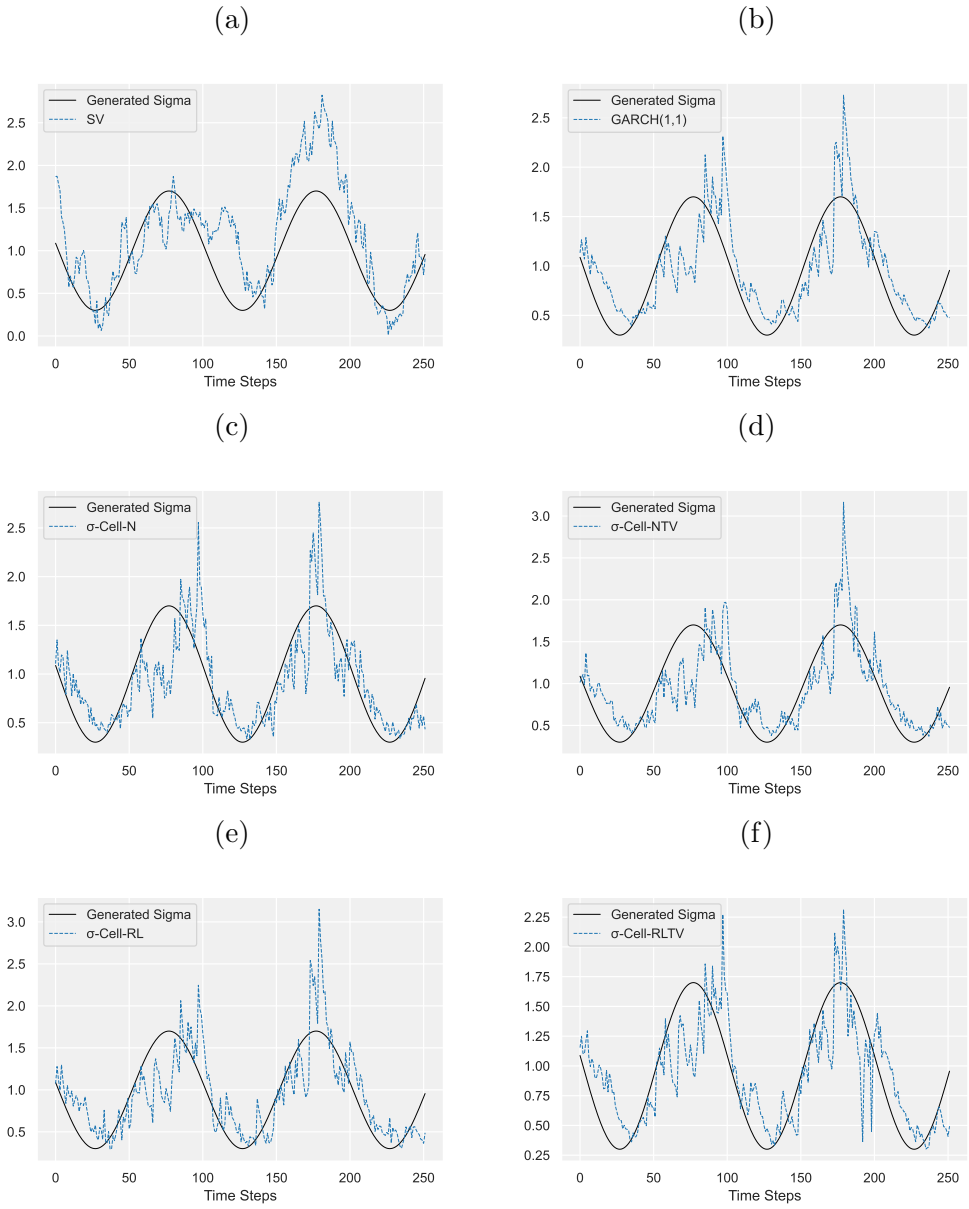

Figure 4.9: The following plot illustrates the prediction for out-of-sample synthetic data for different forecasting models: (a) Stochastic Volatility (SV) model's prediction of generated sigma values. (b) GARCH(1,1) model's prediction of generated sigma values. (c) σ-Cell-N model's prediction of generated sigma values. (d) σ-Cell-NTV model's prediction of generated sigma values. (e) σ-Cell-RL model's prediction of generated sigma values. (f) σ-Cell-RLTV model's prediction of generated sigma values.

Exploring Time-Varying Parameters of σ-Cell-NTV and σ-Cell-RLTV

The σ-Cell-NTV and σ-Cell-RLTV models represent a fusion of time series modeling techniques with the power of deep learning. At their core, these models leverage recurrent neural networks to model the time-dependent structure in financial data, coupled with the introduction of time-varying parameters to capture dynamic shifts in the underlying financial processes.

The σ-Cell-NTV model introduces time-varying weights $W_{s,t}$, and $W_{r,t}$ to model the time-varying nature of financial time series. This is achieved through the transformation of the input vector $\mathbf{x}_t$ via a nonlinear function, followed by the separation of the transformed vector into two components. These components are used to modulate the past variance and the squared residuals in the variance evolution equation. This enables the model to adapt to changing market conditions and capture complex temporal relationships in the data.

On the other hand, the σ-Cell-RLTV model builds upon the σ-Cell-RL by incorporating recurrent, time-varying weights into the variance evolution equation. This provides the model with an improved memory retention capability and heightened resistance to specific noise disturbances. By integrating the recurrent nature of the σ-Cell-RL and the dynamic attributes of the time-varying method, the σ-Cell-RLTV model offers a more robust and nuanced understanding of financial time series data.

Both models demonstrate a clear progression in the evolution of the norms $|W_r|$ and $|W_s|$ during training 4.10, 4.11 Initially, these norms exhibit an unstructured pattern, but as training progresses, a clear structure emerges. The observed structure in the norms reflects the model's ability to capture intricate relationships in the data and provides valuable insights into the underlying financial processes.

While these models offer a promising approach for modeling financial time series data, they come with increased computational complexity and a potential risk of overfitting, especially in sparser data sets. Thus, careful consideration should be given to model selection and hyperparameter tuning to strike a balance between model complexity and predictive performance.

In summary, the σ-Cell-NTV and σ-Cell-RLTV models offer a novel approach to modeling financial time series data by combining traditional time series techniques

with the flexibility and adaptability of deep learning. These models show promise in capturing complex temporal relationships and provide valuable insights into the dynamics of financial processes.

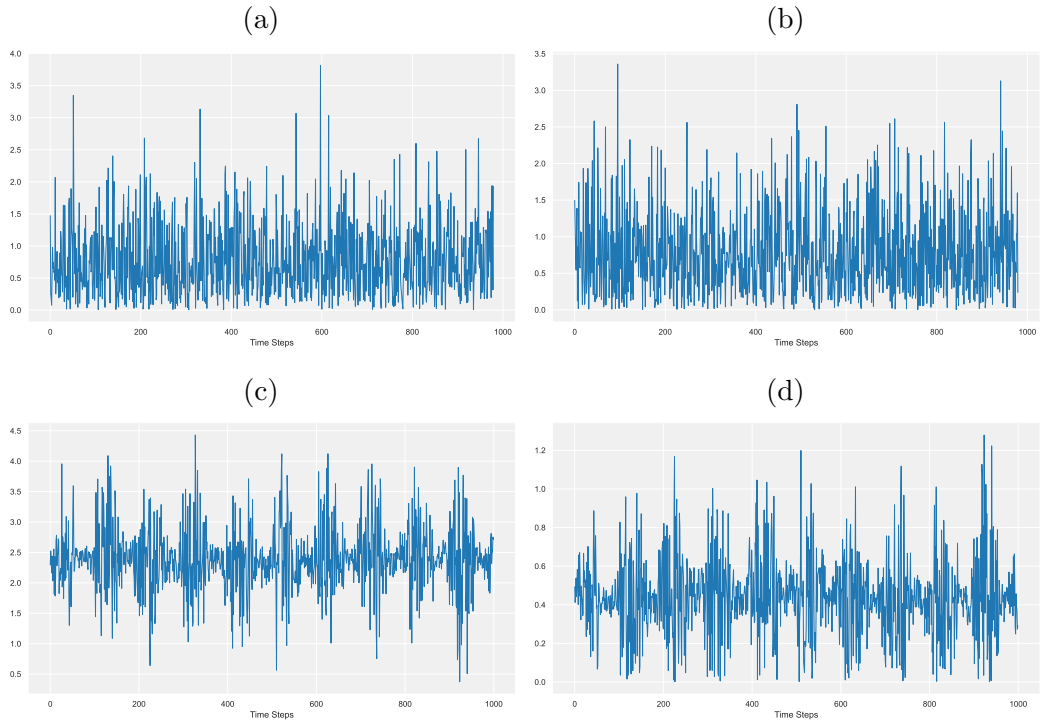

Figure 4.10: The following plot illustrates the evolution of $|W_r|$ and $|W_s|$ in the σ-Cell-NTV model during training. The plots illustrate the progression of norms at two different training epochs, highlighting the emergence of a structured pattern in $|W_r|$ and $|W_s|$ as training progresses. (a) $|W_r|$ at epoch 1: Distribution of the $|W_r|$ during the initial stages of training is mostly noise. (b) $|W_s|$ at epoch 1: Distribution of the $|W_s|$ at the start of training is mostly noise. (c) $|W_r|$ at epoch 100: After 100 epochs, a distinct pattern is visible in the distribution of $|W_r|$. (d) $|W_s|$ at epoch 100: The distribution of $|W_s|$ after 100 epochs, revealing the emergence of a clear structure.

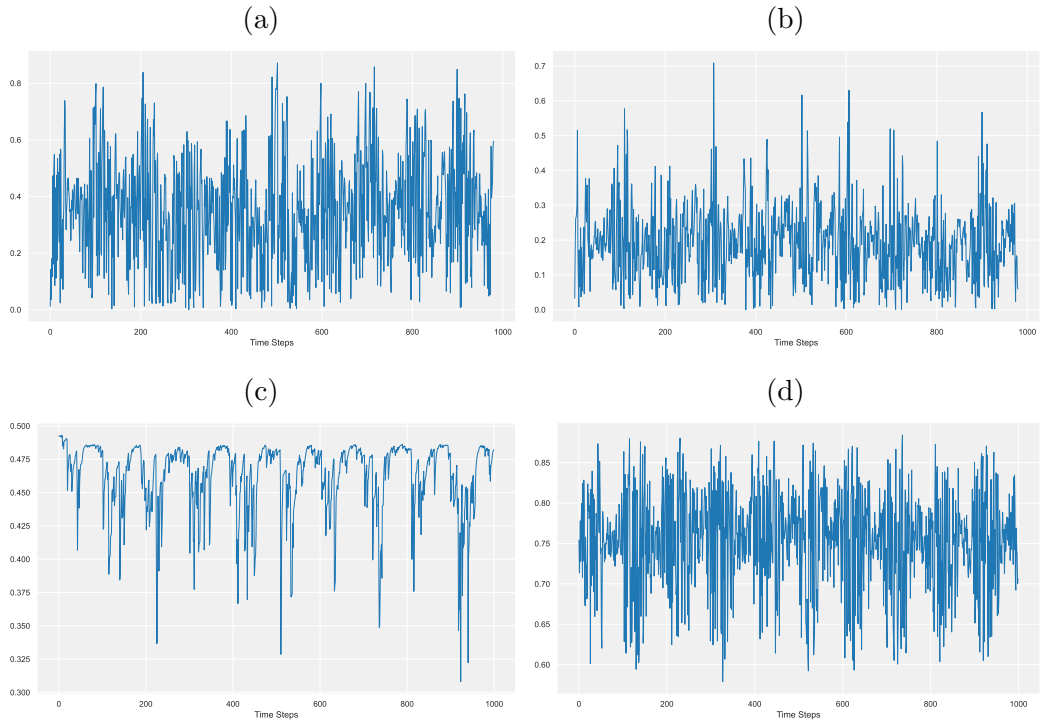

Figure 4.11: The following plot illustrates the evolution of $|W_r|$ and $|W_s|$ in the σ-Cell-RLTV model during training. The plots illustrate the progression of norms at two different training epochs, highlighting the emergence of a structured pattern in $|W_r|$ and $|W_s|$ as training progresses. (a) $|W_r|$ at epoch 1: The $|W_r|$ during the initial stages of training is largely unstructured. (b) $|W_s|$ at epoch 1: The $|W_s|$ at the start of training shows a lack of clear structure. (c) $|W_r|$ at epoch 100: after 100 epochs, a distinct inverse pattern of variance is visible in the $|W_r|$. (d) $|W_s|$ at epoch 100: after 100 epochs $|W_s|$ revealing the emergence of a clear structure.

4.6.2 Real Data

Table 5.3 shows the evaluation of the in-sample performance metrics for a diverse array of volatility forecasting models specifically trained on the S&P 500 index. Within the ambit of σ-Cell models, the σ-Cell-NTV variant merits particular attention for its R^2, which signifies a high degree of predictive accuracy during the in-sample period. This model also manifests superior point forecast accuracy, as evidenced by its comparatively low RMSE. The σ-Cell-RLTV variant is not far behind, also demonstrating robust predictive capabilities as indicated by its R^2.

In juxtaposition with other models, the HAR model is a formidable contender, boasting in-sample solid performance. Its relatively low MAE and RMSE metrics corroborate its point forecast accuracy, while its elevated R^2 underscores its predictive prowess.

Conversely, the SV model languishes at the lower end of the performance spectrum, marred by elevated MAE and RMSE values, which suggest suboptimal point forecast accuracy. Its R^2 further attests to its diminished predictive efficacy relative to the other models under consideration.

Occupying a middle ground, the GARCH variants and EGARCH models exhibit moderate performance metrics. Their R^2 values and RMSE metrics place them in an intermediary position, falling short of the high-performing σ-Cell and HAR models yet surpassing the underperforming SV model.

In summation, the σ-Cell-NTV and σ-Cell-RLTV models distinguish themselves with superior in-sample performance metrics, closely following the traditional HAR model.

Table 5.4 compares the out-of-sample performance of various volatility forecasting models on the S&P 500 index. It particularly focuses on how the σ-Cell models measure up against other models.

Among the σ-Cell models, the σ-Cell-RLTV model performs the best. It has the second-highest R^2 value among all models, indicating strong predictive accuracy. Its lower RMSE compared to other σ-Cell models also suggests a better point forecast accuracy.

The HAR model also performs well, especially considering it uses realized volatility

Table 4.5: In-Sample Performance Metrics for S&P 500 Volatility Forecasting Models

Model	MAE 10^3	RMSE 10^3	HRMSE	QLIKE	R^2
σ-Cell	4.5433	7.1825	2.5177	-3.35507	0.683718
σ-Cell-N	4.2755	7.3268	3.9454	-3.37082	0.703984
σ-Cell-NTV	4.4954	6.7988	2.9629	-3.37653	0.741065
σ-Cell-RL	4.6405	7.4594	3.7466	-3.33337	0.669921
σ-Cell-RLTV	4.2681	6.7396	4.6134	-3.36883	0.72224
GARCH(1,1)	5.3125	8.2736	3.349	-3.36648	0.660096
EGARCH	5.2580	8.3533	3.3518	-3.36965	0.653453
TARCH	5.1771	8.4349	3.232	-3.36978	0.669526
GJR-GARCH	5.1116	8.193	3.2048	-3.36603	0.667675
HAR	4.1901	7.3104	3.7523	-3.3793	0.675116
SV	8.3227	15.8116	13.3486	-3.30144	0.317564

Note: The table presents the in-sample performance of various volatility forecasting models applied to S&P 500 index. The performance metrics include Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Heteroscedasticity-Adjusted RMSE (HRMSE), Quasi-Likelihood (QLIKE), and the coefficient of determination (R^2). Metrics MAE, RMSE are evaluated at a scale of 10^3.

(RV) data as input, which gives it more information. It has the highest R^2 and the lowest RMSE, confirming its strong predictive performance for the S&P 500 index's volatility.

In contrast, the SV model performs poorly. Its MAE and RMSE are much higher than those of the other models, indicating that it may struggle to make accurate predictions for the S&P 500 index's volatility. This could be due to large errors in the model's forecasts.

In summary, the σ-Cell-RLTV model shows the most promise among the σ-Cell models for forecasting S&P 500 volatility.

In summary, among the σ-Cell models, the σ-Cell-RLTV model appears to be the most promising for forecasting the volatility of the S&P 500 index. However, the traditional HAR model also stands out as a strong performer, highlighting the need for further investigation into the comparative advantages of these models in the context of S&P 500 volatility forecasting.

From Table 5.7, we observe the in-sample performance metrics for BTCUSDT volatility forecasting models. The σ-Cell-RLTV model stands out among the

Table 4.6: Out-of-Sample Performance Metrics for S&P 500 Volatility Forecasting Models

Model	MAE 10^3	RMSE 10^3	HRMSE	QLIKE	R^2
σ-Cell	2.9022	4.3506	2.1655	-3.79978	0.292862
σ-Cell-N	2.7759	4.2079	2.4303	-3.85495	0.25181
σ-Cell-NTV	2.8566	4.2148	2.2502	-3.8158	0.327333
σ-Cell-RL	2.8681	4.3061	2.4395	-3.83052	0.257238
σ-Cell-RLTV	2.4940	3.6792	2.1279	-3.86195	0.464835
GARCH(1,1)	2.6258	3.9908	2.6406	-3.868	0.319658
EGARCH	2.8062	4.123	2.671	-3.86277	0.274502
TARCH	2.5811	3.961	2.5103	-3.86102	0.33776
GJR-GARCH	2.4939	3.8503	2.6177	-3.87638	0.372308
HAR	2.3316	3.3896	2.6567	-3.88498	0.516026
SV	68.2419	82.3501	24.6661	-2.64681	0.262184

Note: The table presents the out-of-sample performance of various volatility forecasting models applied to S&P 500 index. The performance metrics include Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Heteroscedasticity-Adjusted RMSE (HRMSE), Quasi-Likelihood (QLIKE), and the coefficient of determination (R^2). Metrics MAE, RMSE are evaluated at a scale of 10^3.

σ-Cell models with a reasonable R^2 value indicating good predictive accuracy. The σ-Cell-RLTV model's performance appears to be competitive, highlighting the potential of this variant for forecasting the volatility of the BTCUSDT trading pair.

Among the traditional models, the HAR model performs as expected well with the highest R^2, making it the best-performing model in this comparison. Moreover, it has the lowest MAE among all models, further indicating the robust forecasting ability of the HAR model in capturing the complex volatility dynamics of the BTCUSDT trading pair.

From Table 5.8, we can see the out-of-sample test results for the BTCUSD trading pair using various volatility forecasting models. Among the σ-Cell models, the σ-Cell-NTV variant stands out with a good R^2 value, showcasing good predictive accuracy in forecasting the BTCUSD trading pair volatility. Further, the model also has a relatively low MAE and RMSE, emphasizing its robust forecasting performance. This demonstrates the effectiveness of the σ-Cell-NTV model in capturing the volatility dynamics of the BTCUSD trading pair.

Regarding traditional models, the GJR-GARCH model performs exceptionally well

Table 4.7: In-Sample Performance Metrics for BTCUSDT Volatility Forecasting Models

Model	MAE 10^3	RMSE 10^3	HRMSE	QLIKE	R^2
σ-Cell	11.4741	17.2758	2.1228	-2.3708	0.406287
σ-Cell-N	9.5845	14.8304	2.0743	-2.40586	0.383233
σ-Cell-NTV	9.8954	14.5676	2.2275	-2.4149	0.40949
σ-Cell-RL	11.8278	17.3736	1.9941	-2.30458	0.323242
σ-Cell-RLTV	9.8975	13.9467	2.2753	-2.41649	0.490842
GARCH(1,1)	10.645	15.2788	2.2737	-2.40314	0.339935
EGARCH	10.4241	15.4131	2.249	-2.40367	0.317588
TARCH	9.815	17.0394	2.2396	-2.40786	0.251025
GJR-GARCH	10.4658	16.0143	2.2568	-2.39506	0.278871
HAR	8.3432	13.046	2.1962	-2.43033	0.519423
SV	16.2048	20.2572	2.632	-2.35452	0.120267

Note: The table presents the in-sample performance of various volatility forecasting models applied to the BTCUSDT trading pair. The performance metrics include Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Heteroscedasticity-Adjusted RMSE (HRMSE), Quasi-Likelihood (QLIKE), and the coefficient of determination (R^2). Metrics MAE, RMSE are evaluated at a scale of 10^3.

in terms of R^2 with the highest value among all models in the table. However, this model has a relatively low MAE and RMSE but is not comparable to σ-Cell-NTV.

Overall, these results emphasize the potential of both σ-Cell and traditional models in predicting the volatility of the BTCUSD trading pair, with the σ-Cell-NTV and GJR-GARCH models showcasing particularly strong performance.

Table 4.9 presents the results of the Diebold-Mariano (DM) test comparing the performance of various volatility forecasting models with the σ-Cell-RLTV model, which serves as the base model for the S&P 500 index. The metrics used for performance evaluation are MSE and MAD losses, and the associated p-values are also reported, signifying the statistical significance of the performance differences.

The σ-Cell, σ-Cell-N, σ-Cell-NTV, and σ-Cell-RL models all show low p-values in both MSE and MAD metrics, indicating their performance is statistically different from the base model, with the σ-Cell model in particular showing extremely low p-values. In contrast, the GARCH(1,1), TARCH, and GJR-GARCH models have relatively high p-values in both metrics, suggesting that their performance is not significantly different from the σ-Cell-RLTV model. The EGARCH model exhibits a low p-value in the MSE

Table 4.8: Out-of-Sample Performance Metrics for BTCUSDT Volatility Forecasting Models

Model	MAE 10^3	RMSE 10^3	HRMSE	QLIKE	R^2
σ-Cell	11.5603	18.5519	2.0116	-2.3075	0.523151
σ-Cell-N	8.6526	15.0106	2.0952	-2.3861	0.54771
σ-Cell-NTV	8.694	15.1973	2.1058	-2.3841	0.552158
σ-Cell-RL	10.2195	16.7332	2.0523	-2.3489	0.481362
σ-Cell-RLTV	8.9647	15.3183	2.1562	-2.3821	0.512133
GARCH(1,1)	11.7684	20.9741	2.2144	-2.3579	0.164693
EGARCH	11.9173	22.4158	2.2097	-2.3569	0.231588
TARCH	11.1243	23.8724	2.212	-2.3619	0.194875
GJR-GARCH	10.316	19.6479	2.1261	-2.3712	0.561019
HAR	8.7206	16.1625	2.1609	-2.3901	0.462834
SV	42.9903	47.1292	3.5722	-2.1328	0.342456

Note: The table presents the out-of-sample performance of various volatility forecasting models applied to the BTCUSDT trading pair. The performance metrics include Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Heteroscedasticity-Adjusted RMSE (HRMSE), Quasi-Likelihood (QLIKE), and the coefficient of determination (R^2). Metrics MAE, RMSE are evaluated at a scale of 10^3.

metric but a relatively high p-value in the MAD metric, indicating that its performance is significantly different in terms of MSE but not in terms of MAD. The HAR model has the lowest MSE and MAD among the models, but its p-values suggest that its performance is not significantly different from the base model.

The SV model has extremely low p-values in both metrics and substantially higher loss values, indicating its significantly inferior performance compared to the base model. In summary, the σ-Cell-RLTV model demonstrates comparable performance to the GARCH(1,1), TARCH, GJR-GARCH, and HAR models in forecasting S&P 500 volatility, while the SV model, σ-Cell, σ-Cell-N, σ-Cell-NTV, and σ-Cell-RL models show significantly different performance.

Table 4.10 presents the results of the DM test comparing the performance of various volatility forecasting models with the HAR model, which serves as the base model for the S&P 500 index.

Most of the models in this table have low p-values in both the MSE and MAD metrics, indicating that their performance is statistically different from the HAR model.

Table 4.9: S&P 500 Volatility Forecasting: Diebold-Mariano Test with σ-Cell-RLTV as the Base Model

Model	MSE Loss	MSE p-value	MAD Loss	MAD p-value
σ-Cell	0.018927	1.840e-05	2.902154	3.608e-13
σ-Cell-N	0.017707	4.069e-03	2.775868	1.204e-02
σ-Cell-NTV	0.017764	1.588e-05	2.856639	1.700e-10
σ-Cell-RL	0.018542	9.517e-03	2.868097	1.937e-02
σ-Cell-RLTV	0.013536	-	2.494049	-
GARCH(1,1)	0.015927	9.404e-02	2.625849	2.729e-01
EGARCH	0.016999	1.675e-02	2.806154	1.275e-02
TARCH	0.015689	2.259e-01	2.581056	5.045e-01
GJR-GARCH	0.014825	3.834e-01	2.493900	9.989e-01
HAR	0.011489	2.247e-01	2.331576	2.589e-01
SV	6.781546	2.434e-34	68.241853	1.336e-63

Note: The table presents the results of the Diebold-Mariano (DM) test comparing the Mean Squared Error (MSE) and Mean Absolute Deviation (MAD) losses of various volatility forecasting models against the σ-Cell-RLTV model for S&P 500 index. The table reports the loss values scaled by 10^3 and the associated p-values. P-values below 0.05 indicate a statistically significant difference in performance from the σ-Cell-RLTV model.

Specifically, the σ-Cell, σ-Cell-N, σ-Cell-NTV, σ-Cell-RL, GARCH(1,1), and EGARCH models all have p-values below 0.05 in both metrics. The extremely low p-values associated with the SV model in both metrics, along with substantially higher loss values, highlight its significantly inferior performance compared to the HAR model.

The σ-Cell-RLTV model, in contrast, shows p-values greater than 0.05 in both metrics, suggesting that its performance is not significantly different from the HAR model. The TARCH and GJR-GARCH models have p-values below 0.05 in the MSE metric but greater than 0.05 in the MAD metric, indicating that their performance differs from the HAR model in terms of MSE but not in terms of MAD.

In summary, the σ-Cell-RLTV model shows comparable performance to the HAR model in forecasting S&P 500 volatility. The TARCH and GJR-GARCH models have mixed performance depending on the metric. In contrast, the σ-Cell, σ-Cell-N, σ-Cell-NTV, σ-Cell-RL, GARCH(1,1), EGARCH, and SV models exhibit statistically different performance from the HAR model in forecasting S&P 500 volatility.

Table 4.11 provides the results of a DM test comparing the performance of models

Table 4.10: S&P 500 Volatility Forecasting: Diebold-Mariano Test with HAR as the Base Model

Model	MSE Loss	MSE p-value	MAD Loss	MAD p-value
σ-Cell	0.018927	4.610e-04	2.902154	5.639e-04
σ-Cell-N	0.017707	1.192e-03	2.775868	3.550e-03
σ-Cell-NTV	0.017764	1.693e-03	2.856639	1.175e-03
σ-Cell-RL	0.018542	1.203e-04	2.868097	1.916e-03
σ-Cell-RLTV	0.013536	2.247e-01	2.494049	2.589e-01
GARCH(1,1)	0.015927	3.345e-03	2.625849	1.337e-02
EGARCH	0.016999	2.934e-04	2.806154	6.765e-05
TARCH	0.015689	4.127e-02	2.581056	6.787e-02
GJR-GARCH	0.014825	1.968e-02	2.493900	1.927e-01
HAR	0.011489	-	2.331576	-
SV	6.781546	2.444e-34	68.241853	3.198e-63

Note: The table presents the results of the Diebold-Mariano (DM) test comparing the Mean Squared Error (MSE) and Mean Absolute Deviation (MAD) losses of various volatility forecasting models against the HAR model for the S&P 500 index. The table reports the loss values scaled by 10^3 and the associated p-values. P-values below 0.05 indicate a statistically significant difference in performance from the HAR model.

to the σ-Cell-RLTV model, which serves as the base model for the BTCUSDT cryptocurrency.

The σ-Cell, σ-Cell-RL, GARCH(1,1), EGARCH, GJR-GARCH, and SV models all have p-values below 0.05 in the MAD metric, indicating a statistically significant difference in performance from the σ-Cell-RLTV model. In particular, the SV model exhibits substantially higher loss values and very low p-values in both metrics, highlighting its significantly inferior performance relative to the σ-Cell-RLTV model.

On the other hand, the σ-Cell-N and σ-Cell-NTV models show p-values greater than 0.05 in both metrics, suggesting that their performance is not significantly different from the σ-Cell-RLTV model. The TARCH model exhibits p-values above 0.05 in both metrics, indicating comparable performance to the σ-Cell-RLTV model as well.

The HAR model also shows p-values greater than 0.05 in both MSE and MAD metrics, indicating that its performance is not statistically different from the σ-Cell-RLTV model.

The σ-Cell and σ-Cell-RL models exhibit mixed performance with p-values below

Table 4.11: BTCUSDT Volatility Forecasting: Diebold-Mariano Test with σ-Cell-RLTV as the Base Model

Model	MSE Loss	MSE p-value	MAD Loss 10^3	MAD p-value
σ-Cell	0.344173	4.188e-02	11.560288	7.333e-05
σ-Cell-N	0.225319	7.433e-01	8.652624	4.743e-01
σ-Cell-NTV	0.230959	9.051e-01	8.693979	4.971e-01
σ-Cell-RL	0.279999	8.147e-02	10.219531	5.952e-03
σ-Cell-RLTV	0.234649	-	8.964701	-
GARCH(1,1)	0.439914	9.324e-02	11.768381	1.272e-03
EGARCH	0.502470	6.094e-02	11.917297	4.814e-03
TARCH	0.569893	1.244e-01	11.124331	8.147e-02
GJR-GARCH	0.386041	6.668e-02	10.315967	8.031e-02
HAR	0.261226	6.955e-01	8.720598	7.188e-01
SV	2.221157	2.886e-27	42.990330	1.941e-59

Note: The table presents the results of the Diebold-Mariano (DM) test comparing the Mean Squared Error (MSE) and Mean Absolute Deviation (MAD) losses of various volatility forecasting models against the σ-Cell-RLTV model for BTCUSDT cryptocurrency. The table reports the loss values scaled by 10^3 and the associated p-values. P-values below 0.05 indicate a statistically significant difference in performance from the σ-Cell-RLTV model.

0.05 in the MAD metric, but above 0.05 in the MSE metric, suggesting significant differences in performance from the σ-Cell-RLTV model in terms of MAD but not MSE. Similarly, the GARCH(1,1), EGARCH, and GJR-GARCH models have p-values below 0.05 in the MAD metric but above 0.05 in the MSE metric, indicating their performance differs from the σ-Cell-RLTV model in terms of MAD but not MSE.

In summary, the σ-Cell-N, σ-Cell-NTV, TARCH, and HAR models show comparable performance to the σ-Cell-RLTV model in forecasting BTCUSDT volatility, whereas the σ-Cell, σ-Cell-RL, GARCH(1,1), EGARCH, GJR-GARCH, and SV models exhibit statistically different performance in terms of MAD.

Table 4.12 presents the results of a DM test comparing the forecasting performance of various models to that of the HAR model, which serves as the base model for the BTCUSDT cryptocurrency.

The σ-Cell-N, σ-Cell-NTV, and σ-Cell-RLTV models exhibit strong performance in both MSE and MAD metrics, with high p-values indicating their performance is not significantly different from the HAR model. The SV model performs the worst with

Table 4.12: BTCUSDT Volatility Forecasting: Diebold-Mariano Test with HAR as the Base Model

Model	MSE Loss 10^3	MSE p-value	MAD Loss 10^3	MAD p-value
σ-Cell	0.344173	2.747e-01	11.560288	1.865e-04
σ-Cell-N	0.225319	5.879e-01	8.652624	9.194e-01
σ-Cell-NTV	0.230959	6.520e-01	8.693979	9.676e-01
σ-Cell-RL	0.279999	7.886e-01	10.219531	4.722e-02
σ-Cell-RLTV	0.234649	6.955e-01	8.964701	7.188e-01
GARCH(1,1)	0.439914	4.965e-03	11.768381	3.452e-07
EGARCH	0.502470	6.950e-03	11.917297	2.073e-05
TARCH	0.569893	8.848e-02	11.124331	1.264e-02
GJR-GARCH	0.386041	2.145e-01	10.315967	7.111e-02
HAR	0.261226	-	8.720598	-
SV	2.221157	6.309e-25	42.990330	1.001e-58

Note: The table presents the results of the Diebold-Mariano (DM) test comparing the Mean Squared Error (MSE) and Mean Absolute Deviation (MAD) losses of various volatility forecasting models against the HAR model for BTCUSDT cryptocurrency. The table reports the loss values scaled by 10^3 and the associated p-values. P-values below 0.05 indicate a statistically significant difference in performance from the HAR model.

extremely low p-values, indicating its performance is significantly worse than the HAR model. GARCH(1,1) and EGARCH models have low p-values in both MSE and MAD loss, suggesting their performance is significantly different from the HAR model.

The σ-Cell-RL model shows mixed performance, with a p-value below 0.05 in the MAD metric but above 0.05 in the MSE metric. This suggests that its performance is significantly different from the HAR model in terms of MAD but not MSE.

In summary, the σ-Cell-N, σ-Cell-NTV, and σ-Cell-RLTV models show comparable performance to the HAR model in forecasting BTCUSDT volatility. On the other hand, the GARCH(1,1), EGARCH, and SV models exhibit statistically different performance from the HAR model in both MSE and MAD metrics. The σ-Cell-RL model shows a nuanced performance, differing from the HAR model in terms of MAD but not MSE.

Table 4.13 presents the results of a Model Confidence Set (MCS) procedure conducted on various volatility forecasting models applied to two different financial markets, the S&P 500 index and the BTCUSDT cryptocurrency. The MCS method is employed to identify which models perform significantly better or worse than others in terms of MSE performance on the test data, using 10,000 bootstrap resamples. Three

metrics are reported for each model: MSE (scaled by 10^3), MCS p-values, and a designation for the set of models that perform at or above the 90% and 75% confidence levels, denoted as $\hat{\mathcal{M}}^*_{90,75\%}$.

For the S&P 500 index, the σ-Cell-RLTV model performs exceptionally well, achieving the lowest MSE among the σ-Cell variants at 0.0135 and a high MCS p-value of 0.791. This high p-value suggests that its performance is statistically indistinguishable from the best-performing model, the HAR model, which has the lowest overall MSE of 0.0114 and a high p-value of 0.839. All σ-Cell variants, as well as the GARCH(1,1), TARCH, and GJR-GARCH models, also perform well, with relatively low MSE values and high p-values, indicating their inclusion in the $\hat{\mathcal{M}}^*_{75\%}$ set. The EGARCH model, however, falls into the $\hat{\mathcal{M}}^*_{90\%}$ set due to its p-value of 0.111. The SV model performs the worst, with a high MSE of 6.7815 and a p-value of 0.000, suggesting it is not well-suited for forecasting S&P 500 volatility.

Turning to the BTCUSDT data, several σ-Cell variants exhibit the lowest MSE values, with the σ-Cell-N model performing best at an MSE of 0.2253. The σ-Cell-RLTV model stands out with a high MCS p-value of 0.945, closely followed by the σ-Cell-NTV and HAR models. All of these models, along with other σ-Cell variants, fall into the $\hat{\mathcal{M}}^*_{75\%}$ set, indicating their strong performance in forecasting BTCUSDT volatility. In contrast, the SV model performs the worst, with an MSE of 2.2211 and a p-value of 0.000, indicating poor predictive accuracy.

In summary, the σ-Cell-RLTV and HAR models are top performers for both the S&P 500 index and BTCUSDT data, showing superior forecasting abilities. Other σ-Cell variants and GARCH-type models also perform well across both data sets. However, the SV model consistently shows the least suitability among the models tested for both markets.

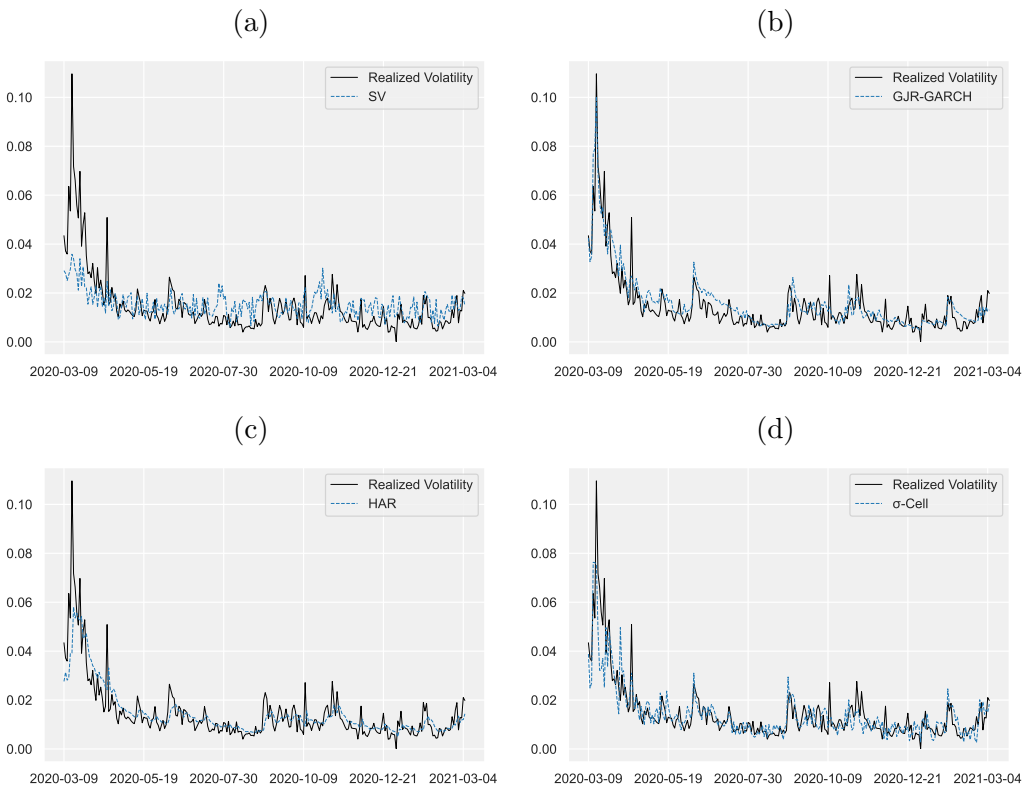

Figure 4.12: The following plot illustrates the prediction for in-sample realized volatility for the S&P 500 index. The presented plots provide a visual assessment of the performance of various models in predicting realized volatility. Each sub-figure displays the true realized volatility along with the model's estimate. (a) Stochastic Volatility (SV) model (b) GJR-GARCH model (c) HAR model (d) σ-Cell model **(Continued on next page.)**

Table 4.13: MCS with 10,000 bootstraps test sample

Model	S&P 500			BTCUSDT		
	MSE 10^3	P-value	$\hat{\mathcal{M}}^*_{90,75\%}$	MSE 10^3	P-value	$\hat{\mathcal{M}}^*_{90,75\%}$
σ-Cell	0.0189	0.117	*	0.3441	0.135	*
σ-Cell-N	0.0177	0.533	**	0.2253	0.617	**
σ-Cell-NTV	0.0177	0.325	**	0.2309	0.769	**
σ-Cell-RL	0.0185	0.639	**	0.2799	0.525	**
σ-Cell-RLTV	0.0135	0.791	**	0.2346	0.945	**
GARCH(1,1)	0.0159	0.444	**	0.4399	0.027	
EGARCH	0.0169	0.111	*	0.5024	0.039	
TARCH	0.0156	0.627	**	0.5698	0.000	
GJR-GARCH	0.0148	0.660	**	0.3860	0.037	
HAR	0.0114	0.839	**	0.2612	0.836	**
SV	6.7815	0.000		2.2211	0.000	

Note: The table presents the average loss over the test sample and the MCS p-values. The realized volatility forecasts in $\hat{\mathcal{M}}^*_{90\%}$ and $\hat{\mathcal{M}}^*_{75\%}$ are indicated by one and two asterisks, respectively. Values highlighted in bold indicate superior performance for the given Loss metric. In cases where multiple models exhibit closely matched performance, the top few models are highlighted to emphasize their comparative effectiveness.

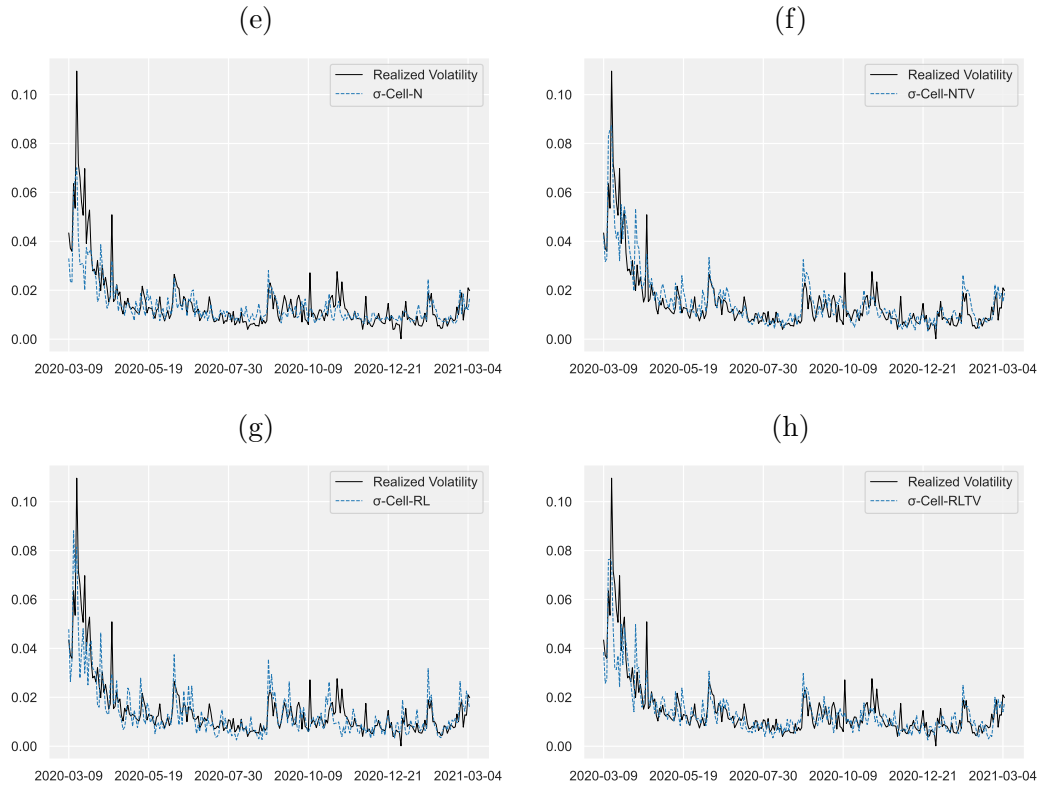

Figure 4.12: (Continued from previous page.) (e) σ-Cell-N model (f) σ-Cell-NTV model (g) σ-Cell-RL model (h) σ-Cell-RLTV model

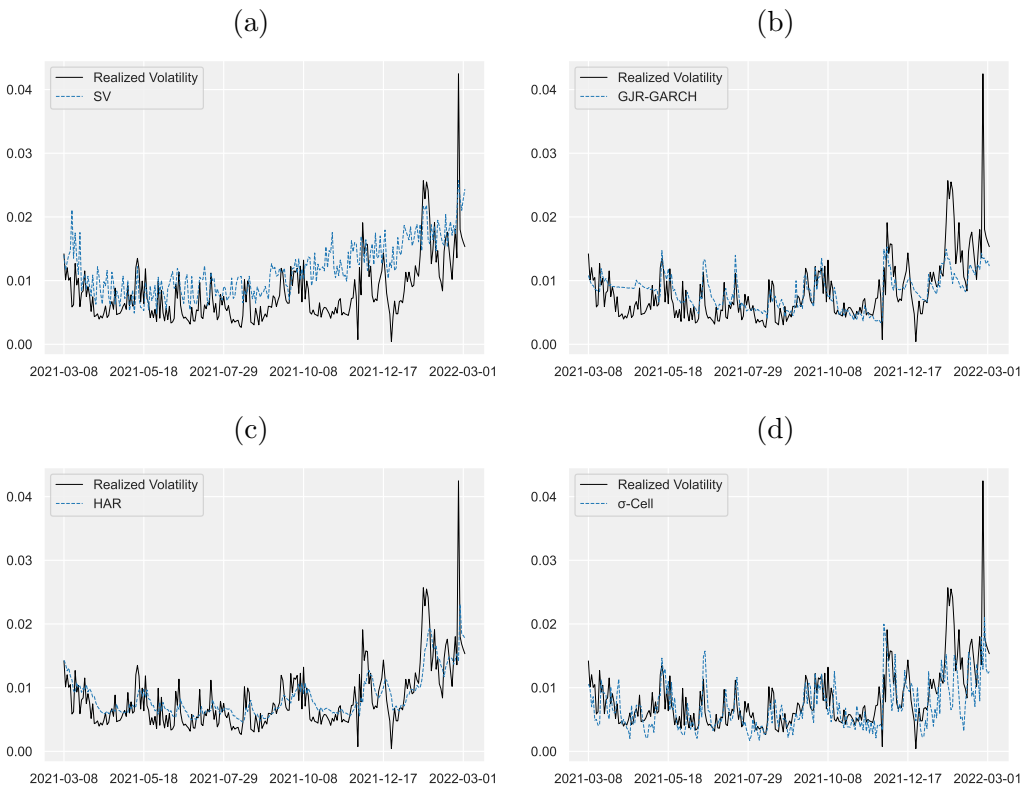

Figure 4.13: The following plot illustrates the prediction for out-of-sample realized volatility for the S&P 500 index. The presented plots provide a visual assessment of the performance of various models in predicting realized volatility. Each sub-figure displays the true realized volatility along with the model's 1-step ahead prediction. (a) Stochastic Volatility (SV) model (b) GJR-GARCH model (c) HAR model (d) σ-Cell model **(Continued on next page.)**

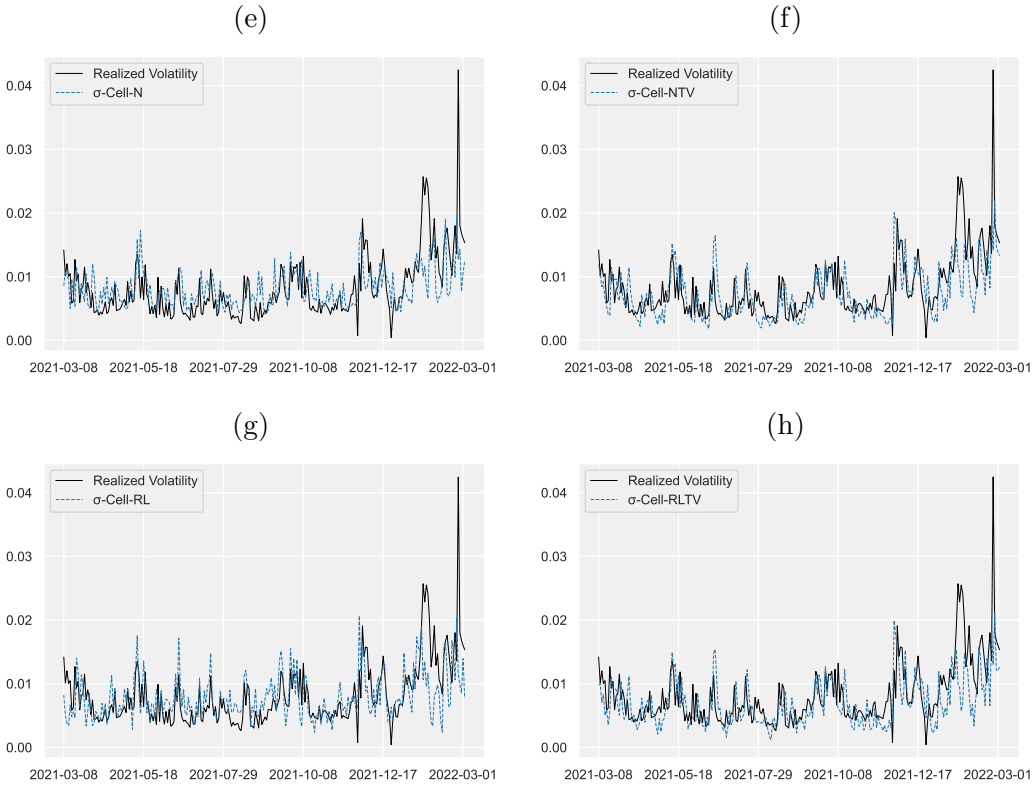

Figure 4.13: (Continued from previous page.) (e) σ-Cell-N model (f) σ-Cell-NTV model (g) σ-Cell-RL model (h) σ-Cell-RLTV model

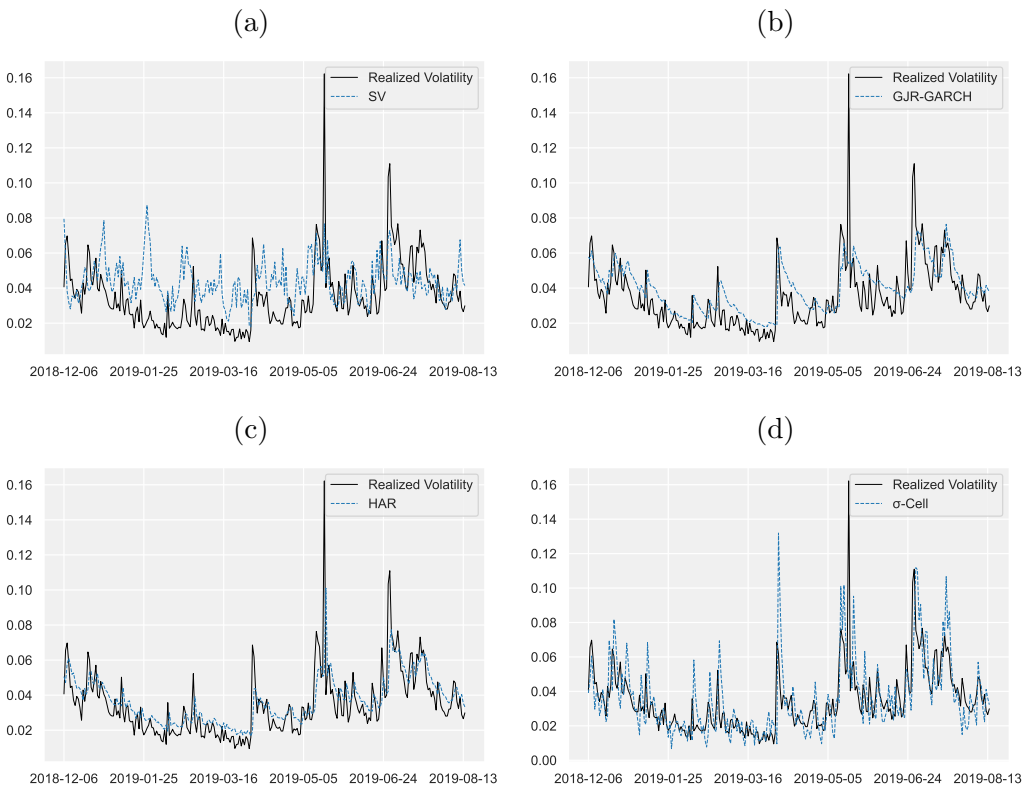

Figure 4.14: The following plot illustrates the prediction for in-sample realized volatility for the BTCUSDT. The presented plots provide a visual assessment of the performance of various models in predicting realized volatility. Each sub-figure displays the true realized volatility along with the model's estimate. (a) Stochastic Volatility (SV) model (b) GJR-GARCH model (c) HAR model (d) σ-Cell model **(Continued on next page.)**

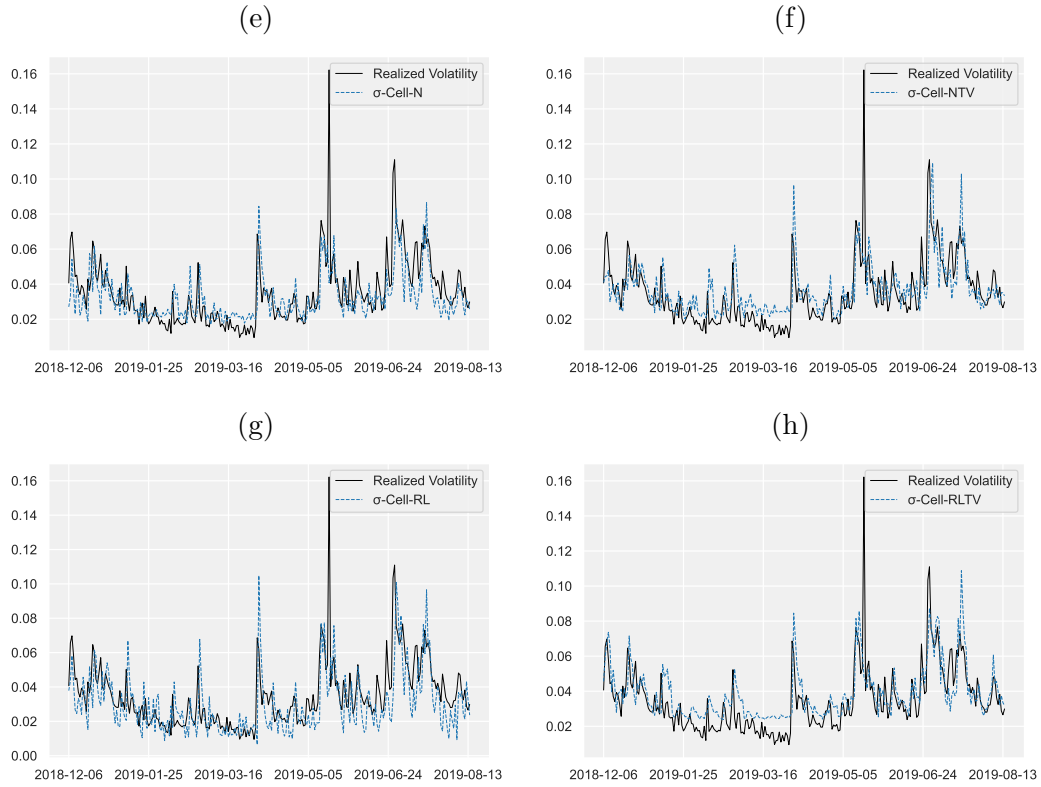

Figure 4.14: (Continued from previous page.) (e) σ-Cell-N model (f) σ-Cell-NTV model (g) σ-Cell-RL model (h) σ-Cell-RLTV model

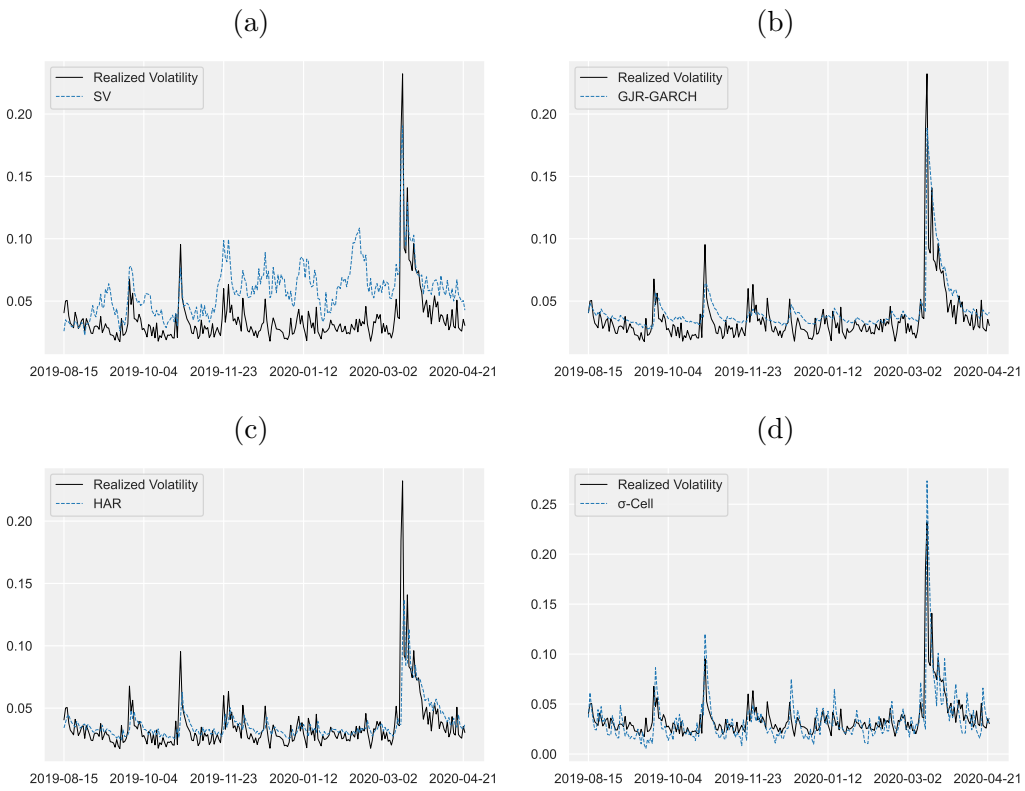

Figure 4.15: The following plot illustrates the predictions for out-of-sample realized volatility for the BTCUSDT pair. The presented plots provide a visual assessment of the performance of various models in predicting realized volatility. Each sub-figure displays the true realized volatility along with the model's 1-step ahead prediction. (a) Stochastic Volatility (SV) model (b) GJR-GARCH model (c) HAR model (d) σ-Cell model **(Continued on next page.)**

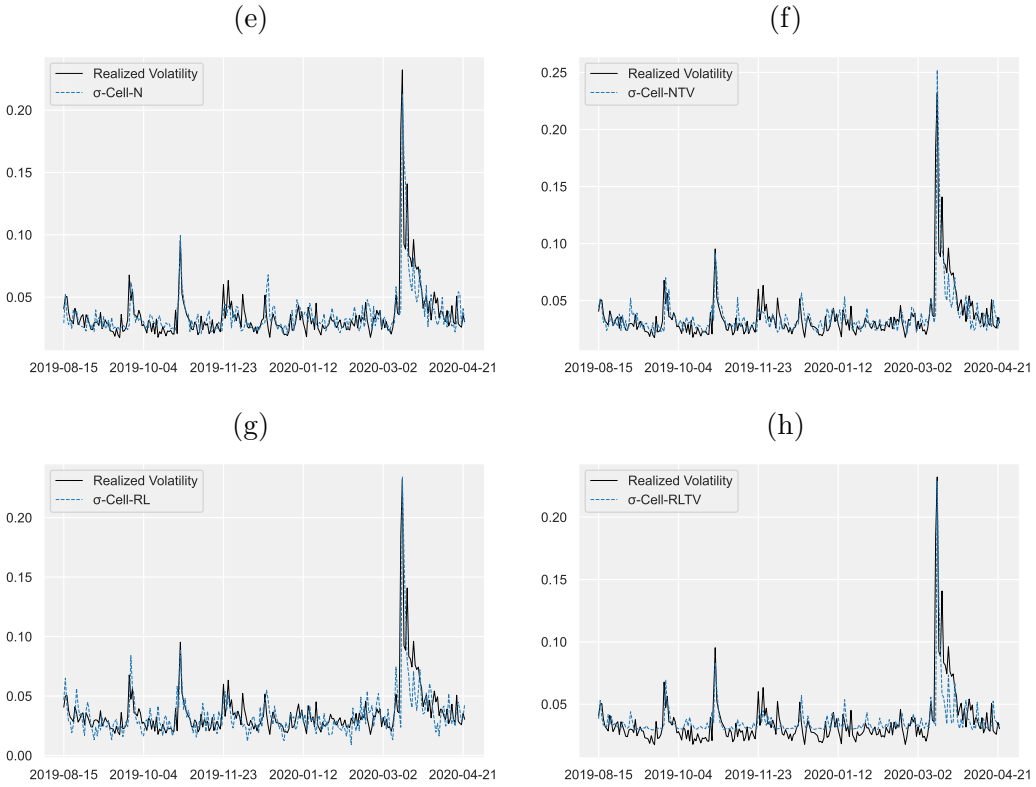

Figure 4.15: (Continued from previous page.) (e) σ-Cell-N model (f) σ-Cell-NTV model (g) σ-Cell-RL model (h) σ-Cell-RLTV model

Pairwise Comparisons using a linear regression framework

Tables 4.14a, 4.14a, 4.15a and 4.15b present the results of pairwise comparisons between different forecasting models for the S&P 500 and BTCUSDT, respectively, using a linear regression framework. These comparisons aim to determine which models provide valuable information for forecasting. The framework is based on a linear regression model of the form $y_{t+1} = \alpha_0 + \alpha_1 \hat{y}_{i,t+1} + \alpha_2 \hat{y}_{j,t+1} + u_t$, where $\hat{y}_{i,t+1}$ represents the forecast from the model in the $i-th$ row, $\hat{y}_{j,t+1}$ represents the forecast from the model in the $j-th$ column, and y_{t+1} is the actual value at time $t+1$. Tables 4.14a, 4.14a, 4.15a, and 4.15b display the estimated coefficients, α_1 and α_2, and their respective p-values for each pairwise comparison.

The significance levels are indicated with asterisks: * denotes significance at the 1% level, ** at the 5% level, and *** at the 10% level. These significance levels serve as evidence for or against the null hypothesis that the respective coefficient equals zero. A significant a1 coefficient suggests that the model in the row provides valuable information for forecasting, while a significant a2 coefficient suggests that the model in the column provides valuable information.

Overall, this analysis offers insights into the relative performance of different forecasting models for the S&P 500 and BTCUSDT.

The data presented in Tables 4.14a and 4.14b offer several insights into the comparative performance of the various forecasting models for the S&P 500 index.

- The σ-Cell models, including σ-Cell, σ-Cell-N, and σ-Cell-NTV, seem to have high significance with each other. This suggests that these models contain valuable information for forecasting the S&P 500 index.

- The σ-Cell-NTV model has high significance (indicated by ***) with the σ-Cell-RLTV model, showing strong evidence against the null hypothesis that the coefficient equals zero. This indicates that the σ-Cell-NTV model and the σ-Cell-RLTV model might have a relationship.

- The GARCH(1,1) model has high significance with the σ-Cell-RL and EGARCH models, which implies that it is highly informative for forecasting the S&P 500 index.

- Interestingly, the TARCH model seems to have significant coefficients with the GJR-GARCH model. This suggests that there might be a connection between these models when forecasting the S&P 500 index.

- The HAR model also exhibits significant coefficients with the SV model, indicating that these models might share valuable information for forecasting the S&P 500 index.

In summary, among the σ-Cell type models, σ-Cell-N and σ-Cell-RL seem to have strong performance overall, with significant coefficients against most of the other models. However, the performance of other σ-Cell type models is mixed and varies depending on the specific models being compared. The GARCH(1,1) and EGARCH models also seem to be significant in forecasting the S&P 500 index. The relationship between the TARCH and GJR-GARCH models, as well as the HAR and SV models, suggests that there may be shared information among these models that can be utilized for better forecasting of the S&P 500 index.

The data presented in Tables 4.15a and 4.15b offer several insights into the comparative performance of the various forecasting models for BTCUSDT.

- The σ models, such as σ-Cell, σ-Cell-N, and σ-Cell-NTV, seem to be highly significant with each other, indicating that they contain valuable information for forecasting the BTCUSDT pair.

- The σ-Cell model has high significance (indicated by ***) with GARCH(1,1), EGARCH, and TARCH models, showing strong evidence against the null hypothesis that the coefficient equals zero.

- The GARCH(1,1) model has high significance with HAR and SV models, which also suggests that it is highly informative for forecasting BTCUSDT.

In summary, among the σ-Cell type models, σ-Cell-N and σ-Cell-RL seem to have strong performance overall, with significant coefficients against most of the other models. However, the performance of other σ-Cell type models is mixed and varies depending on the specific models being compared.

(a) S&p 500 Pairwise Comparison of Forecasting Model Performance, coefficient a1

	σ-Cell	σ-Cell-N	σ-Cell-NTV	σ-Cell-RL	σ-Cell-RLTV	GARCH(1,1)	EGARCH	TARCH	GJR-GARCH	HAR	SV
σ-Cell	-	0.518***	-0.827**	0.507***	0.492	0.349***	0.315***	0.068	0.099	0.253***	0.440***
σ-Cell-N	0.308**	-	0.201	0.578***	0.338**	0.402***	0.382***	0.189*	0.209*	0.317***	0.512***
σ-Cell-NTV	1.458***	0.584***	-	0.522***	1.369***	0.427***	0.380***	0.104	0.151	0.259***	0.447***
σ-Cell-RL	0.436***	0.485***	0.409***	-	0.438***	0.464***	0.455***	0.264***	0.271***	0.506***	0.496***
σ-Cell-RLTV	0.226	0.493***	-0.731**	0.496***	-	0.326***	0.296***	0.042	0.070	0.248***	0.441***
GARCH(1,1)	0.801***	0.924***	0.621***	0.972***	0.827***	-	0.083	-0.442*	-0.379	0.347***	0.874***
EGARCH	0.955***	1.053***	0.793***	1.087***	0.979***	1.324**	-	-0.283	-0.270	0.392***	0.955***
TARCH	1.067***	1.011***	1.018***	0.953***	1.092***	1.425***	1.308***	-	0.894***	0.587***	0.820***
GJR-GARCH	1.070***	1.026***	0.993***	0.968***	1.101***	1.455***	1.364***	0.280	-	0.599***	0.847***
HAR	0.914***	0.937***	0.885***	0.959***	0.918***	0.930***	0.916***	0.733***	0.778***	-	0.792***
SV	0.580***	0.607***	0.558***	0.627***	0.585***	0.584***	0.570***	0.496***	0.526***	0.327***	-

(b) S&p 500 Pairwise Comparison of Forecasting Model Performance, coefficient a2

	σ-Cell	σ-Cell-N	σ-Cell-NTV	σ-Cell-RL	σ-Cell-RLTV	GARCH(1,1)	EGARCH	TARCH	GJR-GARCH	HAR	SV
σ-Cell	-	0.308**	1.458***	0.436***	0.226	0.801***	0.955***	1.067***	1.070***	0.914***	0.580***
σ-Cell-N	0.518***	-	0.584***	0.485***	0.493***	0.924***	1.053***	1.011***	1.026***	0.937***	0.607***
σ-Cell-NTV	-0.827**	0.201	-	0.409***	-0.731**	0.621***	0.793***	1.018***	0.993***	0.885***	0.558***
σ-Cell-RL	0.507***	0.578***	0.522***	-	0.496***	0.972***	1.087***	0.953***	0.968***	0.959***	0.627***
σ-Cell-RLTV	0.492	0.338**	1.369***	0.438***	-	0.827***	0.979***	1.092***	1.101***	0.918***	0.585***
GARCH(1,1)	0.349***	0.402***	0.427***	0.464***	0.326***	-	1.324**	1.425***	1.455***	0.930***	0.584***
EGARCH	0.315***	0.382***	0.380***	0.455***	0.296***	0.083	-	1.308***	1.364***	0.916***	0.570***
TARCH	0.068	0.189*	0.104	0.264***	0.042	-0.442*	-0.283	-	0.280	0.733***	0.496***
GJR-GARCH	0.099	0.209*	0.151	0.271***	0.070	-0.379	-0.270	0.894***	-	0.778***	0.526***
HAR	0.253***	0.317***	0.259***	0.506***	0.248***	0.347***	0.392***	0.587***	0.599***	-	0.327***
SV	0.440***	0.512***	0.447***	0.496***	0.441***	0.874***	0.955***	0.820***	0.847***	0.792***	-

Note: The table (a) and (b) present the results of pairwise comparisons between different forecasting models for S&p 500 using the framework of a linear regression model. The linear regression model is specified as $y_{t+1} = \alpha_0 + \alpha_1 \hat{y}_{i,t+1} + \alpha_2 \hat{y}_{j,t+1} + u_t$, where $\hat{y}_{i,t+1}$ represents the forecast from the model in the i-th row, $\hat{y}_{j,t+1}$ represents the forecast from the model in the j-th column, and y_{t+1} is the true value at time $t+1$. The table displays the estimated coefficients, α_1 and α_2, and their respective p-values for each pairwise comparison. The forecast evaluation period covers the last 252 trading days.
In Table (a), a1 coefficients and their significance levels are presented in the first part of the table. Each cell in this part of the table shows the a1 coefficient for the corresponding model pairing, along with asterisks indicating the significance level. Similarly, Table (b) shows the a2 coefficients and their significance levels.
Significance levels are indicated with asterisks: '*' denotes significance at the 1% level, '**' at the 5% level, and '***' at the 10% level. These significance levels serve as evidence for or against the null hypothesis that the respective coefficient equals zero. A significant a1 coefficient suggests that the model in the row provides valuable information for forecasting, while a significant a2 coefficient suggests that the model in the column provides valuable information.

(a) BTCUSDT Pairwise Comparison of Forecasting Model Performance, coefficient a1

	σ-Cell	σ-Cell-N	σ-Cell-NTV	σ-Cell-RL	σ-Cell-RLTV	GARCH(1,1)	EGARCH	TARCH	GJR-GARCH	HAR	SV
σ-Cell	-	0.172	0.156	0.450***	0.361***	0.613***	0.630***	0.594***	0.267***	0.405***	0.478***
σ-Cell-N	0.618***	-	0.411***	0.679***	0.614***	0.819***	0.888***	0.827***	0.423***	0.602***	0.680***
σ-Cell-NTV	0.606***	0.437***	-	0.700***	0.719***	0.763***	0.787***	0.764***	0.417***	0.574***	0.648***
σ-Cell-RL	0.206*	0.170*	0.107	-	0.270**	0.682***	0.677***	0.685***	0.256***	0.451***	0.578***
σ-Cell-RLTV	0.415**	0.297*	0.107	0.663***	-	0.908***	0.880***	0.887***	0.466***	0.646***	0.780***
GARCH(1,1)	-0.049	0.061	0.105	0.158**	0.179**	-	0.214*	0.350***	0.083	0.008	0.497***
EGARCH	-0.058	-0.058	0.019	0.090	0.134**	0.387***	-	0.341***	-0.107*	-0.091	0.371***
TARCH	0.006	0.023	0.059	0.083	0.142***	0.301***	0.199***	-	-0.032	-0.031	0.353***
GJR-GARCH	0.357***	0.322***	0.316***	0.410***	0.363***	0.538***	0.615***	0.574***	-	0.510***	0.455***
HAR	0.466***	0.426***	0.428***	0.588***	0.569***	1.044***	1.154***	1.082***	0.113	-	0.819***
SV	0.326***	0.295***	0.312***	0.348***	0.332***	0.518***	0.481***	0.512***	0.308***	0.357***	-

(b) BTCUSDT Pairwise Comparison of Forecasting Model Performance, coefficient a2

	σ-Cell	σ-Cell-N	σ-Cell-NTV	σ-Cell-RL	σ-Cell-RLTV	GARCH(1,1)	EGARCH	TARCH	GJR-GARCH	HAR	SV
σ-Cell	-	0.618***	0.606***	0.206*	0.415**	-0.049	-0.058	0.006	0.357***	0.466***	0.326***
σ-Cell-N	0.172	-	0.437***	0.170*	0.297*	0.061	-0.058	0.023	0.322***	0.426***	0.295***
σ-Cell-NTV	0.156	0.411***	-	0.107	0.107	0.105	0.019	0.059	0.316***	0.428***	0.312***
σ-Cell-RL	0.450***	0.679***	0.700***	-	0.663***	0.158**	0.090	0.083	0.410***	0.588***	0.348***
σ-Cell-RLTV	0.361***	0.614***	0.719***	0.270**	-	0.179**	0.134**	0.142***	0.363***	0.569***	0.332***
GARCH(1,1)	0.613***	0.819***	0.763***	0.682***	0.908***	-	0.387***	0.301***	0.538***	1.044***	0.518***
EGARCH	0.630***	0.888***	0.787***	0.677***	0.880***	0.214*	-	0.199***	0.615***	1.154***	0.481***
TARCH	0.594***	0.827***	0.764***	0.685***	0.887***	0.350***	0.341***	-	0.574***	1.082***	0.512***
GJR-GARCH	0.267***	0.423***	0.417***	0.256***	0.466***	0.083	-0.107*	-0.032	-	0.113	0.308***
HAR	0.405***	0.602***	0.574***	0.451***	0.646***	0.008	-0.091	-0.031	0.510***	-	0.357***
SV	0.478***	0.680***	0.648***	0.578***	0.780***	0.497***	0.371***	0.353***	0.455***	0.819***	-

Note: Tables (a) and (b) present the results of pairwise comparisons between different forecasting models for BTCUSDT using the framework of a linear regression model. The linear regression model is specified as $y_{t+1} = \alpha_0 + \alpha_1 \hat{y}_{i,t+1} + \alpha_2 \hat{y}_{j,t+1} + u_t$, where $\hat{y}_{i,t+1}$ represents the forecast from the model in the i-th row, $\hat{y}_{j,t+1}$ represents the forecast from the model in the j-th column, and y_{t+1} is the true value at time $t+1$. The table displays the estimated coefficients, α_1 and α_2, and their respective p-values for each pairwise comparison. The forecast evaluation period covers the last 252 trading days.
In Table (a), a1 coefficients and their significance levels are presented in the first part of the table. Each cell in this part of the table shows the a1 coefficient for the corresponding model pairing, along with asterisks indicating the significance level. Similarly, Table (b) shows the a2 coefficients and their significance levels.
Significance levels are indicated with asterisks: '*' denotes significance at the 1% level, '**' at the 5% level, and '***' at the 10% level. These significance levels serve as evidence for or against the null hypothesis that the respective coefficient equals zero. A significant a1 coefficient suggests that the model in the row provides valuable information for forecasting, while a significant a2 coefficient suggests that the model in the column provides valuable information.

4.7 Conclusion

In conclusion, our exploration of integrating a well-established econometric volatility model with RNNs has provided new models for volatility prediction. We introduced several designs of the new σ-Cell, drawing inspiration from the GARCH process, time-varying recurrent parameters, and inductive biases, thereby enhancing the model's capability to grasp the intricate temporal dynamics present in financial time series.

We employed a distinctive loss function grounded in a log-likelihood-based methodology to optimize the training process. Furthermore, we developed a specific version of the activation function, Adjusted-Softplus, to enhance the training process. We evaluated and compared the forecast performance of the proposed models with a well-established model in the field. The proposed σ-Cell-RLTV and σ-Cell-NTV models outperform traditional methods in out-of-sample predictive tasks, demonstrating the potential for significant advancements in econometric modeling techniques with deep learning.

The promising results obtained from our study pave the way for further explorations in integrating traditional econometric models and advanced neural network architectures. Such amalgamations can provide more precise and reliable predictions, crucial in various financial applications such as risk management, portfolio optimization, and algorithmic trading. Therefore, all innovations presented in this paper substantially enhance the capabilities of neural network-based volatility modeling.

4.8 Application: Algorithm for Volatility Prediction with σ-Cell-RLTV

Algorithm 1 σ-Cell-RLTV Algorithm for Volatility Prediction

Require: Sequence of inputs: $\mathbf{x} \in \mathbb{R}^n$
Ensure: Predicted volatility for the next input: σ_t

Initialize σ_0
for each time step $t = 1, 2, \ldots, n$ **do**
    Compute parameter vector w_t using Eq. 4.22:
        $w_t \leftarrow \tilde{\varphi}(\mathbf{W}\mathbf{x}_{t-1} + \mathbf{b})$
    Compute component $W_{s,t}$ using Eq. 4.23:
        $W_{s,t} \leftarrow \pi_1(w_t)$
    Compute component $W_{r,t}$ using Eq. 4.24:
        $W_{r,t} \leftarrow \pi_2(w_t)$
    Compute residual $\tilde{x}_{t-1}$ using Eq. 4.25:
        $\tilde{x}_{t-1} \leftarrow x_t - f(h_{t-1}, x_{t-1})$
    Compute estimated volatility $\tilde{\sigma}_t^2$ using Eq. 4.28:
        $\tilde{\sigma}_t^2 \leftarrow \phi(\tilde{\sigma}_{t-1}^2 W_{s,t} + \tilde{x}_{t-1}^2 W_{r,t} + b_h)$
    Predict σ_t^2 using Eq. 4.29:
        $\sigma_t^2 \leftarrow \phi_o(\tilde{\sigma}_t^2 W_o + b_o)$
end for
return Predicted volatility $\sigma_t \leftarrow \sqrt{\sigma_t^2}$

Algorithm 1 presents the σ-Cell-RLTV approach, which is designed to forecast the volatility of financial returns. The sequence of returns, denoted as x and with dimension n, serves as the primary input for this algorithm.

The parameter vector w_t is obtained by passing the input vector x_{t-1} through the function $\tilde{\varphi}$, which involves a linear transformation using the weight matrix $\mathbf{W}$ and the bias vector $\mathbf{b}$. Subsequently, the components Ws, t and $W_{r,t}$ are derived from w_t using the functions π_1, and π_2, respectively.

The sequence is passed through the layer described in Equation 4.22, which generates $\tilde{x}_{t-1}$. Then, for each time step t, ranging from 1 to n, $\tilde{x}_t$ is calculated as the difference

between the input and the layer $f\left(h_{t-1}, x_{t-1}\right)$, modulated by a function of the RNN's previous hidden state h_{t-1} and the actual input at that time.

Next, $\tilde{\sigma}_t^2$ is computed as a function of Ws, t, $\tilde{\sigma}_{t-1}^2$, Wr, t, and $\tilde{x}_t^2$. Finally, σ_t^2 is calculated using Equation 4.29, the function ϕ_o ensures that the estimated volatility remains positive. This process generates the estimated volatility for each return in the sequence.

Chapter 5

σ-LSTM

5.1 Motivation

Volatility models, which analyze price variations, have been a key area of study in econometrics. Both theoretical considerations and empirical evidence underpin these models. Recent developments in deep learning, particularly neural networks, have introduced new tools for econometric modeling. However, the application of neural networks to volatility modeling still lacks the incorporation of certain established patterns known as "stylized facts," which could enhance the predictive performance of the neural networks in volatility forecasting.

In this study, we advocate integrating stylized facts related to volatility dynamics, treated as an inductive bias, into the architecture of Long Short-Term Memory (LSTM) cells. This approach seeks to refine model performance. We introduce a novel LSTM cell variant, denoted as σ-LSTM, incorporating a stochastic processing layer. Our findings reveal that this model exhibits strong out-of-sample forecasting capabilities. Additionally, we enhance the training process by employing a specialized loss function derived from the loglikelihood method.

5.2 Introduction

Given the dynamic nature of financial markets, financial data modeling has long been a subject of intense research and scrutiny. Traditional regression models, such as

the linear regression model $\mathbf{y} = \mathbf{X}\beta + \epsilon$, has been foundational in understanding and predicting financial behaviors. In these models, $\mathbf{y}$ denotes the response variable. Meanwhile, β represents unobservable parameters, $\mathbf{X}$ is the non-random explanatory variable, and ϵ signifies the noise or error term.

However, real-world financial data complexities often challenge these models' assumptions. For instance, the assumption of homoscedasticity, where errors have constant variance and the absence of serial correlation between errors, is often violated in financial time series data. In scenarios where these assumptions are challenged, the Gauss-Markov theorem, which asserts the optimality of the ordinary least squares (OLS) estimator, becomes less applicable Huang et al. [1970]. To avoid redundancy, consider: "The field of econometrics has devoted significant effort to address these complexities, especially the heteroskedastic nature of financial data Tsay [2005].

As financial markets evolve, so does the need for more sophisticated models to capture the intricate dynamics of financial volatility. This study delves into volatility modeling, exploring both traditional and novel approaches. Our goal is to bridge the gap between econometric techniques and the capabilities of neural networks (NN).

In this study, we explore the volatility of asset returns, intrinsically marked by heteroscedasticity. This suggests that the variance of these returns is not constant but evolves over time. Volatility is intrinsically tied to risk and the extent of price variations.

From a conditional process standpoint, there are models that treat the variance of returns as a dynamic conditional parameter. Notably, the Autoregressive Conditional Heteroskedasticity (ARCH) model Engle [1982b] and the Generalized Autoregressive Conditional Heteroskedasticity (GARCH) model Bollerslev [1986] stand out. Central to these models is the idea that, based on historical data, the variance of returns can change over periods. However, the overall variance remains mostly stable.

In contrast, with the rise of high-frequency financial data, realized volatility (RV) has emerged as a strong alternative for volatility modeling. A prime example of this is the Heterogeneous Autoregression (HAR) model Corsi [2009]. Due to its robust predictive abilities and simple estimation methods, the HAR model is now a benchmark in the field. Realized volatility is preferred for its precision and dependability in

estimating market volatility.

Recently, several RNN architectures, like LSTM Hochreiter and Schmidhuber [1997c], Gated Recurrent Units (GRU)Cho et al. [2014a], and Statistical Recurrent Units (SRU)Oliva et al. [2017], have shown great promise in forecasting time-series data sets. Among these, the LSTM architecture stands out, especially for its exemplary performance in the intricate domain of volatility prediction Bucci [2020].

The Gated Recurrent Unit (GRU) is an LSTM variant that simplifies learning by removing the cell state. On the other hand, the Statistical Recurrent Unit (SRU) is designed to discern long-term dependencies in data. It achieves this by leveraging simple moving averages of summary statistics and linear combinations of historical data.

In the study by Nguyen et al. [2020], a unique RNN model was introduced that captures the ω-constant of the GARCH process. Another study Nguyen et al. [2022] introduced a hybrid approach that combined the Stochastic Volatility (SV) model with the SRU. While the SRU is adept at capturing long-term memory effects and the auto-dependence of volatility, in this SR-SV model, it precisely models the deterministic dynamics of the hidden states.

As the field of time series forecasting evolves rapidly, one emerging trend is integrating traditional volatility models with RNNs. The σ-Cell methodology, which combines the GARCH process with RNN dynamics, introduces a fresh perspective on volatility estimation Rodikov and Antulov-Fantulin [2023].

Although there has been a recent surge in interest in modeling the heteroskedasticity of returns using RNN architectures, there remains a gap. Prior research has not delved into this challenge from the perspective of a modified LSTM cell Nguyen et al. [2020, 2022].

Our research builds upon the work that introduced the σ-Cell models, the RNN cell design tailored to address the dynamic nature of financial volatility. This design integrates the principles of GARCH with a stochastic layer and time-varying parameters. Our study seeks to augment this foundational approach; while the original σ-Cell models employed a relatively straightforward RNN architecture, they were not without limitations, especially concerning long-term memory and issues of gradient

vanishing/exploding. We posit that the integration of LSTM networks, known for their proficiency in managing long-range dependencies and mitigating gradient-related challenges, can significantly enhance the σ-Cell models.

Our proposed σ-LSTM retains the features of the original σ-Cell models and utilizes a log-likelihood-based loss function. This addition aims to yield a more robust and precise volatility forecasting model. Our work continues and expands upon the original σ-Cell paradigm. It underscores the potential advantages of merging econometric price volatility models with deep learning methodologies, emphasizing the synergistic benefits of combining domain expertise with the computational power of neural networks.

5.3 Preliminaries

5.3.1 Baseline Volatility Models

In this study, we employ the GARCH(1,1) and GJR-GARCH models as our baseline for volatility forecasting. The GARCH(1,1) model is particularly renowned for its parsimony and effectiveness in capturing the stylized facts of financial time series, such as volatility clustering and leverage effects Francq and Zakoian [2010]. Its mathematical tractability and computational efficiency make it a widely used tool in both academic research and practical applications, serving as a benchmark for evaluating more complex models.

On the other hand, the GJR-GARCH model extends the basic GARCH framework by allowing for asymmetric responses to positive and negative shocks in financial returns Glosten et al. [1993b]. This feature is particularly relevant for capturing the "leverage effect," where negative returns tend to induce a higher subsequent volatility than positive returns of the same magnitude. By incorporating both the GARCH(1,1) and GJR-GARCH models as our baselines, we aim to provide a comprehensive and robust foundation for our experiments.

Our study also incorporates the HAR (Heterogeneous Autoregressive) model as an additional baseline for volatility forecasting Corsi [2009]. The HAR model has gained considerable attention for providing stable and accurate estimates of realized volatility over various time horizons Corsi et al. [2012]. Unlike traditional GARCH-type models,

which primarily focus on short-term volatility dynamics, the HAR model captures the long-memory property of volatility by aggregating information across multiple time scales.

The inclusion of the HAR model in our comparative analysis serves multiple purposes. First, it allows us to evaluate the performance of our proposed model against a well-established alternative that has been empirically validated for its forecasting accuracy Corsi et al. [2012]. Second, the HAR model's ability to integrate volatility information from a variety of market participants makes it a valuable addition to our model set, enriching the robustness and comprehensiveness of our analysis.

5.3.2 LSTM-RV

Recent advancements in forecasting methodologies have brought the LSTM model to the forefront, especially in the domains of stock and cryptocurrency markets. The empirical study we reviewed delves deep into the intricacies of LSTM configurations equations 2.22-2.27, Hochreiter and Schmidhuber [1997c], exploring the influence of various parameters such as the number of layers, neurons, activation functions, and optimizers. The study shows the model's capability to outperform established benchmarks. This has spurred our interest in assimilating the LSTM-RV model into our existing model set for a comprehensive comparative analysis Rodikov and Antulov-Fantulin [2022].

One of the pivotal aspects highlighted in the study is the role of hyperparameter tuning in enhancing the model's forecasting accuracy. Specifically, the input dimension size in the preprocessing procedure emerged as a critical parameter. The study suggests that setting the input dimension ranges from 5 to 12 can significantly improve the LSTM's performance Rodikov and Antulov-Fantulin [2022]. Such insights underscore the importance of meticulous parameter tuning in neural network architectures.

The versatility of the LSTM model is further exemplified by its adaptability across diverse markets. Whether it's the traditional stock market or the cryptocurrency domain, the LSTM model has demonstrated a unified approach to processing data. This adaptability is particularly noteworthy given these markets' inherent differences and volatility patterns. The model's ability to seamlessly transition between these markets is a testament to its robustness and potential for broader applications in the

future.

5.3.3 σ-Cell

In the evolving landscape of time series forecasting, integrating traditional volatility models with RNNs has emerged as a promising avenue. The σ-Cell approach, which utilizes the GARCH process with RNN dynamics, offers a novel perspective on volatility estimation 4.4. By incorporating these models into our comparison set, we aim to discern the potential improvements the σ-LSTM model might achieve.

The σ-Cell approach delineates a GARCH-like mechanism tailored for the estimation of conditional volatility inherent in a given time series. Central to this methodology is the explicit volatility estimation, σ_t, for every point within the time series. This dynamic volatility modeling determines σ_t^2 by integrating the current time series value with the preceding volatility and the corresponding error term. However, consider: "The approach offers dynamic mapping, allowing real-time modulation of the GARCH parameters and thus providing enhanced adaptability. The error quantification term captures the difference between actual and anticipated values, serving as a metric to train the model. This foundational approach sets the stage for its enhanced variants, which we delve into next.

The σ-Cell-N stands out due to its integration of a stochastic layer 4.6. This addition brings depth and complexity to its predictive capabilities. The stochastic layer introduces a component to the residuals, ensuring a continuous and coherent propagation of uncertainty through the series. This design choice, reminiscent of GARCH model dynamics, allows for nuanced volatility predictions. By combining past variance, disturbances, and a stochastic layer, the σ-Cell-N offers a comprehensive perspective on volatility dynamics 4.7, 4.8. This model exemplifies the blend of GARCH principles with neural architecture and stochasticity.

The σ-Cell-RLTV model represents another leap in the σ-Cell approach. This model marries the dynamic attributes of the time-varying method with the recurrent, residual features. The approach enhances memory retention and offers heightened resistance to specific noise disturbances. However, while it provides improved predictive capabilities, it also has an increased computational burden and a potential to overfit, especially in

sparser data data sets. This model's integration of time-varying weights and recurrent features showcases the potential of combining traditional volatility estimation techniques with modern neural architectures.

5.4 σ-LSTM

Accurate and timely volatility forecasting is vital for risk assessment, asset pricing, and investment decisions. This has spurred researchers to develop novel and effective models to predict financial volatility. One such innovative approach is the σ-Cell type model, a Recurrent Neural Network (RNN) cell design that integrates the principles of Generalized Autoregressive Conditional Heteroskedasticity (GARCH), a stochastic layer, and time-varying parameters 4.22-4.29. Our paper builds upon this foundational work by introducing new techniques and insights to enhance the σ-Cell model's effectiveness and forecasting performance.

The σ-Cell model employed a simple RNN architecture, which, despite its merits, has inherent limitations. In particular, RNNs are known to struggle with long-range dependencies and may face difficulties in learning patterns spanning long periods of time. Moreover, RNNs are susceptible to the gradient vanishing/exploding problem, which can hinder their training and convergence. We make an attempt to challenge it. We propose incorporating LSTM networks with the σ-Cell model, which we will refer to as the σ-LSTM. LSTM networks are specifically designed to handle long-range dependencies and alleviate the issues of gradient vanishing and exploding, making them an ideal candidate for enhancing the σ-Cell model.

Our σ-LSTM design integrates the novel features introduced in the original σ-Cell models, such as the GARCH principles, the stochastic layer, and the time-varying parameters. As well, σ-LSTM model employs a log-likelihood-based loss function, which aids in achieving a more robust and accurate volatility forecasting model 5.8. The inputs to the σ-LSTM are directly the returns $x_t = r_t$, and the outputs are the predicted predicted variance $\hat{\sigma_t}^2$. The cell maintains a hidden representation h_t for short-term memory and a long-term C_t volatility memory component, in line with the LSTM structure.

We aim to model asset returns $\mathrm{x_t}$ as a function of past returns $\phi(\mathrm{x_{t-1}}, ..., \mathrm{x_{t-p}})$, where $\phi()$ is a differentiable non-linear function. The σ-LSTM, a modified LSTM cell Hochreiter and Schmidhuber [1997c], is designed to capture both long and short-term volatility. The update rules of the σ-LSTM are as follows equations from 5.1 to 5.7.

$$f_t = \sigma_g \left(W_f x_t + U_f h_{t-1} + b_f \right) \tag{5.1}$$

$$i_t = \sigma_g \left(W_i x_t + U_i h_{t-1} + b_i \right) \tag{5.2}$$

$$\tilde{c}_t = \sigma_c \left(W_c x_t + U_c h_{t-1} + b_c \right) \tag{5.3}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{5.4}$$

$$o_t = \mathcal{N}(0, W_o[c_t^2]) \tag{5.5}$$

$$h_t = o_t \odot \sigma_h \left(c_t \right) \tag{5.6}$$

$$\hat{\sigma}_t^2 = \langle c_t \rangle^2 \tag{5.7}$$

$$\mathcal{L} = \sum_{t=1}^{m} \left[-\ln \left(\hat{\sigma}_t^2 \right) - \frac{x_t^2}{\hat{\sigma}_t^2} \right] \tag{5.8}$$

Our work represents a natural continuation and expansion of the original σ-Cell concept, further bridging the gap between econometric price volatility models and deep learning techniques. It showcases the potential benefits of inheriting domain knowledge from financial econometrics with the powerful predictive capabilities of neural networks. The advancements in the σ-LSTM model have the potential to contribute significantly to the ongoing quest for accurate and reliable financial volatility forecasting.

Intuition

The LSTM network was a groundbreaking Recurrent Neural Networks (RNNs) advancement. Its architecture was specifically designed to combat the challenges of long-range dependencies and the notorious gradient vanishing/exploding problem that plagued traditional RNNs. The LSTM's unique cell state and gating mechanisms (input, forget, and output gates) allowed it to maintain and manipulate information over extended sequences, making it a powerful tool for time series forecasting.

The main idea behind σ-LSTM is to combine the strengths of LSTMs and GARCH models. While LSTMs are good at capturing long-term dependencies in data, GARCH models are adept at modeling the time-varying nature of volatility. By combining these two, the σ-LSTM aims to accurately represent time series data with volatile patterns.

Traditional LSTMs have a mechanism to capture both long-term and short-term dependencies in data. This is achieved through the cell state c_t and the hidden state h_t. The cell state acts as the long-term memory, while the hidden state captures short-term dependencies.

Equations from 5.1 to 5.4 represent the canonical LSTM gates and cell state updates. They determine how information flows and is updated in the network.

The σ-LSTM introduces a stochastic layer, as depicted in equation 5.5. This layer, reminiscent of GARCH models, is tailored to capture the time-varying volatility. The quadratic dependence on c_t underscores the pivotal role of the cell state in ascertaining volatility. Unlike traditional LSTMs, which don't inherently factor in the stochastic nature of financial time series, the σ-LSTM, through its stochastic layer, integrates financial markets' inherent randomness and unpredictability.

The σ-LSTM postulates a relationship between the squared cell state (c_t^2) and volatility, as evident from equation 5.5. This deviation from traditional LSTMs aligns the model more closely with GARCH-like behavior. Further, equations 5.6 and 5.7 determine the hidden state and the network output, respectively.

The loss function, represented by 5.8, is designed to ensure that the σ-LSTM learns to minimize the negative log-likelihood of GARCH-like process, where σ-LSTM models time-varying volatility.

5.5 Experiments

This investigation delves into the efficacy of the proposed σ-LSTM model in gauging and forecasting the performance of diverse financial assets, specifically encompassing stocks, indices, and cryptocurrency. Our focal assets for this study are Apple Inc. stock, the S&P 500 index, and the Bitcoin-USD trading pair. The analysis is anchored on the realized volatility (RV) derived from minute-level price data, which is subsequently aggregated on a daily basis. The returns are ascertained from the daily closing prices 5.1.

For a structured approach, the data set was partitioned into three distinct segments: training, validation, and testing. The latter two segments, validation and testing, each comprise 252 data points, sourced from the concluding sections of the data set. A comprehensive statistical summary of the intraday returns for the S&P 500, Apple Inc., and BTCUSDT is presented in Table 5.2. This table elucidates key statistical metrics, including mean, median, standard deviation, skewness, and kurtosis, offering insights into the distributional characteristics of the returns.

To further enhance the understanding of the data distribution, visual representations are provided in Figure 5.3. These graphical depictions, encompassing histograms and box plots, illuminate the distribution, spread, and potential outliers in the returns for the S&P 500, Apple Inc., and Bitcoin. Such visual aids are instrumental in discerning the central tendency, variability, and overall distribution shape of the returns for the selected assets.

In our research, we employed a variety of models to assess their forecasting capabilities, utilizing a range of metrics and tests. We evaluated models using the Mean Absolute Error (MAE) and the Root Mean Squared Error (RMSE), both of which assess the magnitude of errors between predicted and observed values. The heteroskedasticity adjusted root mean square error (HRMSE) was another metric of interest, accounting for heteroscedasticity in the data Bollerslev and Ghysels [1996]. However, given the potential limitations of HRMSE, we also employed the QLIKE loss function, which is robust in the context of volatility forecasting Patton [2011]. We also explored R^2 for forecasting regressions Mincer and Zarnowitz [1969].

The Negative Log-Likelihood (NLL) metric was used to quantify the fit of our

models to the observed data. To compare the performance of various forecasting models, we utilized the Diebold-Mariano (DM) test, focusing on the Mean Squared Error (MSE) as loss functions Diebold and Mariano [1995] through the Model Confidence Set (MCS) test. The MCS test shows statistically significant differences between forecasting models Hansen et al. [2011]. This test, conducted using bootstrapping, allowed us to compare models and identify the best-performing ones with high confidence.

Table 5.1: Data Set Description for S&P 500, AAPL and BTCUSDT

Asset	Time Frame	From	To	RV Points
S&P 500	1 minute	10.03.07	01.03.22	3,800
APPL	1 minute	10.03.07	01.03.22	3,800
BTCUSDT	1 minute	01.01.13	20.04.20	2,667

Note: The table presents an overview of the data used in the analysis, including the asset, the time frame of the data, the date range of the data, and the number of realized volatility (RV) points. The last 252 data points are used for out-of-sample testing, and the preceding 252 data points are used for validation.

Table 5.2: Statistical Summary of Returns for S&P 500, AAPL, and BTCUSDT.

Asset	Mean	Median	STD	Skewness	Kurtosis
S&P 500	0.00029	0.0007	0.0128	-0.1886	13.9915
APPL	0.00104	0.0010	0.0212	-0.4026	9.17605
BTCUSDT	0.00349	0.0020	0.0466	-0.0726	14.3508

Note: The table presents key statistical metrics of the interdaily returns for the S&P 500 index and the Bitcoin-USD (BTCUSDT) trading pair. The metrics include mean, median, standard deviation (STD), skewness, and kurtosis. A positive skewness indicates a right-side heavier tail of the probability density function, while a negative skewness indicates a left-side heavier tail. Kurtosis measures the "tailedness" of the probability distribution of returns. Higher kurtosis indicates a heavier tail, signifying a higher probability of extreme outcomes. For the analysis, we use the entire data set without dividing it into validation and test sets.

In the pursuit of refining the neural network's architecture, a comprehensive exploration of the hyperparameter space was undertaken, guided by established methodologies Panchal et al. [2010]. Post-evaluation on both training and validation

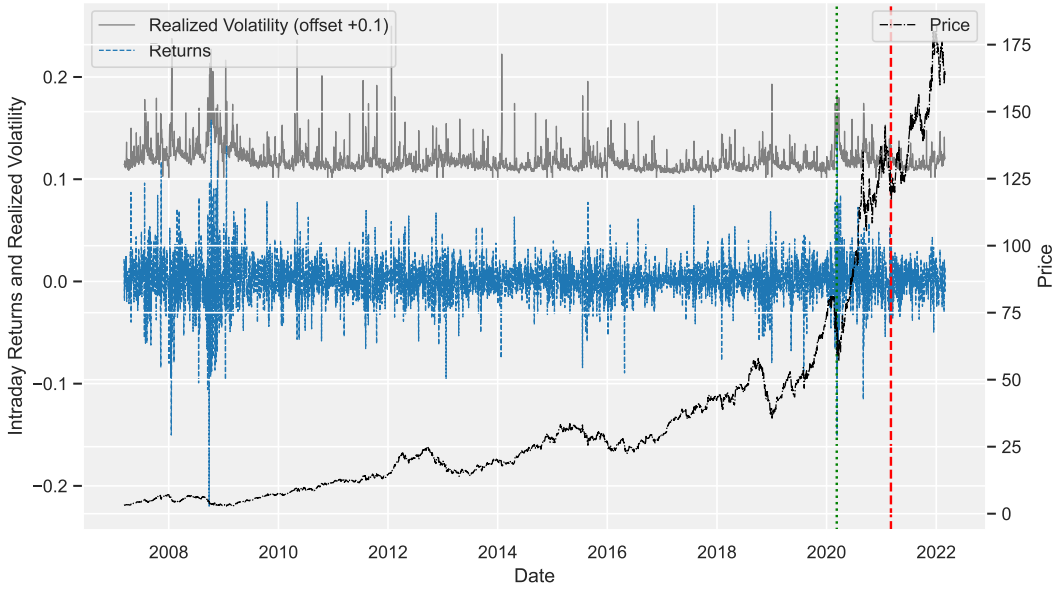

Figure 5.1: This plot depicts the Realized Volatility, Returns, and Price of Apple Inc. stock between 10th March 2007 and 1st March 2022. The gray solid line represents the realized volatility (offset by +0.1), the blue dashed line represents intraday returns, and the black dash-dot line indicates the price. Both the RV and returns, derived from daily data, are presented on the primary y-axis, while the price is shown on a secondary y-axis. The vertical red dashed and green dotted lines demarcate the beginnings of the test and validation sets, respectively, with each set comprising 252 points. All remaining data serve as the training set. For a similar analysis on the S&P 500 and BTCUSDT, see Chapter 4, Figure 4.3, 4.5.

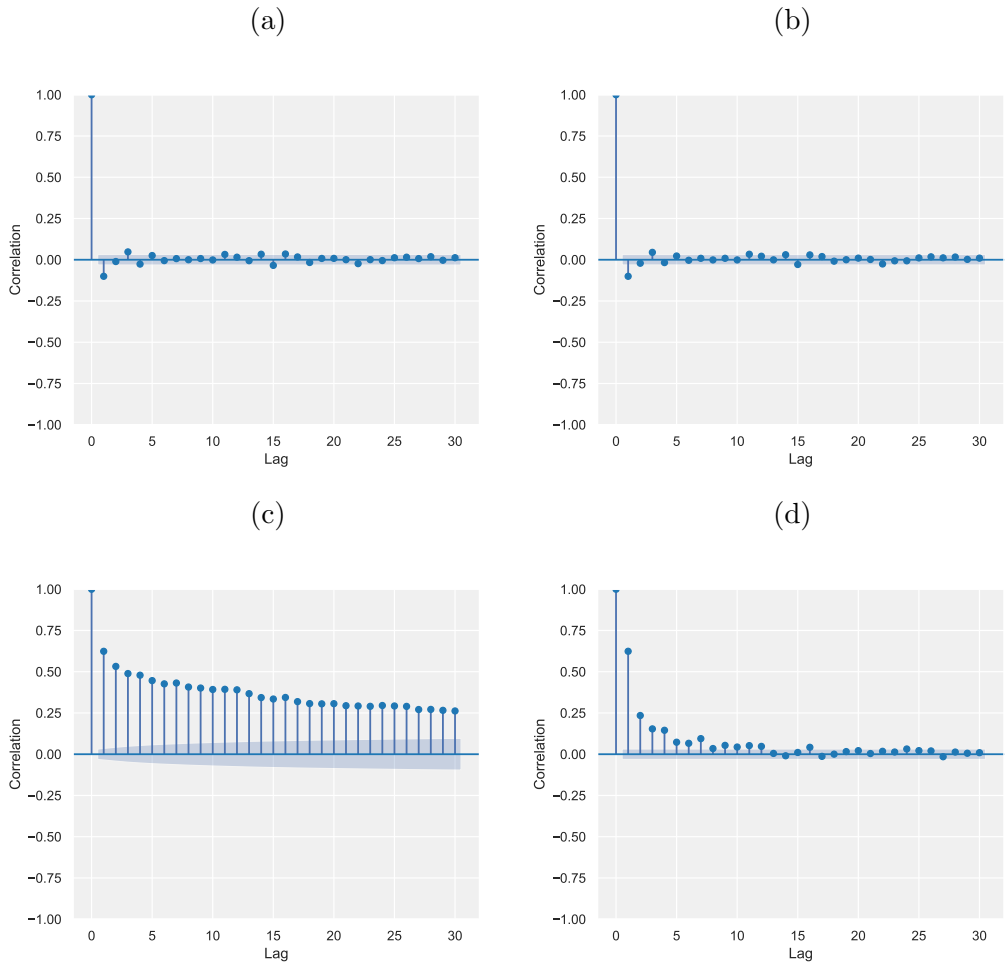

Figure 5.2: This plot displays the autocorrelation (ACF) and partial autocorrelation (PACF) for the returns and volatility of Apple Inc. stock. The ACF illustrates the extent of a linear relationship between current values and their lags, while the PACF captures the correlation between a value and its lag that isn't explained by shorter lags. a) ACF for returns b) PACF for returns c) ACF for volatility d) PACF for volatility. For a similar analysis on the S&P 500 and BTCUSDT, see Chapter 4, Figure 4.4, 4.6.

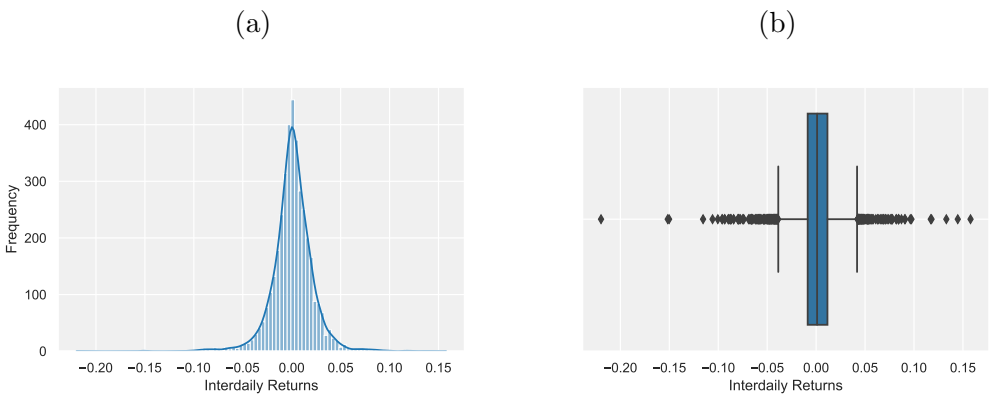

Figure 5.3: This plot showcases the distribution and Box Plot of Returns for the Apple Inc. stock. Panel (a) displays the histogram of the returns, while Panel (b) presents its box plot, highlighting the spread of data and potential outliers. These visualizations offer insights into the central tendency, dispersion, and shape of the return distribution. For a similar analysis on the S&P 500 and BTCUSDT, see Chapter 4, Figure 4.7.

data sets, the selection of optimal hyperparameters was informed by the loss criterion.

Adam, which stands for Adaptive Moment Estimation, is a stochastic optimization technique frequently employed in training deep learning models. Its popularity stems from its efficiency and minimal memory demands. At its core, Adam adjusts learning rates for individual parameters adaptively Kingma and Ba [2014].

5.6 Results

In the subsequent section, we present empirical results derived from our analysis. These results are tabulated to provide a clear and concise overview of the performance metrics associated with various forecasting models.

For the in-sample performance on the S&P 500 index, the σ-LSTM model outperforms the σ-Cell-N model in terms of MAE, RMSE, and R^2, indicating superior precision and fit. Out-of-sample tests further validate its effectiveness, although the LSTM-RV model, which operates on RV values, exhibits the highest efficacy. Generally, the GARCH(1,1) and GJR-GARCH models show moderate performance across the evaluated data sets. For the S&P 500 index, both models perform better than the σ-Cell-N model in terms of MAE and RMSE but are outperformed by the HAR and LSTM-RV models. Their R^2 values indicate a reasonable, but not exceptional, fit to the out-of-sample data.

In the case of Apple Inc. stock, the σ-LSTM model again surpasses the σ-Cell-N model in both in-sample and out-of-sample metrics, particularly in MAE and RMSE. The model's HRMSE and R^2 values underscore its robustness in forecasting stock volatility. For Apple Inc. stock, the GARCH(1,1) model has higher MAE and RMSE values compared to the GJR-GARCH model, but both are outperformed by the σ-LSTM and LSTM-RV models in most metrics. The GJR-GARCH model shows a slightly better R^2 value compared to GARCH(1,1), suggesting a more robust in-sample fit.

For the BTCUSDT trading pair, the σ-LSTM model demonstrates a robust in-sample fit, outperforming the σ-Cell-N model in RMSE and R^2. In out-of-sample tests, it exhibits the lowest HRMSE among the models, although it is slightly outperformed by the LSTM-RV and HAR models in other metrics. For the BTCUSDT trading pair, the GARCH(1,1) model performs relatively poorly, especially in out-of-sample tests. Its R^2 values are also lower, indicating a less robust fit. The GJR-GARCH model shows a slightly better R^2 value in out-of-sample tests. While these models offer a decent baseline, they are generally outperformed by more complex models like LSTM-RV and σ-LSTM in both in-sample and out-of-sample evaluations.

The LSTM-RV model demonstrates the best performance in MAE and RMSE among the evaluated models and data sets for out-of-sample tests. It also achieves the highest R^2 value, indicating the most robust fit to the out-of-sample data. The HAR model follows closely and suggests a reasonable fit but is not as strong as the LSTM-RV model. Interestingly, the σ-LSTM model shows competitive performance using only daily returns instead of realized volatility as LSTM-RV and HAR-RV. While it does not outperform the LSTM-RV and HAR models in terms of MAE and RMSE, it is noteworthy that the margin of difference is relatively small. Moreover, its R^2 values are closely aligned with those of the LSTM-RV and HAR models, and its HRMSE is the lowest among the models. The QLIKE metric further underscores its strong alignment with the out-of-sample data.

The Model Confidence Set (MCS) approach is used to identify a set of models that, at a given confidence level, contain the best model 5.9. The p-values in the MCS context determine if a model can be excluded from the superior set of models. A higher p-value indicates that the model cannot be confidently excluded from the best set,

Table 5.3: In-Sample Performance Metrics for S&P 500 Volatility Forecasting Models

Model	MAE 10^3	RMSE 10^3	HRMSE	QLIKE	R^2
σ-Cell-N	4.2755	7.3268	3.9454	-3.37082	0.703984
σ-Cell-RLTV	4.2681	6.7396	4.6134	-3.36883	0.72224
GARCH(1,1)	5.3125	8.2736	3.349	-3.36648	0.660096
GJR-GARCH	5.1116	8.193	3.2048	-3.36603	0.667675
HAR	4.1901	7.3104	3.7523	-3.3793	0.675116
LSTM-RV	3.839	6.5666	3.566	-3.39294	0.741149
σ-LSTM	4.0836	6.6643	3.4432	-3.37075	0.739557

Note: The table presents the in-sample performance of various volatility forecasting models applied to the S&P 500 index. The performance metrics include Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Heteroscedasticity-Adjusted RMSE (HRMSE), Quasi-Likelihood (QLIKE), and the coefficient of determination (R^2). Metrics MAE, RMSE are evaluated at a scale of 10^3.

Table 5.4: Out-of-Sample Performance Metrics for S&P 500 Volatility Forecasting Models

Model	MAE 10^3	RMSE 10^3	HRMSE	QLIKE	R^2
σ-Cell-N	2.7759	4.2079	2.4303	-3.85495	0.25181
σ-Cell-RLTV	2.4940	3.6792	2.1279	-3.86195	0.464835
GARCH(1,1)	2.6258	3.9908	2.6406	-3.868	0.319658
GJR-GARCH	2.4939	3.8503	2.6177	-3.87638	0.372308
HAR	2.3316	3.3896	2.6567	-3.88498	0.516026
LSTM_RV	1.9963	3.0925	2.4229	-3.89967	0.626805
σ-LSTM	2.4092	3.6171	2.341	-3.87699	0.44324

Note: The table presents the out-of-sample performance of various volatility forecasting models applied to the S&P 500 index. The performance metrics include Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Heteroscedasticity-Adjusted RMSE (HRMSE), Quasi-Likelihood (QLIKE), and the coefficient of determination (R^2). Metrics MAE, RMSE are evaluated at a scale of 10^3.

Table 5.5: In-Sample Performance Metrics for AAPL Volatility Forecasting Models

Model	MAE 10^3	RMSE 10^3	HRMSE	QLIKE	OOS R^2
σ-Cell-N	6.0582	8.8993	2.7425	-2.78755	0.602733
σ-Cell-RLTV	5.5996	8.0974	2.6727	-2.79491	0.619464
GARCH(1,1)	6.7251	9.2401	2.604	-2.78454	0.57517
GJR-GARCH	5.9343	8.3618	2.6604	-2.78979	0.633342
HAR	6.2071	9.3115	2.7176	-2.77971	0.485976
LSTM-RV	5.5129	8.5075	2.5675	-2.79159	0.581793
σ-LSTM	5.5366	7.7249	2.6381	-2.79878	0.645268

Note: The table presents the in-sample performance of various volatility forecasting models applied to the Apple Inc. stock. The performance metrics include Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Heteroscedasticity-Adjusted RMSE (HRMSE), Quasi-Likelihood (QLIKE), and the coefficient of determination (R^2). Metrics MAE, RMSE are evaluated at a scale of 10^3.

Table 5.6: Out-of-Sample Performance Metrics for AAPL Volatility Forecasting Models

Model	MAE 10^3	RMSE 10^3	HRMSE	QLIKE	OOS R^2
σ-Cell-N	4.0946	6.0104	2.3161	-3.149	0.410853
σ-Cell-RLTV	4.4316	6.1656	2.2868	-3.1443	0.254376
GARCH(1,1)	5.3866	6.9943	2.4557	-3.1264	0.047298
GJR-GARCH	5.0156	6.7116	2.4476	-3.1333	0.115035
HAR	3.8178	5.6268	2.3846	-3.1544	0.325561
LSTM-RV	3.3902	5.122	2.2991	-3.1637	0.440552
σ-LSTM	4.0008	5.7351	2.2825	-3.1531	0.419388

Note: The table presents the out-of-sample performance of various volatility forecasting models applied to the Apple Inc. stock. The performance metrics include Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Heteroscedasticity-Adjusted RMSE (HRMSE), Quasi-Likelihood (QLIKE), and the coefficient of determination (R^2). Metrics MAE, RMSE are evaluated at a scale of 10^3.

Table 5.7: In-Sample Performance Metrics for BTCUSDT Volatility Forecasting Models

Model	MAE 10^3	RMSE 10^3	HRMSE	QLIKE	R^2
σ-Cell-N	9.5845	14.8304	2.0743	-2.40586	0.383233
σ-Cell-RLTV	9.8975	13.9467	2.2753	-2.41649	0.490842
GARCH(1,1)	10.645	15.2788	2.2737	-2.40314	0.339935
GJR-GARCH	10.4658	16.0143	2.2568	-2.39506	0.278871
HAR	8.3432	13.046	2.1962	-2.43033	0.519423
LSTM-RV	7.0565	11.3759	2.1466	-2.44429	0.636916
σ-LSTM	10.0536	14.7675	1.9886	-2.38417	0.45642

Note: The table presents the in-sample performance of various volatility forecasting models applied to the BTCUSDT trading pair. The performance metrics include Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Heteroscedasticity-Adjusted RMSE (HRMSE), Quasi-Likelihood (QLIKE), and the coefficient of determination (R^2). Metrics MAE, RMSE are evaluated at a scale of 10^3.

Table 5.8: Out-of-Sample Performance Metrics for BTCUSDT Volatility Forecasting Models

Model	MAE 10^3	RMSE 10^3	HRMSE	QLIKE	R^2
σ-Cell-N	8.6526	15.0106	2.0952	-2.3861	0.54771
σ-Cell-RLTV	8.9647	15.3183	2.1562	-2.3821	0.512133
GARCH(1,1)	11.7684	20.9741	2.2144	-2.3579	0.164693
GJR-GARCH	10.316	19.6479	2.1261	-2.3712	0.561019
HAR	8.7206	16.1625	2.1609	-2.3901	0.462834
LSTM-RV	7.3282	14.1654	2.1167	-2.4046	0.598796
σ-LSTM	8.9932	14.7252	2.0425	-2.3789	0.575294

Note: The table presents the out-of-sample performance of various volatility forecasting models applied to the BTCUSDT trading pair. The performance metrics include Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Heteroscedasticity-Adjusted RMSE (HRMSE), Quasi-Likelihood (QLIKE), and the coefficient of determination (R^2). Metrics MAE, RMSE are evaluated at a scale of 10^3.

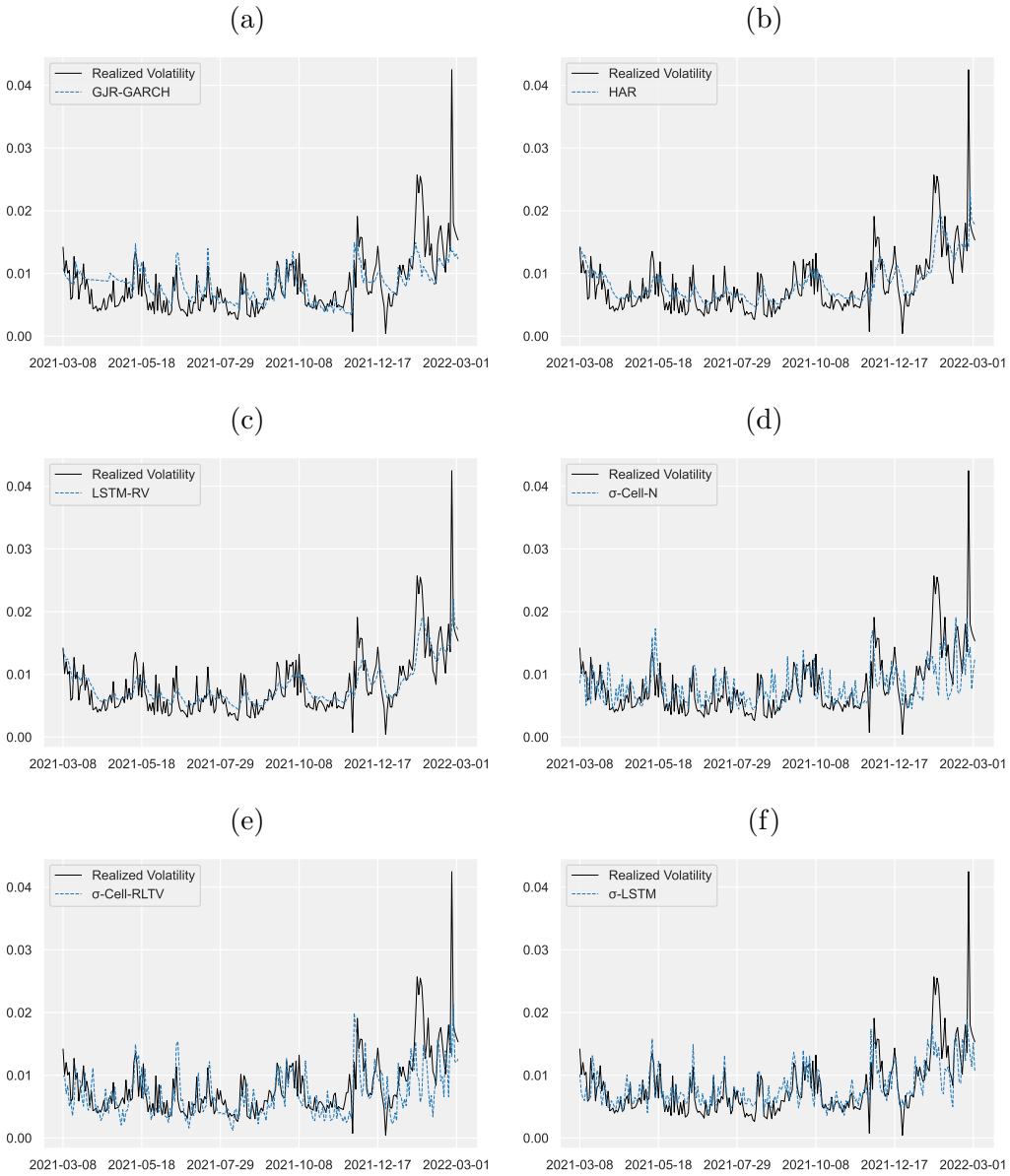

Figure 5.4: The following plot illustrates the prediction for out-of-sample BTCUSDT pair data for Different Forecasting Models: (a) GJR-GARCH model's prediction. (b) HAR model's prediction. (c) LSTM-RV model's prediction. (d) σ-Cell-N model's prediction. (e) σ-Cell-RLTV model's prediction. (f) σ-LSTM model's prediction.

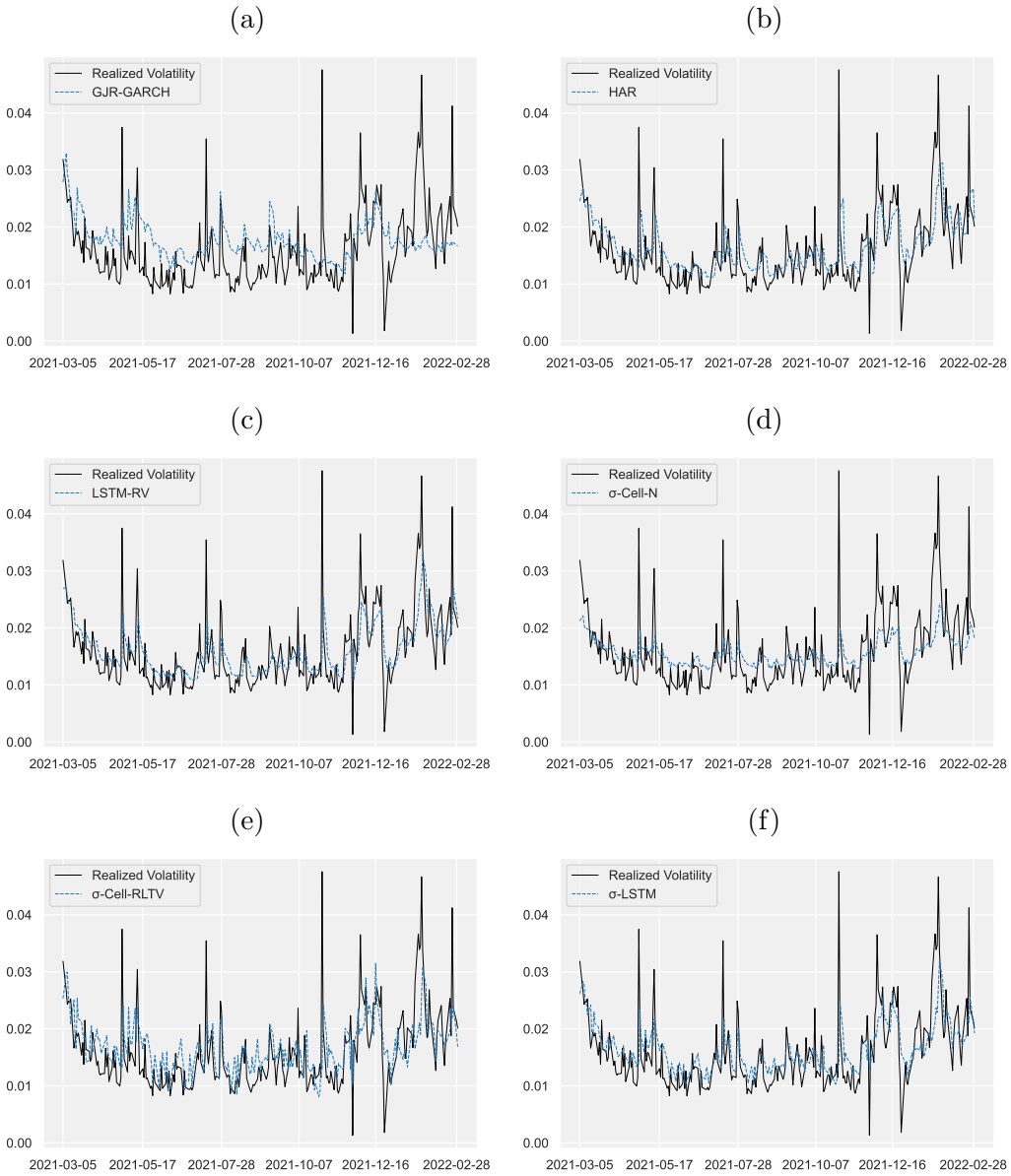

Figure 5.5: The following plot illustrates the prediction for out-of-sample Apple Inc. Stock data for Different Forecasting Models: (a) GJR-GARCH model's prediction. (b) HAR model's prediction. (c) LSTM-RV model's prediction. (d) σ-Cell-N model's prediction. (e) σ-Cell-RLTV model's prediction. (f) σ-LSTM model's prediction.

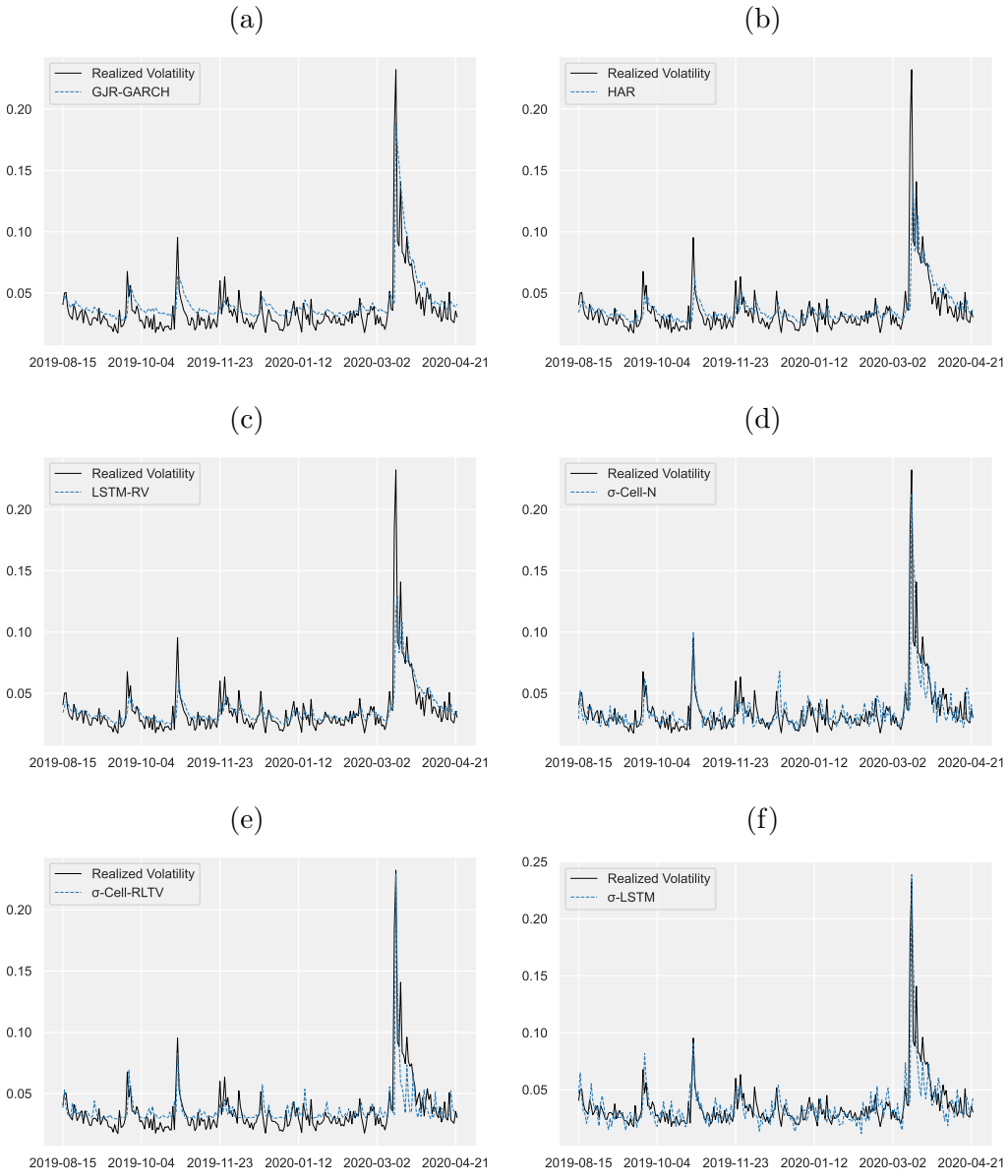

Figure 5.6: The following plot illustrates the prediction for out-of-sample BTCUSDT pair data for Different Forecasting Models: (a) GJR-GARCH model's prediction. (b) HAR model's prediction. (c) LSTM-RV model's prediction. (d) σ-Cell-N model's prediction. (e) σ-Cell-RLTV model's prediction. (f) σ-LSTM model's prediction.

Table 5.9: MCS with 10,000 bootstraps test sample

Model	S&P 500		BTCUSDT		APPL	
	MSE 10^3	P-value	MSE 10^3	P-value	MSE 10^3	P-value
σ-Cell-N	0.0177	0.413**	0.2253	0.391**	0.0361	0.571**
σ-Cell-RLTV	0.0135	0.708**	0.2346	0.132*	0.0380	0.332**
GARCH(1,1)	0.0159	0.000	0.4399	0.005	0.0489	0.000
GJR-GARCH	0.0148	0.450**	0.3860	0.000	0.0450	0.419**
HAR	0.0114	0.374**	0.2612	0.601**	0.0316	0.546**
LSTM-RV	0.0095	0.600**	0.2006	0.422**	0.0262	0.127*
σ-LSTM	0.0130	0.491**	0.2168	0.524**	0.0328	0.654**

Note: The table presents the average loss over the test sample and the MCS p-values. The realized volatility forecasts in $\hat{\mathcal{M}}^*_{90\%}$ and $\hat{\mathcal{M}}^*_{75\%}$ are indicated by one and two asterisks, respectively. Values highlighted in bold indicate superior performance for the given Loss metric. In cases where multiple models exhibit closely matched performance, the top few models are highlighted to emphasize their comparative effectiveness.

while a lower p-value suggests the opposite.

Table 5.9 presents the average loss over the test sample and the Model Confidence Set (MCS) p-values for various volatility forecasting models across three distinct data sets: S&P 500, BTCUSDT, and Apple Inc. (AAPL).

Models marked with two asterisks (**) cannot be confidently excluded from the set of best models at the 90% confidence level. Similarly, models with a single asterisk (*) are part of the superior set at the 75% confidence level.

For instance, for the S&P 500 data set, the GARCH(1,1) model, with a p-value of 0.000, can be confidently excluded from the best set of models. On the other hand, models like the σ-LSTM, with a p-value of 0.491^{**}, remain in the superior set at the 90% confidence level.

Across the three data sets examined, the σ-LSTM model consistently demonstrated good forecasting capabilities. It is particularly noteworthy that the σ-LSTM model consistently outperformed the σ-Cell-N model across all data sets, highlighting the improved predictive capabilities of the LSTM architecture. Utilizing the Model Confidence Set (MCS) approach, it becomes evident that the σ-LSTM model frequently ranks among the top-performing models, reinforcing its reliability across different data sets.

The σ-LSTM model showcased a balance between accuracy and consistency in each

data set. The consistent performance across diverse data sets underscores the model's versatility and reliability in financial forecasting. Future research and applications might consider the σ-LSTM model as a reliable counterpart for volatility forecasting endeavors.

5.7 Conclusion

The rapidly evolving landscape of time series forecasting has witnessed the amalgamation of traditional volatility models with the prowess of Recurrent Neural Networks (RNNs). This study introduces the σ-LSTM model, a specialized cell that seamlessly integrates the stochastic layer into LSTM architectures, thereby enhancing the predictive performance for financial volatility. Instead of approaching the LSTM as a mere black box, our research endeavors to engineer it with a deliberate focus, incorporating long-short volatility memory and a stochastic element, thus embedding a physics-informed inductive bias into the model.

Our results are promising. The σ-LSTM model consistently outperforms several established models in volatility prediction, including the robust GARCH family baseline. This underscores the potential advantages of merging domain-specific knowledge from financial econometrics with the computational capabilities of deep learning. Tailored loss functions further accentuate the model's performance, offering a more robust and precise volatility forecasting approach.

Furthermore, using the Model Confidence Set approach, the σ-LSTM model solidifies its position among the top-performing models across diverse data sets. This attests to the model's reliability and its versatility in handling different financial data sets.

In light of these findings, the σ-LSTM model bridges the gap between traditional econometric models and modern neural network architectures, offering a solution that capitalizes on the strengths of both parts.

As we look ahead, our future endeavors will explore more sophisticated loss functions and other potential enhancements. The goal is to refine the σ-LSTM model further, ensuring quicker learning convergence and even greater accuracy in volatility forecasting.

Chapter 6

Further research questions

The journey of this research has been both enlightening and challenging, offering a new perspective on volatility modeling by integrating machine learning techniques—specifically, LSTM-based recurrent neural networks—with traditional econometric models. While the results are promising, they also lay the groundwork for future research and development. Below are some directions for future work based on the findings of this thesis.

Enhancing Model Efficiency and Scalability. Chapter 3 highlighted the high computational cost associated with the large number of parameters in LSTM models. Future work could explore model pruning techniques or more efficient architectures that maintain predictive accuracy while reducing computational overhead. Such improvements would make the models more scalable and applicable in real-time trading environments where quick decision-making is crucial.

Extending the σ-Cell Framework. Chapter 4 introduced the innovative σ-Cell framework, successfully integrating traditional econometric models like GARCH with RNNs. Future research could extend this framework to include other well-known econometric models, such as Stochastic Volatility models or even jump-diffusion models Bates [1996], to capture more complex market behaviors like sudden jumps or crashes.

Loss Function and Activation Function Innovations. Chapter 4 also highlighted the use of specialized loss and activation functions. Future work could explore alternative loss functions and activation functions that might be better suited for financial time-series data, aiming to optimize the training process further and

improve out-of-sample predictive performance.

Expanding the σ-LSTM Model. Chapter 5 introduced the σ-LSTM model to address potential challenges of σ-Cell with gradients. Future research could refine this model by incorporating more complex loss functions and architectures. Additionally, the model could be tested on additional financial data, such as options or commodities, to assess its versatility. σ-LSTM Model is a task-specific model; the utility of the model is highly task-specific; for the task involves modeling or predicting variances, it could be instrumental. However, introducing stochastic elements could affect the stability and convergence of the training process for different tasks.

The alternative update rule. The alternative equation for o_t combines the standard sigmoid-based gating mechanism with a stochastic term $\mathcal{N}(0, W_o \cdot [c_t^2])$, equation 6.1.

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \odot \mathcal{N}(0, W_o \cdot [c_t^2]) \tag{6.1}$$

Multivariate time series investigation. While the primary focus of this research has been on univariate time series data, it is worth noting that the σ-RNN and σ-LSTM frameworks are not limited to this scope. These architectures could be used for multivariate time series without any modifications. Financial markets are interconnected ecosystems with multiple variables. A multivariate extension of the σ-RNN and σ-LSTM models could capture these intricate relationships, providing a more holistic view of market dynamics. It would be particularly useful for multi-asset portfolio management, where understanding the correlations and covariances between different assets is essential for effective risk diversification.

Given the potential benefits of a multivariate approach, future research could investigate the σ-RNN and σ-LSTM frameworks to handle multivariate time series data. One key challenge would be handling the increased complexity and computational requirements of multivariate data. However, with advancements in computational techniques and hardware, this is a feasible and promising direction for future research. Experimental studies could compare the performance of the multivariate σ-RNN and σ-LSTM models against traditional multivariate GARCH models or other machine learning approaches in capturing asset-specific and systemic risks in financial markets.

In summary, the work presented in this thesis lays a foundation for applying RNN techniques to volatility modeling. It also identifies numerous opportunities for future research to address the limitations and challenges that have been discussed. By bridging the gap between traditional econometric models and modern machine learning techniques, we can aspire to develop more accurate, efficient, and interpretable models for financial market volatility.

Chapter 7

Conclusion

Volatility modeling and forecasting are critical endeavors in the financial domain, serving as the backbone for risk assessment, derivative pricing, and investment decision-making. The landscape of volatility prediction has been fertile ground for both traditional econometric models and, more recently, machine learning techniques. This thesis has aimed to bridge the gap between these two paradigms by offering an approach that leverages the strengths of both to achieve superior predictive performance.

The first line of inquiry focused on applying Long Short-Term Memory (LSTM) networks to predict realized volatility. We found that LSTM models, particularly those with optimized hyperparameters and appropriate input dimensions, offer a compelling alternative to traditional models like HAR and GARCH-family models. The LSTM-RV model, in particular, demonstrated superior out-of-sample accuracy across different market types, thereby confirming the utility of neural networks in capturing the complex temporal structures inherent in financial markets. However, the black-box nature and computational intensity of these models remain challenges that warrant further investigation.

Our second contribution, the σ-Cell, represents a fusion of traditional econometric models with recurrent neural networks. This hybrid approach leverages the GARCH process and introduces time-varying recurrent parameters, thereby enhancing the ability to capture intricate temporal dynamics. The use of specialized loss functions and activation functions further optimized the training and generalization processes. The σ-Cell models outperformed traditional methods in out-of-sample predictive tasks,

suggesting that integrating econometric models with deep learning techniques can significantly advance the field.

Finally, the σ-LSTM model is the culmination of our research. It offers a specialized LSTM cell incorporating a stochastic layer and a process-informed inductive bias. This model outperforms several established models and demonstrates versatility across diverse financial data sets. The Model Confidence Set approach further solidified its position as a reliable and robust method for volatility forecasting.

The overarching theme of this thesis is the symbiotic relationship between traditional econometric models and modern machine-learning techniques. While each has merits and limitations, their integration can lead to more accurate, interpretable, and versatile models. This is particularly important in the context of financial markets, where dynamics are complex.

Future research should address the limitations of the current models, such as the black-box nature of neural networks and the computational intensity of training. Additionally, exploring more sophisticated loss functions and other potential enhancements could refine the models, ensuring quicker learning convergence and even greater accuracy.

In summary, this thesis contributes to the ongoing discourse on volatility modeling by offering a nuanced approach that combines the knowledge of traditional econometric models with the flexibility and computational power of neural networks. This methodology is a basis for future research questions and practical applications, enriching our understanding of financial market volatility and offering robust and adaptable approaches.

Bibliography

Torben G Andersen and Tim Bollerslev. Answering the skeptics: Yes, standard volatility models do provide accurate forecasts. *International economic review*, pages 885–905, 1998.

Torben G Andersen, Tim Bollerslev, Francis X Diebold, and Paul Labys. Modeling and forecasting realized volatility. *Econometrica*, 71(2):579–625, 2003.

Torben G Andersen, Tim Bollerslev, and Francis X Diebold. Roughing it up: Including jump components in the measurement, modeling, and forecasting of return volatility. *The review of economics and statistics*, 89(4):701–720, 2007.

Torben G Andersen, Dobrislav Dobrev, and Ernst Schaumburg. Jump-robust volatility estimation using nearest neighbor truncation. *Journal of Econometrics*, 169(1):75–93, 2012.

Timotheos Angelidis and Stavros Degiannakis. Volatility forecasting: Intra-day versus inter-day models. *Journal of International Financial Markets, Institutions and Money*, 18(5):449–465, 2008.

Timotheos Angelidis, Alexandros Benos, and Stavros Degiannakis. The use of garch models in var estimation. *Statistical methodology*, 1(1-2):105–128, 2004.

Timotheos Angelidis, Stavros Degiannakis, et al. *Backtesting var models: An expected shortfall approach*. SSRN, 2008.

Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International conference on machine learning*, pages 1120–1128. PMLR, 2016.

Josip Arnerić, Tea Poklepović, and Zdravka Aljinović. Garch based artificial neural networks in forecasting conditional variance of stock returns. *Croatian Operational Research Review*, pages 329–343, 2014.

Josip Arnerić, Tea Poklepović, and Juin Wen Teai. Neural network approach in forecasting realized variance using high-frequency data. *Business Systems Research: International journal of the Society for Advancing Innovation and Research in Economy*, 9(2):18–34, 2018.

Philippe Artzner, Freddy Delbaen, Jean-Marc Eber, and David Heath. Coherent measures of risk. *Mathematical finance*, 9(3):203–228, 1999.

Susan Athey and Guido W Imbens. Machine learning methods for estimating heterogeneous causal effects. *stat*, 1050(5):1–26, 2015.

Louis Bachelier. Théorie de la spéculation. In *Annales scientifiques de l'École normale supérieure*, volume 17, pages 21–86, 1900.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Maria Maddalena Barbieri and James O Berger. Optimal predictive model selection. 2004.

Ole E Barndorff-Nielsen and Neil Shephard. Econometric analysis of realized volatility and its use in estimating stochastic volatility models. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 64(2):253–280, 2002a.

Ole E Barndorff-Nielsen and Neil Shephard. Estimating quadratic variation using realized variance. *Journal of Applied econometrics*, 17(5):457–477, 2002b.

David S Bates. Jumps and stochastic volatility: Exchange rate processes implicit in deutsche mark options. *The Review of Financial Studies*, 9(1):69–107, 1996.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

Fischer Black. The pricing of commodity contracts. *Journal of financial economics*, 3 (1-2):167–179, 1976.

Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–654, 1973.

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

Tim Bollerslev and Eric Ghysels. Periodic autoregressive conditional heteroscedasticity. *Journal of Business & Economic Statistics*, 14(2):139–151, 1996.

Tim Bollerslev, Andrew J Patton, and Rogier Quaedvlieg. Exploiting the errors: A simple approach for improved volatility forecasting. *Journal of Econometrics*, 192 (1):1–18, 2016.

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

Chris Brooks and Gita Persand. Volatility forecasting for risk management. *Journal of forecasting*, 22(1):1–22, 2003.

Andrea Bucci. Realized volatility forecasting with neural networks. *Journal of Financial Econometrics*, 18(3):502–531, 2020.

Andrea Bucci et al. Forecasting realized volatility: a review. *Journal of Advanced Studies in Finance (JASF)*, 8(16):94–138, 2017.

Kenneth P Burnham and David R Anderson. Multimodel inference: understanding aic and bic in model selection. *Sociological methods & research*, 33(2):261–304, 2004.

Rodolfo C Cavalcante, Rodrigo C Brasileiro, Victor LF Souza, Jarley P Nobrega, and Adriano LI Oliveira. Computational intelligence and financial markets: A survey and future directions. *Expert Systems with Applications*, 55:194–211, 2016.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014a.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014b.

Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. volume 28, 2015.

Peter Christoffersen, Kris Jacobs, Chayawat Ornthanalai, and Yintian Wang. Option valuation with long-run and short-run volatility components. *Journal of Financial Economics*, 90(3):272–297, 2008.

Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. *Advances in neural information processing systems*, 28, 2015.

Marc Claesen and Bart De Moor. Hyperparameter search in machine learning. *arXiv preprint arXiv:1502.02127*, 2015.

Todd Clark and Michael McCracken. Advances in forecast evaluation. *Handbook of economic forecasting*, 2:1107–1201, 2013.

Todd E Clark and Michael W McCracken. Tests of equal forecast accuracy and encompassing for nested models. *Journal of econometrics*, 105(1):85–110, 2001.

Michael P Clements and Hans-Martin Krolzig. A comparison of the forecast performance of markov-switching and threshold autoregressive models of us gnp. *The Econometrics Journal*, 1(1):47–75, 1998.

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

Fulvio Corsi. A simple approximate long-memory model of realized volatility. *Journal of Financial Econometrics*, 7(2):174–196, 2009.

Fulvio Corsi, Francesco Audrino, and Roberto Renó. Har modeling for realized volatility forecasting. -, 2012.

Anthony Christopher Davison and David Victor Hinkley. *Bootstrap methods and their application*. Number 1. Cambridge university press, 1997.

Francis X Diebold. *Elements of forecasting*. Citeseer, 1998.

Francis X Diebold. Comparing predictive accuracy, twenty years later: A personal perspective on the use and abuse of diebold–mariano tests. *Journal of Business & Economic Statistics*, 33(1):1–1, 2015.

Francis X Diebold and Roberto S Mariano. Com paring predictive accu racy. *Journal of Business and Economic Statistics*, 13(3):253–263, 1995.

Zhuanxin Ding, Clive WJ Granger, and Robert F Engle. A long memory property of stock market returns and a new model. *Journal of empirical finance*, 1(1):83–106, 1993.

Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-fourth international joint conference on artificial intelligence*, 2015.

Kevin Dowd. *Measuring market risk*. John Wiley & Sons, 2007.

Timothy Dozat. Incorporating nesterov momentum into adam. 2016.

Darrell Duffie, Jun Pan, and Kenneth Singleton. Transform analysis and asset pricing for affine jump-diffusions. *Econometrica*, 68(6):1343–1376, 2000.

Dutta and Shekhar. Bond rating: a nonconservative application of neural networks. In *IEEE 1988 International Conference on Neural Networks*, pages 443–450. IEEE, 1988.

Bradley Efron and Robert J Tibshirani. *An introduction to the bootstrap*. CRC press, 1994.

R. F. Engle. Autoregressive conditional heteroskedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, 50(4):987–1007, 1982a.

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

Robert F Engle and Giampiero M Gallo. A multiple indicators model for volatility using intra-daily data. *Journal of econometrics*, 131(1-2):3–27, 2006.

Robert F Engle and Kenneth F Kroner. Multivariate simultaneous generalized arch. *Econometric theory*, 11(1):122–150, 1995.

Robert F Engle, Eric Ghysels, and Bumjean Sohn. Stock market volatility and macroeconomic fundamentals. *Review of Economics and Statistics*, 95(3):776–797, 2013.

Bjørn Eraker, Michael Johannes, and Nicholas Polson. The impact of jumps in volatility and returns. *The Journal of Finance*, 58(3):1269–1300, 2003.

Eugene F Fama. The behavior of stock-market prices. *The journal of Business*, 38(1): 34–105, 1965.

Eugene F Fama. Market efficiency, long-term returns, and behavioral finance. *Journal of financial economics*, 49(3):283–306, 1998.

Matthias Feurer and Frank Hutter. Hyperparameter optimization. *Automated machine learning: Methods, systems, challenges*, pages 3–33, 2019.

Stephen Figlewski. Estimating the implied risk neutral density. 2008.

Thomas Fischer and Christopher Krauss. Deep learning with long short-term memory networks for financial market predictions. *European journal of operational research*, 270(2):654–669, 2018.

Marco Fraccaro, Søren Kaae Sønderby, Ulrich Paquet, and Ole Winther. Sequential neural models with stochastic layers. *Advances in neural information processing systems*, 29, 2016.

C. Francq and J. M. Zakoian. *GARCH models: structure, statistical inference and financial applications*. John Wiley & Sons, 2010.

Christian Francq and Jean-Michel Zakoïan. Risk-parameter estimation in volatility models. *Journal of Econometrics*, 184(1):158–173, 2015.

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

Philip Hans Franses and Dick Van Dijk. Forecasting stock market volatility using (non-linear) garch models. *Journal of forecasting*, 15(3):229–235, 1996.

Ana-Maria Fuertes, Marwan Izzeldin, and Elena Kalotychou. On forecasting daily stock volatility: The role of intraday information and market conditions. *International Journal of Forecasting*, 25(2):259–281, 2009.

Jim Gatheral. The volatility surface: a practitioner's guide. jon wiley & sons. *Inc., Hoboken, New Jersey*, 2006.

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

Raffaella Giacomini and Halbert White. Tests of conditional predictive ability. *Econometrica*, 74(6):1545–1578, 2006.

Pierre Giot. Market risk models for intraday data. *The European Journal of Finance*, 11(4):309–324, 2005.

X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

Lawrence R Glosten, Ravi Jagannathan, and David E Runkle. On the relation between the expected value and the volatility of the nominal excess return on stocks. *The journal of finance*, 48(5):1779–1801, 1993a.

Lawrence R Glosten, Ravi Jagannathan, and David E Runkle. On the relation between the expected value and the volatility of the nominal excess return on stocks. *The journal of finance*, 48(5):1779–1801, 1993b.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.

Arthur Gretton, Alex Smola, Jiayuan Huang, Marcel Schmittfull, Karsten Borgwardt, Bernhard Schölkopf, et al. Covariate shift by kernel mean matching. *Dataset shift in machine learning*, 3(4):5, 2009.

Patrick S Hagan, Deep Kumar, Andrew S Lesniewski, and Diana E Woodward. Managing smile risk. *The Best of Wilmott*, 1:249–296, 2002.

Ehsan Hajizadeh, Abbas Seifi, MH Fazel Zarandi, and IB Turksen. A hybrid modeling approach for forecasting the volatility of s&p 500 index return. *Expert Systems with Applications*, 39(1):431–436, 2012.

Shaikh A Hamid and Zahid Iqbal. Using neural networks for forecasting volatility of s&p 500 index futures prices. *Journal of Business Research*, 57(10):1116–1125, 2004.

James D Hamilton. *Time series analysis*. Princeton university press, 2020.

Peter R Hansen and Asger Lunde. A forecast comparison of volatility models: does anything beat a garch (1, 1)? *Journal of applied econometrics*, 20(7):873–889, 2005.

Peter R Hansen and Asger Lunde. Realized variance and market microstructure noise. *Journal of Business & Economic Statistics*, 24(2):127–161, 2006.

Peter R Hansen, Asger Lunde, and James M Nason. The model confidence set. *Econometrica*, 79(2):453–497, 2011.

Andrew Harvey, Esther Ruiz, and Neil Shephard. Multivariate stochastic variance models. *The Review of Economic Studies*, 61(2):247–264, 1994.

David Harvey, Stephen Leybourne, and Paul Newbold. Testing the equality of prediction mean squared errors. *International Journal of forecasting*, 13(2):281–291, 1997.

Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

Darryll Hendricks. Evaluation of value-at-risk models using historical data. *Economic policy review*, 2(1), 1996.

Steven L Heston. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *The review of financial studies*, 6(2): 327–343, 1993.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997a.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997b.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997c.

Yan Hu, Jian Ni, and Liu Wen. A hybrid deep learning approach by integrating lstm-ann networks with garch model for copper price volatility prediction. *Physica A: Statistical Mechanics and its Applications*, 557:124907, 2020.

D. S. Huang, X. P. Zhang, and Y. L. Zhu. *Regression and Econometrics*. China Financial and Economic Publishing House, 1970.

John Hull and Alan White. The pricing of options on assets with stochastic volatilities. *The journal of finance*, 42(2):281–300, 1987a.

John Hull and Alan White. The pricing of options on assets with stochastic volatilities. *The journal of finance*, 42(2):281–300, 1987b.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*, pages 507–523. Springer, 2011.

Christian Igel and Michael Hüsken. Empirical evaluation of the improved rprop learning algorithms. *Neurocomputing*, 50:105–123, 2003.

Eric Jacquier, Nicholas G Polson, and Peter E Rossi. Bayesian analysis of stochastic volatility models. *Journal of Business & Economic Statistics*, 20(1):69–87, 2002.

Robert Jarrow, Philip Protter, et al. A short history of stochastic integration and mathematical finance: The early years, 1880–1970. In *A festschrift for Herman Rubin*, pages 75–91. Institute of Mathematical Statistics, 2004.

Philippe Jorion. Value at risk. -, 2000.

Philippe Jorion. *Value at risk: the new benchmark for managing financial risk*. The McGraw-Hill Companies, Inc., 2007.

Philippe Jorion et al. *Financial risk manager handbook*, volume 406. John Wiley & Sons, 2007.

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *International conference on machine learning*, pages 2342–2350. PMLR, 2015.

Ken-ichi Kamijo and Tetsuji Tanigawa. Stock price pattern recognition-a recurrent neural network approach. In *1990 IJCNN international joint conference on neural networks*, pages 215–221. IEEE, 1990.

Bekir Karlik and A Vehbi Olgac. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, 1(4):111–122, 2011.

Gregor Kastner and Sylvia Frühwirth-Schnatter. Ancillarity-sufficiency interweaving strategy (asis) for boosting mcmc estimation of stochastic volatility models. *Computational Statistics & Data Analysis*, 76:408–423, 2014.

Md Ashraful Islam Khan. Financial volatility forecasting by nonlinear support vector machine heterogeneous autoregressive model: evidence from nikkei 225 stock index. *International Journal of Economics and Finance*, 3(4):138, 2011.

Sangjoon Kim, Neil Shephard, and Siddhartha Chib. Stochastic volatility: likelihood inference and comparison with arch models. *The review of economic studies*, 65(3): 361–393, 1998.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Christopher Krauss, Xuan Anh Do, and Nicolas Huck. Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the s&p 500. *European Journal of Operational Research*, 259(2):689–702, 2017.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. volume 25, 2012.

Nikolay Laptev, Jason Yosinski, Li Erran Li, and Slawek Smyl. Time-series extreme event forecasting with neural networks at uber. In *International conference on machine learning*, volume 34, pages 1–5, 2017.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553): 436–444, 2015.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Rui Luo, Weinan Zhang, Xiaojun Xu, and Jun Wang. A neural stochastic volatility model. In *proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Leandro Maciel, Fernando Gomide, and Rosangela Ballini. Evolving fuzzy-garch approach for financial volatility modeling and forecasting. *Computational Economics*, 48:379–398, 2016.

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International conference on machine learning*, pages 2113–2122. PMLR, 2015.

John M Maheu and Thomas H McCurdy. News arrival, jump dynamics, and volatility components for individual stock returns. *The Journal of Finance*, 59(2):755–793, 2004.

Cecilia Mancini. Non-parametric threshold estimation for models with stochastic diffusion coefficient and jumps. *Scandinavian Journal of Statistics*, 36(2):270–296, 2009.

Benoit Mandelbrot. The variation of some other speculative prices. *The Journal of Business*, 40(4):393–413, 1967.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, pages 1045–1048. Makuhari, 2010.

Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5528–5531. IEEE, 2011.

Jacob A Mincer and Victor Zarnowitz. The evaluation of economic forecasts. In *Economic forecasts and expectations: Analysis of forecasting behavior and performance*, pages 3–46. NBER, 1969.

Ryotaro Miura, Lukáš Pichl, and Taisei Kaizoji. Artificial neural networks for realized volatility prediction in cryptocurrency time series. In *Advances in Neural Networks–ISNN 2019: 16th International Symposium on Neural Networks, ISNN 2019, Moscow, Russia, July 10–12, 2019, Proceedings, Part I 16*, pages 165–172. Springer, 2019.

Jose G Moreno-Torres, Troy Raeder, Rocío Alaiz-Rodríguez, Nitesh V Chawla, and Francisco Herrera. A unifying view on dataset shift in classification. *Pattern recognition*, 45(1):521–530, 2012.

Ulrich A Müller, Michel M Dacorogna, Rakhal D Davé, Olivier V Pictet, Richard B Olsen, and J Robert Ward. Fractals and intrinsic time: A challenge to econometricians. *Unpublished manuscript, Olsen & Associates, Zürich*, page 130, 1993.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

Daniel B Nelson. Stationarity and persistence in the garch (1, 1) model. *Econometric theory*, 6(3):318–334, 1990.

Daniel B Nelson. Conditional heteroskedasticity in asset returns: A new approach. *Econometrica: Journal of the econometric society*, pages 347–370, 1991.

Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2003.

T-N Nguyen, M-N Tran, and R Kohn. Recurrent conditional heteroskedasticity. *arXiv preprint arXiv:2010.13061*, 2020.

T-N Nguyen, M-N Tran, D Gunawan, and R Kohn. A statistical recurrent stochastic volatility model for stock markets. *Journal of Business & Economic Statistics*, (just-accepted):1–40, 2022.

Maureen O'hara. *Market microstructure theory*. John Wiley & Sons, 1998.

Christopher Olah. Understanding lstm networks. *https://colah.github.io/posts/2015-08-Understanding-LSTMs/*, 2015.

Junier B Oliva, Barnabás Póczos, and Jeff Schneider. The statistical recurrent unit. In *International Conference on Machine Learning*, pages 2671–2680. PMLR, 2017.

Leonard Salomon Ornstein. On the theory of the brownian motion. *Physical review*, 36:823–841, 1930.

Adrian Pagan. The econometrics of financial markets. *Journal of empirical finance*, 3 (1):15–102, 1996.

Gaurang Panchal, Amit Ganatra, YP Kosta, and Devyani Panchal. Searching most efficient neural network architecture using akaike's information criterion (aic). *International Journal of Computer Applications*, 1(5):41–44, 2010.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. Pmlr, 2013.

Andrew J Patton. Volatility forecast comparison using imperfect volatility proxies. *Journal of Econometrics*, 160(1):246–256, 2011.

Andrew J Patton and Kevin Sheppard. Evaluating volatility and correlation forecasts. In *Handbook of financial time series*, pages 801–838. Springer, 2009.

Andrew J Patton and Kevin Sheppard. Good volatility, bad volatility: Signed jumps and the persistence of volatility. *Review of Economics and Statistics*, 97(3):683–697, 2015.

Efthymios Pavlidis, Ivan Paya, David Peel, et al. A new test for rational speculative bubbles using forward exchange rates: The case of the interwar german hyperinflation. *Department of Economics, Lancaster university Management school, UK*, 2012.

Christophe Pérignon, Zi Yin Deng, and Zhi Jun Wang. Do banks overstate their value-at-risk? *Journal of Banking & Finance*, 32(5):783–794, 2008.

Dimitris N Politis and Joseph P Romano. The stationary bootstrap. *Journal of the American Statistical association*, 89(428):1303–1313, 1994.

Ser-Huang Poon. *A practical guide to forecasting financial market volatility*. John Wiley & Sons, 2005.

Ser-Huang Poon and Clive WJ Granger. Forecasting volatility in financial markets: A review. *Journal of economic literature*, 41(2):478–539, 2003.

Joaquin Quinonero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence. *Dataset shift in machine learning*. Mit Press, 2008.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286. PMLR, 2014.

R Tyrrell Rockafellar and Stanislav Uryasev. Conditional value-at-risk for general loss distributions. *Journal of banking & finance*, 26(7):1443–1471, 2002.

German Rodikov and Nino Antulov-Fantulin. Can lstm outperform volatility-econometric models? *arXiv preprint arXiv:2202.11581*, 2022.

German Rodikov and Nino Antulov-Fantulin. Introducing the σ-cell: Unifying garch, stochastic fluctuations and evolving mechanisms in rnn-based volatility forecasting. *arXiv preprint arXiv:2309.01565*, 2023.

Raul Rosa, Leandro Maciel, Fernando Gomide, and Rosangela Ballini. Evolving hybrid neural fuzzy network for realized volatility forecasting with jumps. In *2014 IEEE Conference on Computational Intelligence for Financial Engineering & Economics (CIFEr)*, pages 481–488. IEEE, 2014.

S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Gleb Sandmann and Siem Jan Koopman. Estimation of stochastic volatility models via monte carlo maximum likelihood. *Journal of Econometrics*, 87(2):271–301, 1998.

Stephen Satchell and John Knight. *Forecasting volatility in the financial markets*. Elsevier, 2011.

Neil Shephard. *Stochastic volatility: selected readings*. OUP Oxford, 2005.

Steven E Shreve et al. *Stochastic calculus for finance II: Continuous-time models*, volume 11. Springer, 2004.

Sima Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. The performance of lstm and bilstm in forecasting time series. In *2019 IEEE International conference on big data (Big Data)*, pages 3285–3292. IEEE, 2019.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

Xuanyi Song, Yuetian Liu, Liang Xue, Jun Wang, Jingzhe Zhang, Junqiang Wang, Long Jiang, and Ziyan Cheng. Time-series well performance prediction based on

long short-term memory (lstm) neural network model. *Journal of Petroleum Science and Engineering*, 186:106682, 2020.

Ilya Sutskever. *Training recurrent neural networks*. University of Toronto Toronto, ON, Canada, 2013.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

SJ Taylor. Financial returns modelled by the product of two stochastic processes, a study of daily sugar prices 1961-1979. time series analysis: Theory and practice 1. anderson od, 1982.

Chris Tofallis. A better measure of relative prediction accuracy for model selection and model estimation. *Journal of the Operational Research Society*, 66:1352–1362, 2015.

Ruey S Tsay. *Analysis of financial time series*. John wiley & sons, 2005.

Dick Van Dijk and Philip Hans Franses. Selecting a nonlinear time series model using weighted tests of equal forecast accuracy. *Oxford Bulletin of Economics and Statistics*, 65:727–744, 2003.

Dimitrios I Vortelinos. Forecasting realized volatility: Har against principal components combining, neural networks and garch. *Research in international business and finance*, 39:824–839, 2017.

Ronald J Williams and Jing Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural computation*, 2(4):490–501, 1990.

Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. *Advances in neural information processing systems*, 30, 2017.

Yue Wu, José Miguel Hernández-Lobato, and Ghahramani Zoubin. Dynamic covariance models for multivariate financial time series. pages 558–566, 2013.

Jean-Michel Zakoian. Threshold heteroskedastic models. *Journal of Economic Dynamics and control*, 18(5):931–955, 1994.

Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Aston Zhang, Zachary C Lipton, Mu Li, and Alexander J Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.

Zeyu Zheng, Zhi Qiao, Tetsuya Takaishi, H Eugene Stanley, and Baowen Li. Realized volatility and absolute return volatility: a comparison indicating market risk. *PloS one*, 9(7):e102940, 2014.